



**HAL**  
open science

## Analysis of Self-\* and P2P Systems using Refinement (Full Report)

Manamiary Bruno Andriamiarina, Dominique Méry, Neeraj Kumar Singh

► **To cite this version:**

Manamiary Bruno Andriamiarina, Dominique Méry, Neeraj Kumar Singh. Analysis of Self-\* and P2P Systems using Refinement (Full Report). [Research Report] 2014. hal-01018162

**HAL Id: hal-01018162**

**<https://inria.hal.science/hal-01018162>**

Submitted on 3 Jul 2014

**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.

# Analysis of Self- $\star$ and P2P Systems using Refinement $\star$

Manamiary Bruno Andriamiarina<sup>1</sup>, Dominique Méry<sup>1</sup>, and Neeraj Kumar Singh<sup>2</sup>

Université de Lorraine, LORIA, BP 239, 54506 Vandœuvre-lès-Nancy, France

{Manamiary.Andriamiarina, Dominique.Mery}@loria.fr

McMaster Centre for Software Certification, McMaster University, Hamilton, Ontario, Canada

singhn10@mcmaster.ca, Neerajkumar.Singh@loria.fr

**Abstract.** Distributed systems and applications require efficient and effective techniques (e.g. self-(re)configuration, self-healing, etc.) for ensuring safety, security and more generally dependability properties, as well as *convergence*. The complexity of these systems is increased by features like dynamic (changing) topology, interconnection of heterogeneous components or failures detection. This paper presents a methodology for verifying protocols and satisfying safety and convergence requirements of the distributed self- $\star$  systems. The self- $\star$  systems are based on the idea of managing complex infrastructures, software, and distributed systems, with or without minimal user interactions. *Correct-by-construction* and *service-as-event* paradigms are used for formalizing the system requirements, where the formalization process is based on incremental refinement in EVENT B. Moreover, this paper describes a fully mechanized proof of correctness of the self- $\star$  systems along with an interesting case study related to the P2P-based self-healing protocol.

**Keywords:** Distributed systems, self- $\star$ , self-healing, self-stabilization, P2P, EVENT B, liveness, *service-as-event*

## 1 Introduction

Nowadays, our daily lives are affected by various advanced technologies including computers, chips, and smart-phones. These technologies are integrated into distributed systems with different types of complexities like mobility, heterogeneity, security, fault-tolerance, and dependability. Distributed systems are largely used in many applications and provide required functionalities from the interactions between a large collection of possibly heterogeneous and mobile components (nodes and/or agents). Within the domain of distributed computing, there is an increasing interest in the self-stabilizing systems, which are able to autonomically recover from occurring the faults [7]. The autonomous property of the self- $\star$  systems tends to take a growing importance in the analysis and development of distributed systems. It is an imperative that we need to get a better understanding of the self- $\star$  systems (emergent behaviours, interactions between agents, etc.), if we want to reason about their security, correctness and trustworthiness.

---

$\star$  The current report is the companion paper of the paper [4] accepted for publication in the volume 8477 of the serie Lecture Notes in Computer Science. The Event-B models are available at the link <http://eb2all.loria.fr>. Processed on July 3, 2014.

Fortunately, the formal methods community has been analysing a similar class of systems for years, namely distributed algorithms.

In this study, we use the *correct by construction* approach [12] for modelling the distributed self- $\star$  systems. Moreover, we also emphasize the use of the *service-as-event* [3] paradigm, that identifies the phases of *self-stabilization* mechanism, which can be simplify into more stable and simple coordinated steps.

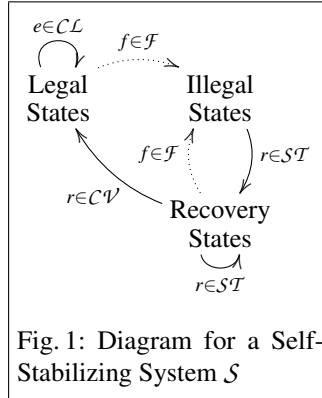


Fig. 1: Diagram for a Self-Stabilizing System  $\mathcal{S}$

We consider that a given system  $\mathcal{S}$  (see in Fig.1) is characterized by a set of *events* (*procedures* modelling either phases or basic actions according to an abstraction level) that modifies the state of the system. *Legal states* (correct states) satisfying a *safety property*  $P$  are defined by a subset  $CL$  of possible events of the system  $\mathcal{S}$ . The events of  $CL$  represent the possible big or small computation steps of the system  $\mathcal{S}$  and introduce the notion of *closure* [5], where any computation starting from a *legal state* satisfying the *property*  $P$  leads to another *legal state* that also satisfies the *property*  $P$ . The occurrence of a fault  $f$  leads the system  $\mathcal{S}$  into an *illegal state* (incorrect state), which violates the *property*  $P$ . The fault  $f$  is defined as an

event  $f$  that belongs to a subset  $\mathcal{F}$  of events. When considering the hypothesis of having a self- $\star$  system, we assume that there are *procedures* (*protocols* or *actions*) which implement the identification of *current illegal states* and recovery for *legal states*. There is a subset  $\mathcal{ST}$  of events modelling *recovery* phases for demonstrating the *stabilization* process. The system recovers using a finite number of *stabilization* steps ( $r$ ). The process is modelled as an event  $r$  of  $\mathcal{CV}(\subseteq \mathcal{ST})$  eventually leading to the *legal states* (*convergence* property) from *recovery* states. During the *recovery* phase, a fault may occur (see dotted transitions in Fig.1).

The system  $\mathcal{S}$  can be represented by a set of events  $\mathcal{M} = CL \cup \mathcal{ST} \cup \mathcal{F}$ , where the model  $\mathcal{M}$  contains a set ( $CL$ ) of events for representing the *computation steps* of the system  $\mathcal{S}$ . When a fault occurs, a set ( $\mathcal{ST}$ ) of events *simulates* the *stabilization* process that is performed by  $\mathcal{S}$ . The formal representation expresses a *closed* model but we do not know what is the complete set of events modelling faults/failures. We characterise the fault model in a very abstract way and it may be possible to develop the fault model according to the assumptions on the environment, but we do not consider this in the current study. We restrict our study by making explicit the events of  $\mathcal{ST}$  modelling the *stabilization* of the system from illegal/failed states. We ensure that the *convergence* is always possible: a subset  $\mathcal{CV}$  of  $\mathcal{ST}$  eventually leads  $\mathcal{S}$  into the *legal states* satisfying the invariant  $P$  of the system. Whenever the system  $\mathcal{S}$  is in a *legal state*, we consider that the events of  $\mathcal{ST}$  are either not operative or simply preserve the invariant  $P$  of the system.

In the previous paragraph, we name *procedures* (*protocols* or *actions*) by the term *events*. An *event* is modelling a process which is defined by its pre and post specifications or a state transformation belonging to a larger process. It means that we need to play with abstraction levels to develop a self- $\star$  system. For instance, one can state that an event called *stabilise* is ensuring the functionality of getting a stable system (the *what*) without giving details of the detailed process itself (the *how*). Hence, the notion of

event is identified to an abstraction level and can be either modelling a global process (the *what*) or a local update of a variable (the *how*). We formalise the system  $S$  using the EVENT B modelling language [1], dealing with *events* and *invariant* properties including *convergence* properties by using a temporal framework. The *service-as-event* paradigm [3] helps to express this *concretisation* process: the procedures (1) *leading* from the *illegal states* to the *recovery states*, and (2) *leading* from the *recovery states* to the *legal states* are stated by (abstract) events, during the first stages of the EVENT B development. The next step is to unfold each (abstract procedure) event, by refinement, to a set of coordinated and concrete events, which form the body of the procedure.

This paper is organised as follows. Section 2 presents related works. Section 3 introduces the EVENT B modelling framework including *service-as-event* paradigm and a formal definition of self- $\star$  systems. Section 4 presents the formal verification approach and illustrates the proposed methodology with the study of the self-healing P2P-based protocol [14]. Section 5 discusses on approaches for studying temporal properties for EVENT B models. Finally, Section 6 concludes the paper along with future work.

## 2 Related Works on Formal Modelling for Self- $\star$ Systems

Systems usually run in intricate environments, with frequent and unexpected changes. This feature increases interest towards autonomous and self- $\star$  architectures, as they are able to adapt themselves according to the changes that may occur in the systems (faults, etc.) or in the environment. Applying formal methods to self- $\star$  systems originates from the needs of understanding how these systems behave and how they meet their specifications. A self- $\star$  system relies on *emergent behaviours*, resulting from interactions between components of the system [21].

Traditionally, the correctness of self- $\star$  and autonomous systems is validated through the simulation and testing [20, 22]. However, simulation and testing are not sufficient to cover the whole set of possible states of a system [2]. Therefore, formal methods appear as a promising land for validating self- $\star$  systems, as long as formal techniques can assert the correctness of these systems and certify target properties, like trustworthiness, security, efficiency, etc. under the rigorous mathematical reasoning [6, 8, 24].

Smith et al. [21] have applied the stepwise refinement using Z to study a case of self-reconfiguration, where a set of autonomous robotic agents is able to assemble and to reach a global shape. They do not validate models using an adequate tool (e.g. proof checker, proof assistant, etc.) and models are not localized. Calinescu et al. [6] have used Alloy to demonstrate the correctness of the autonomic computing policies (ACP). However, Alloy does not provide a mechanism for expressing the *correct-by-construction* paradigm. Méry et al. [2] have also investigated a self-reconfiguring system (Network-on-Chip: adaptative XY routing) using the EVENT B framework and the *correct-by-construction* approach.

State exploration approaches such as model-checking are also used to study self- $\star$  systems. Model-checkers like SPIN, PRISM, SMV, UPPAAL are used for properties specification and getting evidences that properties, such as flexibility, robustness of the self- $\star$  systems hold [6, 8, 10, 24]. Moreover, these tools allow users to obtain the metrics

for the self- $\star$  systems, such as performance, and quantitative evaluations [6, 8, 10, 24]. Model-checking and state-space evaluation can be used during the conception of self- $\star$  systems, but they are especially used for runtime verification [10, 24]. The limit of model checking is clearly the size of models.

Other formal techniques like static analysis and design by contract are also applied for the formal specification of self- $\star$  systems [23]. These techniques are mainly used for *runtime verification*. Graphical approaches, such as Petri Nets, are used to model the temporal aspects and communication flows between different components of a self- $\star$  system, and helped to study the cases like self-reconfiguration (replacement of a component, removal of a link between two components, etc.) [24].

Finally, graphical notations (e.g. UML) help to represent self- $\star$  systems with understandable figures [25]. Their general purpose is to provide users an insight of a self- $\star$  system by describing its architecture, the relationships between agents of the system (OperA methodology [17], ADELFE [20]) or by presenting the system as a composition of extendable/instantiable primitives (FORMS [25]). These notations are generally graphical front-ends for the more complex representations of self- $\star$  systems, where the source code [20], and formal models [25] can be generated from the notations.

Our proposed methodology integrates the EVENT B method and elements of temporal logics. Using the refinement technique, we gradually build models of self- $\star$  systems in the EVENT B framework. Moreover, we use the *service-as-event* paradigm to describe the *stabilization* and *convergence* from *illegal* states to *legal* ones. Self- $\star$  systems require the expression of traces properties like liveness properties and we borrow a minimal set of inference rules for deriving liveness properties. The concept of *refinement diagrams* intends to capture the intuition of the designer for deriving progressively the target self- $\star$  system. The RODIN platform provides a laboratory for checking, animating and validating the formal models.

### 3 Modelling Framework

#### 3.1 EVENT B

We advocate the use of *correct-by-construction* paradigm for modelling the self- $\star$  systems. The key concept is the incremental refinement (simulation) which provides link between discrete models by preserving properties. The EVENT B modelling language designed by Abrial [1] is based on *set theory* and the *refinement* of models: an abstract model expressing the requirements of a given system can be verified and validated easily; a concrete model corresponding to the actual system is *constructed* progressively by *refining* the abstraction. EVENT B is supported by a complete toolset RODIN [19] providing features like refinement, proof obligations generation, proof assistants and model-checking.

**Modelling Actions over States** The EVENT B modelling language can express *safety properties*, which are either *invariants* or *theorems* in a model corresponding to the system. Two main structures are available in EVENT B : **(1)** Contexts express static

informations about the model (for instance, graph properties as connectivity); **(2)** Machines express dynamic informations about the model, safety properties, and events. An EVENT B model is defined by a context and a machine. A machine organises events (or actions) modifying state variables and uses static informations defined in a context. An EVENT B model is characterised by a (finite) list  $x$  of *state variables* possibly modified by a (finite) list of *events*. An invariant  $I(x)$  states properties that must always be satisfied by the variables  $x$  and *maintained* by the activation of the events. The general form of an event  $e$  is as follows: ANY  $t$  WHERE  $G(t, x)$  THEN  $x : |P(t, x, x')$  END and corresponds to the transformation of the state of the variable  $x$ , which is described by a *before-after* predicate  $BA(e)(x, x')$ : the predicate is semantically equivalent to  $\exists t \cdot G(t, x) \wedge P(t, x, x')$  and expresses the relationship linking the values of the state variables before ( $x$ ) and just after ( $x'$ ) the *execution* of the event  $e$ . Proof obligations are produced by RODIN, from events: INV1 and INV2 state that an invariant condition  $I(x)$  is preserved; their general form follows immediately from the definition of the before-after predicate  $BA(e)(x, x')$  of each event  $e$ ; FIS expresses the feasibility of an event  $e$ , with respect to the invariant  $I$ . By proving feasibility, we achieve that  $BA(e)(x, z)$  provides a next state whenever the guard  $grd(e)(x)$  holds: the guard is the enabling condition of the event.

INV1	INV2	FIS
$Init(x) \Rightarrow I(x)$	$I(x) \wedge BA(e)(x, x') \Rightarrow I(x')$	$I(x) \wedge grd(e)(x) \Rightarrow \exists z \cdot BA(e)(x, z)$

**Model Refinement** The refinement of models extends the structures described previously, and relates an abstract model and a concrete model. This feature allows us to develop EVENT B models of the self- $\star$  approach gradually and validate each decision step using the proof tool. The refinement relationship is expressed as follows: a model  $AM$  is refined by a model  $CM$ , when  $CM$  *simulates*  $AM$  (i.e. when a concrete event  $ce$  occurs in  $CM$ , there must be a corresponding enabling abstract event  $ae$  in  $AM$ ). The final concrete model is closer to the behaviour of a real system that observes events using real source code. The relationships between contexts, machines and events are illustrated by the following diagrams (Fig. 2), which consider refinements of events and machines.

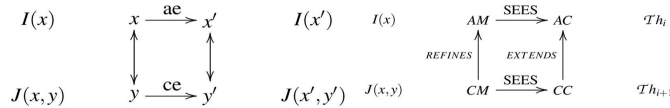


Fig. 2: Machines and Contexts relationships

The refinement of a formal model allows us to enrich the model via a *step-by-step* approach and is the foundation of our *correct-by-construction* approach [12]. Refinement provides a way to strengthen invariants and to add details to a model. It is also used to transform an abstract model to a more concrete version by modifying the state description. This is done by extending the list of state variables (possibly suppressing some of them), by refining each abstract event to a set of possible concrete versions, and by adding new events. We suppose (see Fig. 2) that an abstract model  $AM$  with variables  $x$  and an invariant  $I(x)$  is refined by a concrete model  $CM$  with variables  $y$ . The abstract state variables,  $x$ , and the concrete ones,  $y$ , are linked together by means of a, so-called, gluing invariant  $J(x, y)$ . Event  $ae$  is in abstract model  $AM$  and event  $ce$  is in

concrete model  $CM$ . Event  $ce$  refines event  $ae$ .  $BA(ae)(x, x')$  and  $BA(ce)(y, y')$  are predicates of events  $ae$  and  $ce$  respectively; we have to discharge the following proof obligation:  $I(x) \wedge J(x, y) \wedge BA(ce)(y, y') \Rightarrow \exists x' \cdot (BA(ae)(x, x') \wedge J(x', y'))$

Due to limitations on the number of pages, we have briefly introduced the EVENT B modelling language and the structures proposed for organising the formal development. However, more details are available in [1] and on the Internet<sup>1</sup>. In fact, the refinement-based development of EVENT B requires a very careful derivation process, integrating possible *tough* interactive proofs. For assisting the development of the self- $\star$  systems, we use the *service description* and *decomposition* that is provided by the *service-as-event* [3] paradigm (derived from the *call-as-event* approach [15]).

### 3.2 The Service-as-Event Paradigm

This section introduces the *refinement diagrams* [3, 15] and presents the *service-as-event* paradigm. A brief overview on the usage of these formalisms for modelling the self- $\star$  systems is given.

**Objectives** The *service-as-event* paradigm [3, 15] is a semantical extension of EVENT B and introduces a way to deal with liveness properties and traces, for modelling the self- $\star$  systems.

**A Definition of Self- $\star$  Mechanism** We characterize a self-stabilizing system  $\mathcal{S}$  (more generally a self- $\star$  system) by its ability to recover autonomously from an *illegal* (faulty) state (violating the invariant  $P$  of the system) to a *legal* (correct) state satisfying the invariant property  $P$  of system  $\mathcal{S}$ . Temporal logic [3, 11, 15, 18] can be used to describe such mechanism, using the liveness properties: we represent the *stabilization* (especially the *convergence*) property as a *service* where a system  $\mathcal{S}$ , in an *illegal* state (characterized by  $\neg P$ ), reaches *eventually* a *legal* state (satisfying  $P$ ). This service is expressed, with the *leads to* ( $\rightsquigarrow$ ) operator, as follows:  $(\neg P) \rightsquigarrow P$ . This *leads to* property (equivalently  $((\neg P) \Rightarrow \diamond P)$ ) states that every *illegal* state (satisfying  $\neg P$ ) will *eventually* (at some point in the future) lead to a *legal* state (satisfying  $P$ ).

We define a temporal framework for the EVENT B model  $M$  of the studied system  $\mathcal{S}$  by the following TLA specification:  $Spec(M): Init(y) \wedge \square[Next]_y \wedge L$ , where  $Init(y)$  is the predicate specifying initial states;  $Next \equiv \exists e \in E. BA(e)(y, y')$  is an action formula representing the next-state relation; and  $L$  is a conjunction of formulas  $WF_y(e)$ : we express a *weak fairness* assumption over each event  $e$  modelling a step of the recovery process (we do not add any fairness on events leading to *illegal states* (*faults*)).

<sup>1</sup> [http://lfm.iti.kit.edu/download/EventB\\_Summary.pdf](http://lfm.iti.kit.edu/download/EventB_Summary.pdf)

**Refinement Diagrams** We express the self- $\star$  mechanism using EVENT B , together

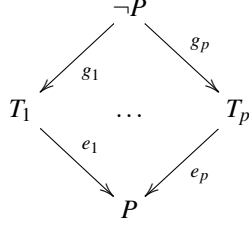


Fig. 3: A Refinement Diagram

with liveness properties under fairness assumptions. *Refinement diagrams* (see in Fig.3), introduced by Méry et al in [3, 15], allow to develop EVENT B models and add control inside these models. They are also used for stating (proofs of) liveness properties (under fairness assumptions), and for supporting refinement. Therefore, these diagrams are suitable for representing the models of self- $\star$  systems. A *refinement diagram*  $D \hat{=} PD(M)$  for a machine  $M$  is defined as follows:  $PD(M) = (A, M, G, E)$ , where  $A$  is a set of assertions,  $G$  a set of assertions for  $M$  called conditions/guards of the form  $g(x)$ ,  $E$  is the set of events of  $M$ . The diagram  $PD(M)$  is a labelled directed graph over  $A$ , with labels from  $G$  or  $E$ , satisfying the following rules: (1) if an assertion  $R$  is related to another assertion  $S$ , by an unique *non-dotted* arrow labelled  $e \in E$  (where  $e$  does not model a fault), then the property  $R \rightsquigarrow S$  is satisfied; (2) if  $R$  is related to  $S_1, \dots, S_p$ , then each arrow from  $R$  to  $S_i$  is labelled by a guard  $g_i \in G$ . The diagram  $D$  possesses proved properties:

1. If  $M$  satisfies  $P \rightsquigarrow Q$  and  $Q \rightsquigarrow R$ , it satisfies  $P \rightsquigarrow R$ .
2. If  $M$  satisfies  $P \rightsquigarrow Q$  and  $R \rightsquigarrow Q$ , it satisfies  $(P \vee R) \rightsquigarrow Q$ .
3. If  $I$  is invariant for  $M$  and if  $M$  satisfies  $P \wedge I \rightsquigarrow Q$ , then  $M$  satisfies  $P \rightsquigarrow Q$ .
4. If  $I$  is invariant for  $M$  and if  $M$  satisfies  $P \wedge I \Rightarrow Q$ , then  $M$  satisfies  $P \rightsquigarrow Q$ .
5. If  $P \xrightarrow{e} Q$  is a link of  $D$  for the machine  $M$ , then  $M$  satisfies  $P \rightsquigarrow Q$ .
6. If  $P$  and  $Q$  are two nodes of  $D$  such that there is a path in  $D$  from  $P$  to  $Q$  and any path from  $P$  can be extended in a path containing  $Q$ , then  $M$  satisfies  $P \rightsquigarrow Q$ .
7. If  $I, U, V, P, Q$  are assertions such that  $I$  is the invariant of  $M$ ;  $P \wedge I \Rightarrow U$ ;  $V \Rightarrow Q$ ; and there is a path from  $U$  to  $V$  and each path from  $U$  leads to  $V$ ; then  $M$  satisfies  $P \rightsquigarrow Q$ .

These *refinement diagrams* are attached to EVENT B models and are used for deriving liveness properties. As an example, the diagram in Fig.3 represents a model of a self-stabilizing system: the diagram relates a pair of assertions  $(\neg P, P)$ , where  $\neg P$  is a precondition stating that the studied system is in an *illegal* state ( $P$  does not hold); and  $P$  is the post-condition, describing the *desired legal* state. We observe that the *leads to* property  $(\neg P) \rightsquigarrow P$ , demonstrating the stabilization and convergence, is satisfied by the diagram and the model linked to it.

**Applying the Service-as-Event Paradigm [3]** We apply the *service-as-event* paradigm, for formalizing the self- $\star$  systems.

1. **Describing stabilization and convergence as a service.** We express the *stabilization* and *convergence* properties of a self- $\star$  system  $S$ , where service is stated by the following property:  $(\neg P) \rightsquigarrow P$ . An abstract event ( $e$ ) is used for describing the service/procedure represented by  $(\neg P) \rightsquigarrow P$ :  $(\neg P) \xrightarrow{e} P$ ; where  $(\neg P)$  is a *pre-condition* for triggering event ( $e$ ); and  $P$  is a *post-condition* defined by the actions of event ( $e$ ), which should be satisfied by the "execution" of event.

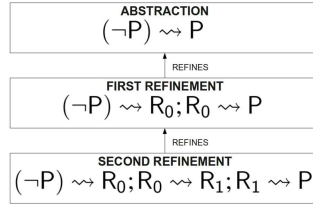


2. **Decomposing stabilization and convergence into simple steps.** We decompose the abstract service stated by  $(\neg P) \rightsquigarrow P$  into simple *sub-procedures/steps*, using the *inference rules* [11] related to the *leads to* properties:

$$\frac{\frac{\dots}{(\neg P) \rightsquigarrow R_0} \text{trans}_2 \quad \frac{\dots}{R_0 \rightsquigarrow R_1} \text{trans}_4 \quad \frac{\dots}{R_1 \rightsquigarrow P} \text{trans}_5}{R_0 \rightsquigarrow P} \text{trans}_3}{(\neg P) \rightsquigarrow P} \text{trans}_1$$

Fig. 4: Proof Tree - Usage of Inference Rules

This process is similar to refinement (see Fig.5), since we add, at each level of the proof tree, a new state  $R_k$  ( $0 \leq k \leq n$ ) leading from  $(\neg P)$  to  $P$ . The initial property  $(\neg P) \rightsquigarrow P$  is decomposed, until the identification of the *stabilization* steps is satisfactory. The *stabilization* phase is expressed by the property



$(\neg P) \rightsquigarrow R_0 \wedge R_0 \rightsquigarrow R_1 \wedge \dots \wedge R_{n-1} \rightsquigarrow R_n \wedge R_n \rightsquigarrow P$ , which states the *convergence* leading to the *desired legal* states. Each level of the proof tree corresponds to a level of refinement (see Fig.5) in the formal development. Each *leads to* property demonstrates a *service* of

Fig. 5: Decomposition and Refinement property demonstrates a *service* of *stabilization*, which is defined by an event in the model.

## 4 Stepwise Design of the Self-Healing Approach

### 4.1 Introduction to the Self-Healing P2P-Based Approach

The development of *self-healing P2P-based approach* is proposed by Marquezan et al. [14], where *system reliability* is the main concern. The self-healing process ensures the maintenance of proper functioning of the system services. If a service fails then it switches from a *legal* state to a *faulty* state. The self-healing/recovery procedure ensures that the service switches back to the *legal* state. The services run in a distributed (P2P) system composed of *agents/peers* executing instances of tasks. The *services* and *peers* notions are introduced as: **(1) Management Services:** Tasks/Services are executed by the peers; **(2) Instances of Management Services:** Peers executing a certain type of management service; **(3) Management Peer Group (MPG):** Instances of the same management service. The self-healing property can be described as follows: **(1) Self-identification** triggers to detect the failure of service. This mechanism identifies running or failed instances of a management service. **(2) Self-activation** is started, whenever a management service will be detected fail by the *self-identification*. *Self-activation* evaluates if the management service needs a recovery, based on the criticality of the failure: if there are still enough instances for running the service, the recovery procedure is not started; otherwise, the *self-configuration* mechanism is triggered for repairing the service. **(3) Self-configuration** is activated if the failure of service is critical: the role of this mechanism is to instantiate the failed management service, and to return the service into a *legal* state.

## 4.2 The Formal Design

Figure 6 depicts the formal design of *self-healing P2P-based approach*. The model M0 abstracts the self-healing approach. The refinements M1, M2, M3 introduce step-by-step the *self-detection*, *self-activation* and *self-configuration* phases, respectively. The remaining refinements, from M4 to M20, are used for localisation of the system: each step of the algorithm is made *local* to a node. The last refinement M21 presents a local model that describes a set of procedures for recovering process of P2P system.

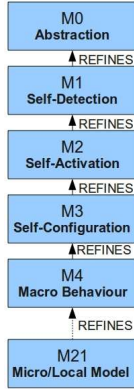
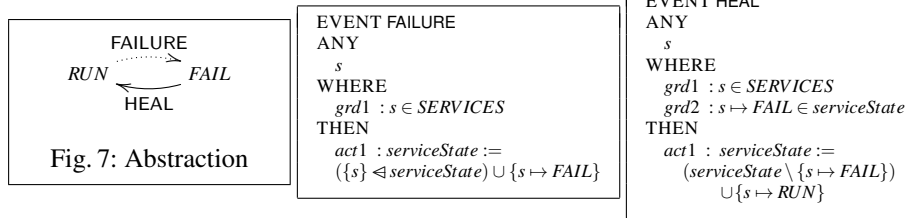


Fig. 6: Architecture

**Abstracting the Self-Healing Approach (M0)** This section presents an abstraction of the self-healing procedure for a failed service. Each service ( $s$ ) is described by two states: *RUN* (*legal/running* state) and *FAIL* (*illegal/faulty* state). A variable *serviceState* is defined as  $s \mapsto st \in \text{serviceState}$ , where  $s$  denotes a service and  $st$  denotes a possible state. A property  $P$  expresses that a service ( $s$ ) is in a *legal running* state that is formalised as  $P \hat{=} (s \mapsto \text{RUN} \in \text{serviceState})$ . An event FAILURE models a faulty behaviour, where service ( $s$ ) enters into a faulty state (*FAIL*), satisfying  $\neg P$ . The *self-healing* of management service ( $s$ ) is expressed as  $(\neg P) \rightsquigarrow P$ . The *recovery* procedure is stated by an event HEAL  $((\neg P) \xrightarrow{\text{HEAL}} P)$ , where service ( $s$ ) recovers from an *illegal faulty* state (*FAIL*) to a *legal running* state (*RUN*). The refinement diagram<sup>1</sup> (see Fig.7) and events sum up the abstraction of a *recovery* procedure.



This *macro/abstract* view of the *self-healing* is detailed by refinement<sup>2</sup>, using intermediate steps. A set of new variables is introduced to capture the system requirements. The variables are denoted by  $NAME\_ \{Refinement\ Level\}$ .

**Introducing the Self-Detection (M1)** The variable *serviceState* is replaced, by refinement, with a new variable *serviceState\_1*, since new states are introduced. The states *RUN*, *FAIL* are refined into *RUN\_1*, *FAIL\_1*, and a new state (*FL\_DT\_1*) is defined. A service ( $s$ ) can *suspect* and *identify* a failure state (*FAIL\_1*) before triggering the recovery (HEAL). We introduce a property  $R_0 \hat{=} (s \mapsto \text{FL\_DT\_1} \in \text{serviceState\_1})$  and a new event FAIL\_DETECT in this *self-detection* mechanism. Let  $P$  and  $\neg P$  be redefined as follows:  $P \hat{=} (s \mapsto \text{RUN\_1} \in \text{serviceState\_1})$  and  $\neg P \hat{=} (s \mapsto \text{FAIL\_1} \in \text{serviceState\_1})$ .

<sup>1</sup> The assertions  $(s \mapsto st \in \text{serviceState})$ , describing the state ( $st$ ) of a service ( $s$ ), are shortened into  $(st)$ , in the nodes of the refinement diagrams, for practical purposes.

<sup>2</sup>  $\oplus$ : to add elements to a model,  $\ominus$ : to remove elements from a model

The intermediate steps of self-detection are introduced according to the refinement diagram (see Fig.8) and proof tree.

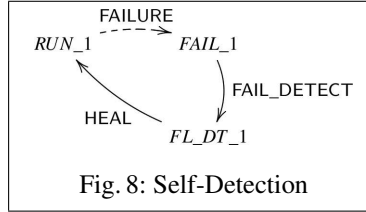


Fig. 8: Self-Detection

$$\frac{(\neg P) \rightsquigarrow R_0 \quad R_0 \rightsquigarrow P}{(\neg P) \rightsquigarrow P} \text{trans}$$

The event FAIL\_DETECT is introduced to express the *self-detection*: the failure state ( $FAIL\_1$ ) of a service ( $s$ ) is detected (state  $FL\_DT\_1$ ).

The property  $(\neg P) \rightsquigarrow R_0$  is expressed by the event FAIL\_DETECT, where the failure ( $FAIL\_1$ ) of service ( $s$ ) is identified ( $FL\_DT\_1$ ).  $R_0 \rightsquigarrow P$  is defined by the event HEAL, where the service ( $s$ ) is restored to a *legal running* state ( $RUN\_1$ ) after failure detection. The same method is applied to identify all the phases of *self-healing* algorithm. Due to limited space, we focus on the interesting parts of models and liveness properties. The complete formal development of models can be downloaded from web<sup>3</sup>.

<pre>EVENT FAILURE REFINES FAILURE ... WHERE ⊕ s ↦ RUN_1 ∈ serviceState_1 THEN ⊖ act1 : ... ⊕ serviceState_1 := (serviceState_1 \ {s ↦ RUN_1}) ∪ {s ↦ FAIL_1}</pre>	<pre>EVENT FAIL_DETECT ANY s WHERE grd1 : s ∈ SERVICES grd2 : s ↦ FAIL_1 ∈ serviceState_1 THEN act1 : serviceState_1 := (serviceState_1 \ {s ↦ FAIL_1}) ∪ {s ↦ FL_DT_1}</pre>	<pre>EVENT HEAL REFINES HEAL ... WHERE ⊖ grd2 ... ⊕ s ↦ FL_DT_1 ∈ serviceState_1 THEN ⊖ act1 ... ⊕ serviceState_1 := (serviceState_1 \ {s ↦ FL_DT_1}) ∪ {s ↦ RUN_1}</pre>
---	---	---

**Introducing the Self-Activation (M2) and Self-Configuration (M3)** The *self-activation* is introduced in this refinement M2 (see Fig. 9), where a failure of a service ( $s$ ) is evaluated in terms of critical or non-critical using a new state  $FL\_ACT\_2$  and an event FAIL\_ACTIV. The *self-configuration* step is introduced in the next refinement M3 (see Fig.10), which expresses that if the failure of service ( $s$ ) is critical, then the *self-configuration* procedure for a service ( $s$ ) will be triggered (state  $FL\_CONF\_3$ ), otherwise, the failure will be ignored (state  $FL\_IGN\_3$ ).

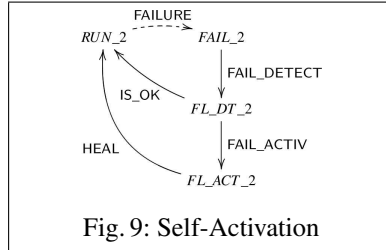


Fig. 9: Self-Activation

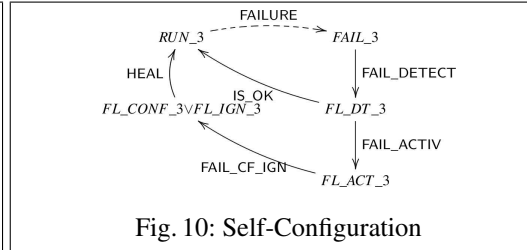


Fig. 10: Self-Configuration

**The Global Behaviour (M4)** The developed models are refined and decomposed into several steps (see Fig.11) [14]. These steps are: **(1) Self-Detection**, **(2) Self-Activation**, and **(3) Self-Configuration**. Self-Detection phase is used to detect any failure in the autonomous system using two events FAIL\_DETECT and IS\_OK. The event FAIL\_DETECT models the failure detection; and the event IS\_OK states that if a detected failure of a service ( $s$ ) is a *false alarm*, then the service ( $s$ ) returns to a *legal* state ( $RUN\_4$ ). Self-Activation process is used to evaluate when actual failures are identified, using

<sup>3</sup> [http://eb2all.loria.fr/html\\_files/files/selfhealing/self-healing.zip](http://eb2all.loria.fr/html_files/files/selfhealing/self-healing.zip)

the following events: FAIL\_ACTIV, FAIL\_IGN, IGNORE, and FAIL\_CONF. The events FAIL\_IGN and IGNORE are used to ignore the failure of service ( $s$ ) when failure is not in critical state ( $FL\_IGN\_4$ ). The event FAIL\_CONF is used to evaluate the failure of service ( $s$ ) when failure is critical ( $FL\_CONF\_4$ ). The last phase *Self-Configuration* presents the healing procedure of a *failed* service using an event REDEPLOY.

From model M5 to M20, we localise the events (we switch from a *service* point of view to the instances/peers point of view) and detail the macro (global) steps by

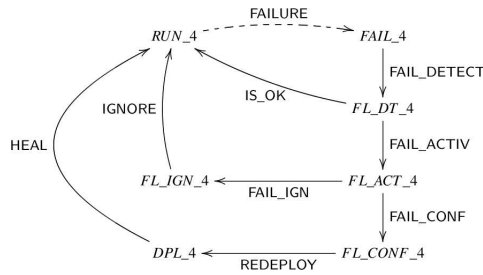


Fig. 11: Self-Healing steps

adding new events, variables, and constraints. The refinements M5, M6, M7 introduce the running ( $run\_peers(s)$ ), faulty ( $fail\_peers[\{s\}]$ ), suspicious ( $susp\_peers(s)$ ) and deployed peers/instances ( $dep\_inst[\{s\}]$ ) for a service ( $s$ ). A function ( $min\_inst$ ) associates each service ( $s$ ) with the minimal number of instances that is required for running service ( $s$ ), and helps to detail the *self-activation* phase: if the number of running instances of service ( $s$ ) is below than minimum, then the failure is critical. The models M8, M9, M10 detail the *self-detection* and *self-configuration* phases to introduce the *token owners* for the services. Models from M11 to M20 localise gradually the events (to switch from a *service* point of view to the instances/peers point of view). The detailed formal development of various steps from M5 to M20 are given in the archive<sup>3</sup>. Due to limited space, in the following section, we present only the local model M21.

**The Local Model (M21)** This model details locally the *self-healing* procedure of a service ( $s$ ). The peers instantiating management service ( $s$ ) are introduced, as well as the notion of *token owner*. The *token owner* is a peer instance of service ( $s$ ) that is marked as a *token owner* for the Management Peer Group (MPG). It can perform the *self-healing* procedure using *self-detection*, *self-activation*, and *self-configuration* steps.

**(1) Self-Detection** introduces an event SUSPECT\_INST that states that the *token owner* for service ( $s$ ) is able to *suspect* a set ( $susp$ ) of unavailable peers instances of service ( $s$ ). Other events RECONTACT\_INST\_OK and RECONTACT\_INST\_KO are used to specify the successful recontact, and failed recontact, respectively, of the unavailable instances for ensuring the failed states. Moreover, the *token owner* is able to monitor the status of service ( $s$ ) using two events FAIL\_DETECT, and IS\_OK. If there are unavailable instances after the recontacting procedure, the *token owner* informs the safe members of MPG of failed instances using the event FAIL\_DETECT, otherwise, the *token owner* indicates that service is running properly. **(2) Self-Activation** introduces an event FAIL\_ACTIV that states that if there are failed instances of service ( $s$ ), then the *token owner* evaluates if the failure is critical. Another event FAIL\_IGNORE specifies that the failure is not critical. An event IGNORE can ignore the failure if several instances (more than minimum) are running correctly. If the number of instances for the running service ( $s$ ) will be less than the minimum required services, then the failure will be declared critical, and the *self-healing* process will be triggered using an event FAIL\_CONFIGURE. **(3) Self-**

**Configuration** introduces three events REDEPLOY\_INSTC, REDEPLOY\_INSTS and REDEPLOY that specify that if the failure of service ( $s$ ) is critical, then new instances of running service ( $s$ ) can be deployed until to reach the minimal number of instances, and after the event HEAL can be triggered corresponding to the *convergence* of the self-healing process.

It is noticeable that the *architectural decomposition* of the self-healing process is emphasized in this model, by the events related to the algorithm. There is also a set of events describing actions related to the environment. MAKE\_PEER\_UNAVAIL: a set of

```

MACHINE M21 ...
EVENT SUSPECT_INST
ANY
  s, susp
WHERE
  grd1 : s ∈ SERVICES
  grd2 : susp ⊆ PEERS
  grd3 : susp = run_inst(token_owner(s) ↦ s) ∩ unav_peers
  grd4 : suspc_inst(token_owner(s) ↦ s) = ∅
  grd5 : inst_state(token_owner(s) ↦ s) = RUN_4
  grd6 : susp ≠ ∅
THEN
  act1 : suspc_inst(token_owner(s) ↦ s) := susp
END
EVENT FAILURE ...
EVENT RECONTACT_INST_OK ...
EVENT RECONTACT_INST_KO ...
EVENT FAIL_DETECT ...
EVENT IS_OK ...
EVENT FAIL_ACTIV ...
EVENT FAIL_IGNORE ...
EVENT IGNORE ...
EVENT FAIL_CONFIGURE ...
EVENT REDEPLOY_INSTC ...
EVENT REDEPLOY_INSTS ...
EVENT REDEPLOY ...
EVENT HEAL ...
EVENT MAKE_PEER_UNAVAIL ...
EVENT UNFAIL_PEER ...
EVENT MAKE_PEER_AVAIL ...

```

peers ( $prs$ ) becomes unavailable (can not be contacted); MAKE\_PEER\_AVAIL: a formerly unavailable instance ( $p$ ) becomes available; UNFAIL\_PEER: a failed instance re-enters a *legal running state*.

This model M21 describes locally the *Self-Healing P2P-Based Approach*, where we have formulated *hypotheses* for ensuring the correct functioning of the self-healing process: (1) *Event MAKE\_PEER\_AVAIL: If the token owner of a service ( $s$ ) becomes unavailable, at least one peer, with the same characteristics as the disabled token owner (state, local informations about running, failed peers, etc.) can become the new token*

*owner of service ( $s$ ); (2) Event REDEPLOY\_INSTC: There is always a sufficient number of available peers that can be deployed to reach the legal running state of a service ( $s$ ).*

In a nutshell, we say that our methodology allows users to understand the self- $\star$  mechanisms and to gain insight into their architectures (components, coordination, etc.); and gives evidences of the correctness of self- $\star$  systems under some assumptions/hypotheses.

## 5 Analysis of Temporal Properties for Event-B Models

Leuschel et al. [13] developed a tool ProB for animating, model-checking, and verifying the consistency of Event-B models. ProB provides two ways for analysing Event-B models : constraint-based checking and temporal model-checking. We focus on temporal model-checking, since we are interested in liveness properties. Temporal model-checking [13] allows ProB to detect problems with a model (invariants violation, deadlocks, etc.) and to verify if the model satisfies LTL properties: ProB explores the state space of the model and tries to find a counter-example (i.e. a sequence of events) leading to the violation of invariants or LTL properties.

A difference with TLC (model-checker for TLA<sup>+</sup>) is that ProB does not support *fairness* [9], allowing unfair traces to be analysed during model-checking. Therefore, the TLA<sup>+</sup> framework is more suited to our work, since we are verifying liveness properties, in Event-B models, under fairness assumptions.

## 6 Discussion, Conclusion and Future Work

We present a methodology based on liveness properties and *refinement diagrams* for modelling the self- $\star$  systems using EVENT B. We characterize the self- $\star$  systems by three modes (abstract states): 1) *legal (correct)* state, 2) *illegal (faulty)* state, and 3) *recovery* state. We have proposed a generic pattern for deriving correct self- $\star$  systems (see Fig.1). The *service-as-event* and *call-as-event* paradigms provide a way to express the relationships between modes for ensuring required properties as convergence. The *correct-by-construction* principle gives us the possibility to refine procedures from events and to link modes. The key idea is to identify the modes (considered as abstract states) and the required abstract steps to allow the navigation between modes, and then to gradually enrich abstract models, using refinement to introduce the concrete states and events. We have illustrated our methodology by the *self-healing approach* [14].

The complexity of the development is measured by the number of proof obligations (PO) which are automatically/manually discharged (see Table 1). It should be noted that a large majority ( $\sim 70\%$ ) of the 1177 manual proofs is solved by simply running the provers. The actual summary of proof obligations is given by Table 2. The manually discharged POs (327) require analysis and skills: searching and adding premises, simplifying the complex predicates, and even transforming goals are needed to discharge these POs. Examples of difficult POs are related to proving the *finiteness* of *Management Peer Groups (MPG)*, during the *redeployment operation of the self-configuration phase*.

Model	Total	Auto	Interactive
CONTEXTS	30	26	86.67%
M0	3	3	100%
M1	21	15	71.4%
M2	46	39	84.8%
M3	68	0	0%
M4	142	16	11.27%
M5	46	17	39.95%
OTHER MACHINES	1065	141	12.44%
M21	13	0	0%
TOTAL	1434	257	17.9%

Table 1: Summary of Proof Obligations

Total	Auto	Quasi-Auto	Manual
1434	257	17.9%	850
			59.3%
			327
			22.8%

Table 2: Synthesis of POs

Furthermore, our refinement-based formalization allows us to produce final local models close to the *source code*. Our future works include the development of techniques for generating applications from the resulting model extending tools like EB2ALL [16]. Moreover, further case studies will help us to discover new patterns; these patterns will be added to a catalogue of patterns that could be implemented in the Rodin platform. Finally, another point would be to take into account dependability properties in our methodology.

## References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. M. B. Andriamiarina, H. Daoud, M. Belarbi, D. Méry, and C. Tanougast. Formal Verification of Fault Tolerant NoC-based Architecture. In *First International Workshop on Mathematics and Computer Science (IWMCS2012)*, Tiaret, Algérie, Dec. 2012.
3. M. B. Andriamiarina, D. Méry, and N. K. Singh. Integrating proved state-based models for constructing correct distributed algorithms. In E. B. Johnsen and L. Petre, editors, *IFM*, volume 7940 of *Lecture Notes in Computer Science*, pages 268–284. Springer, 2013.

4. M. B. Andriamarina, D. Méry, and N. K. Singh. Analysis of self-\* and p2p systems using refinement. In Y. A. Ameur and K.-D. Schewe, editors, *ABZ*, volume 8477 of *Lecture Notes in Computer Science*, pages 117–123. Springer, 2014.
5. A. Berns and S. Ghosh. Dissecting self-\* properties. In *Proceedings of the 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '09, pages 10–19, Washington, DC, USA, 2009. IEEE Computer Society.
6. R. Calinescu, S. Kikuchi, and M. Kwiatkowska. Formal methods for the development and verification of autonomic it systems. In *Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification*, IGI Global, pages 90–104. Cong-Vinh, P. (ed.), 2011.
7. S. Dolev. *Self-Stabilization*. MIT Press, 2000.
8. M. GÜdemann, F. Ortmeier, and W. Reif. Safety and dependability analysis of self-adaptive systems. In *Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, ISOLA '06, pages 177–184, Washington, DC, USA, 2006. IEEE Computer Society.
9. D. Hansen and M. Leuschel. Translating B to TLA+ for validation with TLC: There and back again. Technical Report STUPS/2013/xx, Institut für Informatik, Heinrich-Heine-Universität Düsseldorf, 2013.
10. M. U. Iftikhar and D. Weyns. A case study on formal verification of self-adaptive behaviors in a decentralized system. In *FOCLASA'12*, pages 45–62, 2012.
11. L. Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.
12. G. T. Leavens, J.-R. Abrial, D. S. Batory, M. J. Butler, A. Coglio, K. Fislser, E. C. R. Hehner, C. B. Jones, D. Miller, S. L. P. Jones, M. Sitaraman, D. R. Smith, and A. Stump. Roadmap for enhanced languages and methods to aid verification. In S. Jarzabek, D. C. Schmidt, and T. L. Veldhuizen, editors, *GPCE*, pages 221–236. ACM, 2006.
13. M. Leuschel and M. Butler. ProB: A model checker for B. In A. Keijiro, S. Gnesi, and M. Dino, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 855–874. Springer-Verlag, 2003.
14. C. C. Marquezan and L. Z. Granville. *Self-\* and P2P for Network Management - Design Principles and Case Studies*. Springer Briefs in Computer Science. Springer, 2012.
15. D. Méry. Refinement-based guidelines for algorithmic systems. *International Journal of Software and Informatics*, 3(2-3):197–239, June/September 2009.
16. D. Méry and N. K. Singh. Automatic code generation from event-b models. In *Proceedings of the Second Symposium on Information and Communication Technology*, SoICT '11, pages 179–188, New York, NY, USA, 2011. ACM.
17. L. Penserini, H. Aldewereld, F. Dignum, and V. Dignum. Adaptivity within an organizational development framework. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO '08, pages 477–478, Washington, DC, USA, 2008. IEEE Computer Society.
18. I. S. W. B. Prasetya and S. D. Swierstra. Formal design of self-stabilizing programs: Theory and examples, 2000.
19. Project RODIN. Rigorous open development environment for complex systems. <http://www.eventb.org/>, 2004-2010.
20. M. Puviani, G. D. M. Serugendo, R. Frei, and G. Cabri. A method fragments approach to methodologies for engineering self-organizing systems. *ACM Trans. Auton. Adapt. Syst.*, 7(3):33:1–33:25, Oct. 2012.
21. G. Smith and J. W. Sanders. Formal development of self-organising systems. In *Proceedings of the 6th International Conference on Autonomic and Trusted Computing*, ATC '09, pages 90–104, Berlin, Heidelberg, 2009. Springer-Verlag.

22. J. Sudeikat, J.-P. Steghöfer, H. Seebach, W. Reif, W. Renz, T. Preisler, and P. Salchow. Design and simulation of a wave-like self-organization strategy for resource-flow systems. In *MALLOW'10*, pages –1–1, 2010.
23. D. Tosi. Research perspectives in self-healing systems. Technical report, DISE LTA, 2004.
24. D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C\* Conference on Computer Science and Software Engineering, C3S2E '12*, pages 67–79, New York, NY, USA, 2012. ACM.
25. D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Trans. Auton. Adapt. Syst.*, 7(1):8:1–8:61, May 2012.



## **A Appendix : EVENT-B models**

C00

```
CONTEXT
  C00    >
SETS
  SERVICES    >
  STATES      >
CONSTANTS
  RUN    >
  FAIL   >
  initState    >
AXIOMS
  axm1: SERVICES ≠ ∅ not theorem >
  axm2: STATES = {RUN, FAIL} not theorem >
  axm3: RUN ≠ FAIL not theorem >
  axm4: initState ∈ SERVICES ↔ STATES not theorem >
  axm5: ∀ s · s ∈ SERVICES ⇒ s ↦ RUN ∈ initState not theorem >
  axm6: ∀ s, st1, st2 · s ∈ SERVICES ∧ st1 ∈ STATES ∧ st2 ∈ STATES ∧ s ↦
st1 ∈ initState ∧ s ↦ st2 ∈ initState ⇒ st1 = st2 not theorem >
END
```

C01

```
CONTEXT
  C01 >
EXTENDS
  C00
SETS
  STATES_1 >
CONSTANTS
  RUN_1 >
  FAIL_1 >
  FAIL_DETECT_1 >
  initState_1 >
AXIOMS
  axm1: partition(STATES_1, {RUN_1},{FAIL_1},{FAIL_DETECT_1}) not
theorem >
  axm2: initState_1 ∈ SERVICES ↔ STATES_1 not theorem >
  axm3: ∀ s · s ∈ SERVICES ⇒ s ↦ RUN_1 ∈ initState_1 not theorem >
  axm4: ∀ s, st1, st2 · s ∈ SERVICES ∧ st1 ∈ STATES_1 ∧ st2 ∈ STATES_1 ∧
s ↦ st1 ∈ initState_1 ∧ s ↦ st2 ∈ initState_1 ⇒ st1 = st2 not theorem >
END
```

C02

```
CONTEXT
  C02    >
EXTENDS
  C01
SETS
  STATES_2    >
CONSTANTS
  RUN_2      >
  FAIL_2     >
  FAIL_DETECT_2  >
  FAIL_ACTIV_2  >
  initState_2 >
AXIOMS
  axm1:    partition(STATES_2, {RUN_2},{FAIL_2},{FAIL_DETECT_2},
{FAIL_ACTIV_2}) not theorem >
  axm2:    initState_2 ∈ SERVICES ↔ STATES_2 not theorem >
  axm3:    ∀ s · s ∈ SERVICES ⇒ s ↦ RUN_2 ∈ initState_2 not theorem >
  axm4:    ∀ s, st1, st2 · s ∈ SERVICES ∧ st1 ∈ STATES_2 ∧ st2 ∈ STATES_2 ∧
s ↦ st1 ∈ initState_2 ∧ s ↦ st2 ∈ initState_2 ⇒ st1 = st2 not theorem >
END
```

## C03

```
CONTEXT
  C03 >
EXTENDS
  C02
SETS
  STATES_3 >
CONSTANTS
  RUN_3 >
  FAIL_3 >
  FAIL_DETECT_3 >
  FAIL_ACTIV_3 >
  FAIL_CONFIG_3 >
  FAIL_IGN_3 >
  initState_3 >
AXIOMS
  axm1: partition(STATES_3, {RUN_3},{FAIL_3},{FAIL_DETECT_3},
{FAIL_ACTIV_3},{FAIL_CONFIG_3},{FAIL_IGN_3}) not theorem >
  axm2: initState_3 ∈ SERVICES ↔ STATES_3 not theorem >
  axm3: ∀ s · s ∈ SERVICES ⇒ s ↦ RUN_3 ∈ initState_3 not theorem >
  axm4: ∀ s, st1, st2 · s ∈ SERVICES ∧ st1 ∈ STATES_3 ∧ st2 ∈ STATES_3 ∧
s ↦ st1 ∈ initState_3 ∧ s ↦ st2 ∈ initState_3 ⇒ st1 = st2 not theorem >
END
```

## C04

```
CONTEXT
  C04    >
EXTENDS
  C03
SETS
  STATES_4    >
CONSTANTS
  RUN_4      >
  FAIL_4     >
  FAIL_DETECT_4  >
  FAIL_ACTIV_4    >
  FAIL_CONFIG_4   >
  FAIL_IGN_4     >
  DPL_4         >
  initState_4   >
AXIOMS
  axm1:  partition(STATES_4, {RUN_4},{FAIL_4},{FAIL_DETECT_4},
{FAIL_ACTIV_4},{FAIL_CONFIG_4},{FAIL_IGN_4},{DPL_4}) not theorem >
  axm2:  initState_4 ∈ SERVICES ↔ STATES_4 not theorem >
  axm3:  ∀ s · s ∈ SERVICES ⇒ s ↦ RUN_4 ∈ initState_4 not theorem >
  axm4:  ∀ s, st1, st2 · s ∈ SERVICES ∧ st1 ∈ STATES_4 ∧ st2 ∈ STATES_4 ∧
s ↦ st1 ∈ initState_4 ∧ s ↦ st2 ∈ initState_4 ⇒ st1 = st2 not theorem >
END
```

## C05

### CONTEXT

C05 >

### EXTENDS

C04

### CONSTANTS

min\_inst >

init\_inst >

### AXIOMS

axm1:  $\text{min\_inst} \in \text{SERVICES} \rightarrow \text{N1}$  not theorem >

axm2:  $\text{init\_inst} \in \text{SERVICES} \rightarrow \text{N1}$  not theorem >

axm3:  $\forall s \cdot s \in \text{SERVICES} \Rightarrow \text{min\_inst}(s) \geq 2$  not theorem >

axm4:  $\forall s \cdot s \in \text{SERVICES} \Rightarrow \text{init\_inst}(s) \geq \text{min\_inst}(s)$  not theorem >

axm5:  $\forall s \cdot s \in \text{SERVICES} \Rightarrow \text{init\_inst}(s) \geq 2$  theorem >

### END

C06

```

CONTEXT
  C06    >
EXTENDS
  C05
SETS
  PEERS  >Set of PEERS
CONSTANTS
  InitSrvcPeers  >Initial set of peers / instances per service
AXIOMS
  axm1:  InitSrvcPeers ∈ SERVICES → P1(PEERS) not theorem >each service
is provided by a non empty set of peers/instances
  axm2:  ∀ s · s ∈ SERVICES ⇒ finite(InitSrvcPeers(s)) not theorem >each
service is provided by a finite set of peers/instances
  axm3:  ∀ s · s ∈ SERVICES ⇒ card(InitSrvcPeers(s)) = init_inst(s) not
theorem >each service s is provided by peers/instances, whose number is
init_inst(s)
  axm4:  ∀ s1, s2 · s1 ⊆ PEERS ∧ s2 ⊆ PEERS ∧ s1 ≠ ∅ ∧ s2 ≠ ∅ ∧ finite(s1)
∧ finite(s2) ∧ s1 ⊆ s2 ⇒ card(s1) ≤ card(s2)-1 not theorem >
  axm5:  ∀ s1 · s1 ⊆ PEERS ∧ s1 ≠ ∅ ∧ finite(s1) ⇒ card(s1) > 0 theorem >
  axm6:  ∀ s1, s2 · s1 ⊆ PEERS ∧ s2 ⊆ PEERS ∧ finite(s1) ∧ finite(s2) ∧ s1
⊆ s2 ⇒ card(s2) - card(s1) = card(s2\s1) not theorem >
END

```



## C07

```
CONTEXT
  C07    >
EXTENDS
  C06
CONSTANTS
  deplo_inst  >
AXIOMS
  axm1:   $\forall \text{ set}, s1, s2 \cdot \text{set} \subseteq \text{SERVICES} \times \text{PEERS} \wedge s1 \in \text{SERVICES} \wedge s2 \in \text{SERVICES} \wedge s1 = s2 \Rightarrow (\{s1\} \triangleleft \text{set})[\{s2\}] = \emptyset$  theorem >
  axm2:   $\forall \text{ set}, s1, s2 \cdot \text{set} \subseteq \text{SERVICES} \times \text{PEERS} \wedge s1 \in \text{SERVICES} \wedge s2 \in \text{SERVICES} \wedge s1 \neq s2 \Rightarrow (\{s1\} \triangleleft \text{set})[\{s2\}] = \text{set}[\{s2\}]$  theorem >
  axm3:   $\forall \text{ set}, s1, s2, p \cdot \text{set} \subseteq \text{SERVICES} \times \text{PEERS} \wedge s1 \in \text{SERVICES} \wedge s2 \in \text{SERVICES} \wedge p \in \text{PEERS} \wedge s1 = s2 \Rightarrow (\text{set} \cup \{s1 \mapsto p\})[\{s2\}] = \text{set}[\{s2\}] \cup \{p\}$  theorem >
  axm4:   $\forall \text{ set}, s1, s2, p \cdot \text{set} \subseteq \text{SERVICES} \times \text{PEERS} \wedge s1 \in \text{SERVICES} \wedge s2 \in \text{SERVICES} \wedge p \in \text{PEERS} \wedge s1 \neq s2 \Rightarrow (\text{set} \cup \{s1 \mapsto p\})[\{s2\}] = \text{set}[\{s2\}]$  theorem >
  axm5:   $\text{deplo\_inst} \in \text{SERVICES} \rightarrow \mathbb{N}1$  not theorem >
END
```

```

CONTEXT
  C08      >
EXTENDS
  C07
CONSTANTS
  init_tok      >
  InitStatus    >
  InitSuspPeers >
  InitFail      >
AXIOMS
  axm1:  init_tok ∈ SERVICES → PEERS not theorem >
  axm2:  ∀ s · s ∈ SERVICES ⇒ init_tok(s) ∈ InitSrvcPeers(s) not theorem
>
  axm3:  ∀ a1, a2 · a1 ∈ PEERS ↔ (SERVICES×PEERS) ∧ a2 ∈ PEERS ↔
(SERVICES×PEERS) ∧ finite(a1) ∧ a2 ⊆ a1 ⇒ finite(a2) not theorem >
  axm4:  InitStatus ∈ (PEERS × SERVICES) ↔ STATES_4 not theorem >
  axm5:  ∀ s, p · s ∈ SERVICES ∧ p ∈ PEERS ∧ p = init_tok(s) ⇒ (p ↦ s) ↦
RUN_4 ∈ InitStatus not theorem >
  axm6:  ∀ s, p, stt · s ∈ SERVICES ∧ p ∈ PEERS ∧ stt ∈ STATES_4 ∧ (p ↦
s) ↦ stt ∈ InitStatus ⇒ p = init_tok(s) ∧ stt = RUN_4 not theorem >
  axm7:  InitSuspPeers ∈ (PEERS×SERVICES) ↔ P̄(PEERS) not theorem >
  axm8:  ∀ p, s, sp · p ∈ PEERS ∧ s ∈ SERVICES ∧ sp ⊆ PEERS ∧ (p ↦ s) ↦
sp ∈ InitSuspPeers ⇒ p = init_tok(s) ∧ sp = ∅ not theorem >
  axm9:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p = init_tok(s) ⇒ (p ↦ s) ↦
∅ ∈ InitSuspPeers not theorem >
  axm10: InitFail ∈ SERVICES → P̄(PEERS) not theorem >
  axm11: ∀ s · s ∈ SERVICES ⇒ InitFail(s) = ∅ not theorem >
END

```

```

CONTEXT
  C09    >
EXTENDS
  C08
CONSTANTS
  initStateSrv    >
  InitSuspPrs    >
  InitRunPeers    >
AXIOMS
  axm1:  initStateSrv ∈ PEERS × SERVICES ↔ STATES_4 not theorem >
  axm2:  ∀ s, p · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ∈ InitSrvPeers(s) ⇒ (p ↦
s) ↦ RUN_4 ∈ initStateSrv not theorem >
  axm3:  ∀ s, p, stt · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ↦ stt ∈
InitStateSrv ⇒ p ∈ InitSrvPeers(s) ∧ stt = RUN_4 not theorem >
  axm4:  InitSuspPrs ∈ PEERS × SERVICES ↔ ℙ(PEERS) not theorem >
  axm5:  ∀ s, p · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ∈ InitSrvPeers(s) ⇒ (p ↦
s) ↦ ∅ ∈ InitSuspPrs not theorem >
  axm6:  ∀ s, p, stt · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ↦ stt ∈
InitSuspPrs ⇒ p ∈ InitSrvPeers(s) ∧ stt = ∅ not theorem >
  axm7:  InitRunPeers ∈ PEERS × SERVICES ↔ ℙ(PEERS) not theorem >
  axm8:  ∀ s, p · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ∈ InitSrvPeers(s) ⇒ (p ↦
s) ↦ InitSrvPeers(s) ∈ InitRunPeers not theorem >
  axm9:  ∀ s, p, stt · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ↦ stt ∈
InitRunPeers ⇒ p ∈ InitSrvPeers(s) ∧ stt = InitSrvPeers(s) not theorem >
END

```

M00

```

MACHINE
  M00 >
SEES
  C00
VARIABLES
  serviceState >
INVARIANTS
  inv1:  serviceState ∈ SERVICES ↔ STATES not theorem >
  inv2:  ∀ s, st1, st2 · s ∈ SERVICES ∧ st1 ∈ STATES ∧ st2 ∈ STATES ∧ s ↦
st1 ∈ serviceState ∧ s ↦ st2 ∈ serviceState ⇒ st1 = st2 not theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
    THEN
      act1:  serviceState = InitState >
    END

  FAIL:  not extended ordinary >
    ANY
      s >
    WHERE
      grd1:  s ∈ SERVICES not theorem >
    THEN
      act1:  serviceState = ({s} ◀ serviceState) ∪ {s ↦ FAIL} >
    END

  HEAL:  not extended ordinary >
    ANY
      s >
    WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  s ↦ FAIL ∈ serviceState not theorem >
    THEN
      act1:  serviceState = (serviceState \ {s ↦ FAIL}) ∪ {s ↦ RUN} >
    END
END

```

M01

```

MACHINE
  M01 >
REFINES
  M00
SEES
  C01
VARIABLES
  serviceState_1 >
INVARIANTS
  inv1:  serviceState_1 ∈ SERVICES ↔ STATES_1 not theorem >
  gluing_run1:  ∀ s · s ∈ SERVICES ∧ s ↦ RUN ∈ serviceState ⇒ s ↦ RUN_1
∈ serviceState_1 not theorem >
  gluing_run2:  ∀ s · s ∈ SERVICES ∧ s ↦ RUN_1 ∈ serviceState_1 ⇒ s ↦
RUN ∈ serviceState not theorem >
  gluing_fail1:  ∀ s · s ∈ SERVICES ∧ s ↦ FAIL ∈ serviceState ⇒ (s ↦
FAIL_1 ∈ serviceState_1 ∨ s ↦ FAIL_DETECT_1 ∈ serviceState_1) not theorem >
  gluing_fail2:  ∀ s, st · s ∈ SERVICES ∧ st ∈ STATES_1 ∧ st ∈
{FAIL_1, FAIL_DETECT_1} ∧ s ↦ st ∈ serviceState_1 ⇒ s ↦ FAIL ∈ serviceState not
theorem >
  gluing_state3:  ∀ s, st1, st2 · s ∈ SERVICES ∧ st1 ∈ STATES_1 ∧ st2 ∈
STATES_1 ∧ s ↦ st1 ∈ serviceState_1 ∧ s ↦ st2 ∈ serviceState_1 ⇒ st1 = st2 not
theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
    THEN
      act1:  serviceState_1 :=  InitState_1 >
    END

  FAIL:  not extended ordinary >
    REFINES
      FAIL
    ANY
      s >
    WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  s ↦ RUN_1 ∈ serviceState_1 not theorem >
    THEN
      act1:  serviceState_1 := (serviceState_1 \ {s ↦ RUN_1}) ∪ {s ↦
FAIL_1} >
    END

  FAIL_DETECT:  not extended ordinary >
    REFINES
      FAIL
    ANY
      s >
    WHERE
      grd1:  s ∈ SERVICES not theorem >

```

M01

```

    grd2:  s ↦ FAIL_1 ∈ serviceState_1 not theorem >
  THEN
    act1:  serviceState_1 := (serviceState_1 \ {s ↦ FAIL_1}) ∪ {s ↦
FAIL_DETECT_1} >
  END

  HEAL:  not extended ordinary >
  REFINES
    HEAL
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_DETECT_1 ∈ serviceState_1 not theorem >
  THEN
    act1:  serviceState_1 := (serviceState_1 \ {s ↦ FAIL_DETECT_1})
∪ {s ↦ RUN_1} >
  END

END
```

M02

```

MACHINE
  M02 >
REFINES
  M01
SEES
  C02
PRIORS
  servi2 >
INvariants
  i >
  R0servi2 : s SE R CES ^ s RON servi > => s
  RNServi2 : s SE R CES ^ s RON 2 servi2 => s
  FAI2servi2 : s SER CES ^ s FAI servi > => s
  FAI2servi2 : s SER CES ^ s FAI 2 servi2 => s
  FAIServi2 : s SER CES ^ s FAI ETECT servi > => s
=> (s FAI FAI 2 servi2 v s FAI ETECT servi 2)
t >
  FAI FAI ETECT ^ s st SER CES ^ st STATES ^ st
servi2 >
  STATES ^ s st 1 servi2 ^ s st2 ser vi2 st 1 = st2
EVENTS
  INITIALISATION: >
  THEN
    a > I 2 >
  END
  FAIL >
  REFINES
    FAI
  AND
    >
  BRE
    >
  THEN
    a > (servi2 2 R > )
FAI2 >
  END
  FAIL_DETECT >
  REFINES

```

```

FAIERTECT
  AN
  BRE
  SERVICES FAIERTECT
  THEN
    a
  EN
FAIERTECT
IS_OK
  REFINES
  HEAD
  AN
  BRE
  SERVICES FAIERTECT CT2
  THEN
    a
  EN
FAIERTECT
  REFINES
  HEAD
  AN
  BRE
  SERVICES FAIERTECT CT2
  THEN
    a
  EN
FAIERTECT
  REFINES
  HEAD
  AN
  BRE
  SERVICES FAIERTECT CT2
  THEN
    a
  EN
EAL
  REFINES
  HEAD
  AN
  BRE
  SERVICES FAIERTECT CT2
  THEN
    a
  EN

```



M02

EN

□□□□

```

MACHINE
  M03 >
REFINES
  M0
SEES
  C03
PRIORS
  servi
INITIALS
  i
  RUN
  RUN
  FAI
  FAI
  (s
  FAI
  FAI
  STATES
  t
EVENTS
  INITIALISATION:
  THEN
  END
  FAIL:
  REFINES
  AN
  BRE
  THEN
  FAI

```

```

END
FAIL_DETECT:  @>
REFINES
  FAIL_DETECT
AND
  >
  BRE
    SERVICES >
    FAIL servi >
  THEN
a 3 (servi 3 FAIL >
FAIL_DETECT
END

IS_OK:  @>
REFINES
  IS_OK
AND
  >
  BRE
    SERVICES >
    FAIL CTB servi 3 >
  THEN
a 3 (servi 3 FAIL TECT >
  RUN
  >
  END

FAIL_ACTI:  @>
REFINES
  FAIL_ACTI
AND
  >
  BRE
    SERVICES >
    FAIL CTB servi 3 >
  THEN
a 3 (servi 3 FAIL TECT >
  FAIL_ACTI 3 >
  END

FAIL_CONF_I: not extended ordinary >
ANY
  s >
  st >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd3: st ∈ {FAIL_CONFIG_3, FAIL_IGN_3} not theorem >

```

M03

```

    grd2:  s ↦ FAIL_ACTIV_3 ∈ serviceState_3 not theorem >
  THEN
    act1:  serviceState_3 := (serviceState_3 \ {s ↦ FAIL_ACTIV_3}) ∪
{s ↦ st} >
  END

  HEAL:  not extended ordinary >
  REFINES
    HEAL
  ANY
    s    >
    st   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd3:  st ∈ {FAIL_CONFIG_3, FAIL_IGN_3} not theorem >
    grd2:  s ↦ st ∈ serviceState_3 not theorem >
  THEN
    act1:  serviceState_3 := (serviceState_3 \ {s ↦ st}) ∪ {s ↦
RUN_3} >
  END

  END
```



```

AN
  >
  HERE
    VERES not theorem >
    gr RON4 S4 >
  THEN
    a S 4 S 4 R
FAI4 >
  EN

```

```

FAIL_DETECT  >
  REFINES
    FAIE
  AN
    >
    HERE
      VERES not theorem >
      gr FAI4 S4 >
  THEN
    a S 4 S 4 FA
FAIE >
  EN

```

```

IS_OK  >
  REFINES
    IS
  AN
    >
    HERE
      VERES not theorem >
      gr FAIECT4 S 4 >
  THEN
    a S 4 S 4 CT4 4 >
  > RON4 >
  EN

```

```

FAIL_ACTIV  >
  REFINES
    FAIIV
  AN
    >
    HERE
      VERES not theorem >
      gr FAIECT4 S 4 >
  THEN
    a S 4 S 4 CT4 4 >
  > FAIIV _4 >
  EN

```



M04

```

s >
[HERE
  [VERES not theorem >
  gr[CONF I04 [S0000004>
THEN
  a[CONF S000000 4 [S0000006[CONF NFI04004 [
[0004 >
EN]

[EAL] [CONF S000000 [CONF S000000 >
REFINES
  HEAL
  AN[
  [ >
  [HERE
    [VERES not theorem >
    gr[CONF 04 [ [S0000004[CONF >
  [IH
    [CONF [CONF >3
  THEN
    a[CONF S000000 4 [S000000 [CONF 04 [CONF [
RON04 >
  EN]
EN]

```



```

MACHINE
  M05 >
REFINES
  M04
SEES
  C05
VARIABLES
  serviceState_4 >
  num_run >
  num_susp >
INVARIANTS
  inv1: num_run ∈ SERVICES → N1 not theorem >
  inv2: num_susp ∈ SERVICES → N not theorem >
  inv3: ∀ s, st · s ∈ SERVICES ∧ st ∈ STATES_4 ∧ st ∉ {FAIL_4,
  FAIL_DETECT_4} ∧ s ↦ st ∈ serviceState_4 ⇒ num_susp(s) = 0 not theorem >
  inv4: ∀ s · s ∈ SERVICES ∧ s ↦ RUN_4 ∈ serviceState_4 ⇒ num_susp(s) =
  0 theorem >
  inv5: ∀ s · s ∈ SERVICES ∧ s ↦ FAIL_CONFIG_4 ∈ serviceState_4 ⇒
  num_run(s) < min_inst(s) not theorem >
  inv6: ∀ s · s ∈ SERVICES ⇒ num_susp(s) < num_run(s) not theorem >
EVENTS
  INITIALISATION: extended ordinary >
    THEN
      act1: serviceState_4 := InitState_4 >
      act2: num_run := init_inst >
      act3: num_susp := SERVICES×{0} >
    END

  FAIL: extended ordinary >
    REFINES
      FAIL
    ANY
      s >
      nb_fail >
    WHERE
      grd1: s ∈ SERVICES not theorem >
      grd2: s ↦ RUN_4 ∈ serviceState_4 not theorem >
      grd3: nb_fail ∈ N1 not theorem >
      grd4: nb_fail < num_run(s) not theorem >
    THEN
      act1: serviceState_4 := (serviceState_4 \ {s ↦ RUN_4}) ∪ {s ↦
  FAIL_4} >
      act2: num_susp(s) := nb_fail >
    END

  FAIL_DETECT: extended ordinary >
    REFINES
      FAIL_DETECT

```

```

ANY
  s >
  num_safe >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_4 ∈ serviceState_4 not theorem >
  grd3: num_safe ∈ N not theorem >
  grd4: num_safe ≤ num_susp(s) not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_4}) ∪ {s ↦
FAIL_DETECT_4} >
  act2: num_susp(s) := num_susp(s) - num_safe >
END

IS_OK: extended ordinary >
REFINES
  IS_OK
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
  grd3: num_susp(s) = 0 not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ RUN_4} >
END

FAIL_ACTIV: extended ordinary >
REFINES
  FAIL_ACTIV
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
  grd3: num_susp(s) > 0 not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ FAIL_ACTIV_4} >
  act2: num_run(s) := num_run(s) - num_susp(s) >
  act3: num_susp(s) := 0 >
END

FAIL_CONFIGURE: extended ordinary >
REFINES
  FAIL_CONFIGURE
ANY

```

```

s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
  grd3: num_run(s) < min_inst(s) not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_CONFIG_4} >
END

FAIL_IGNORE: extended ordinary >
REFINES
  FAIL_IGNORE
ANY
s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
  grd3: num_run(s) ≥ min_inst(s) not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_IGN_4} >
END

IGNORE: extended ordinary >
REFINES
  IGNORE
ANY
s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_IGN_4 ∈ serviceState_4 not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_IGN_4}) ∪
{s ↦ RUN_4} >
END

REDEPLOY: extended ordinary >
REFINES
  REDEPLOY
ANY
s >
  new_run >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
  grd3: new_run ∈ N1 not theorem >
  grd4: new_run ≥ min_inst(s) not theorem >

```

M05

```
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_CONFIG_4})
u {s ↦ DPL_4} >
  act2:  num_run(s) := new_run >
END

□EAL:  extended ordinary >
REFINES
  HEAL
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ DPL_4 ∈ serviceState_4 not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ DPL_4}) ∪ {s ↦
RUN_4} >
  END
END
```

M06

```

MACHINE
  M06    >
REFINES
  M05
SEES
  C06
VARIABLES
  serviceState_4  >
  run_peers      >
  susp_peers     >
  fail_peers     >
INVARIANTS
  inv1:  run_peers ∈ SERVICES → ℙ1(PEERS) not theorem >
  inv2:  susp_peers ∈ SERVICES ↔ ℙ(PEERS) not theorem >
  inv3:  fail_peers ∈ SERVICES ↔ PEERS not theorem >
  gluing_run1:  ∀ s · s ∈ SERVICES ⇒ finite(run_peers(s)) not theorem
>the number of instances providing a service s is finite
  gluing_run2:  ∀ s · s ∈ SERVICES ⇒ num_run(s) = card(run_peers(s))
not theorem >the number of instances providing a service s is num_run_peers(s)
  gluing_susp1:  ∀ s · s ∈ SERVICES ∧ s ∈ dom(susp_peers) ⇒ finite
(susp_peers(s)) not theorem >the number of suspect instances of a service s is
finite
  gluing_susp2:  ∀ s · s ∈ SERVICES ∧ s ∈ dom(susp_peers) ⇒ num_susp(s)
= card(susp_peers(s)) not theorem >the number of suspect instances of a service
s is num_susp_peers(s)
  inv4:  ∀ s · s ∈ SERVICES ⇒ run_peers(s) ∩ fail_peers[{s}] = ∅ not
theorem >an instance of a service s is either failed or providing the service s
  inv5:  ∀ s · s ∈ SERVICES ∧ s ∈ dom(susp_peers) ⇒ susp_peers(s) ⊆
run_peers(s) not theorem >suspicious instances of s are a subset of the
instances providing s
  inv6:  ∀ s, st · s ∈ SERVICES ∧ st ∈ STATES_4 ∧ st ∈ {FAIL_4,
FAIL_DETECT_4} ∧ s ⇒ st ∈ serviceState_4 ⇒ s ∈ dom(susp_peers) not theorem >
  inv7:  ∀ s, st · s ∈ SERVICES ∧ st ∈ STATES_4 ∧ st ∈ {FAIL_4,
FAIL_DETECT_4} ∧ s ⇒ st ∈ serviceState_4 ⇒ susp_peers(s) ⊆ run_peers(s) not
theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
  THEN
    act1:  serviceState_4 := InitState_4 >
    act2:  run_peers := InitSrvcPeers >
    act3:  susp_peers := ∅ >
    act4:  fail_peers := ∅ >
  END

  FAIL:  not extended ordinary >
REFINES
  FAIL
ANY

```

```

s      >
fp     >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ RUN_4 ∈ serviceState_4 not theorem >
  grd5:  fp ⊆ PEERS not theorem >
  grd3:  fp ≠ ∅ not theorem >
  grd4:  fp ⊂ run_peers(s) not theorem >
WITH
  nb_fail:  nb_fail=card(fp) >
THEN
  act1:  serviceState_4 = (serviceState_4\{s ↦ RUN_4}) ∪ {s ↦
FAIL_4} >
  act2:  susp_peers(s) = fp >
END

FAIL_DETECT:  not extended ordinary >
REFINES
  FAIL_DETECT
ANY
  s      >
  sf     >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_4 ∈ serviceState_4 not theorem >
  grd5:  susp_peers(s) ≠ ∅ not theorem >
  grd6:  sf ⊆ PEERS not theorem >
  grd7:  sf ⊆ susp_peers(s) not theorem >
WITH
  num_safe:  num_safe=card(sf) >
THEN
  act1:  serviceState_4 = (serviceState_4\{s ↦ FAIL_4}) ∪ {s ↦
FAIL_DETECT_4} >
  act2:  susp_peers(s) = susp_peers(s) \ sf >
END

IS_OK:  not extended ordinary >
REFINES
  IS_OK
ANY
  s      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
  grd5:  susp_peers(s) = ∅ not theorem >
THEN
  act1:  serviceState_4 = (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ RUN_4} >

```

```

END

FAIL_ACTIV:      not extended ordinary >
REFINES
  FAIL_ACTIV
ANY
  s      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
  grd5:  susp_peers(s) ≠ ∅ not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
u {s ↦ FAIL_ACTIV_4} >
  act2:  run_peers(s) := run_peers(s) \ susp_peers(s) >
  act3:  susp_peers(s) := ∅ >
  act4:  fail_peers := fail_peers ∪ ({s} × susp_peers(s)) >
END

FAIL_CONFIGURE:  not extended ordinary >
REFINES
  FAIL_CONFIGURE
ANY
  s      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
  grd3:  card(run_peers(s)) < min_inst(s) not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_CONFIG_4} >
END

FAIL_IGNORE:    not extended ordinary >
REFINES
  FAIL_IGNORE
ANY
  s      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
  grd3:  card(run_peers(s)) ≥ min_inst(s) not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_IGN_4} >
END

IGNORE:         extended ordinary >

```

```

REFINES
  IGNORE
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_IGN_4 ∈ serviceState_4 not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_IGN_4}) ∪
{s ↦ RUN_4} >
END

REDEPLOY□: not extended ordinary >
REFINES
  REDEPLOY
ANY
  s >
  new_inst >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
  grd3: new_inst ⊆ PEERS not theorem >
  grd5: new_inst ≠ ∅ not theorem >
  grd6: finite(new_inst) not theorem >
  grd7: run_peers(s) ∩ new_inst = ∅ not theorem >
  grd8: fail_peers[{s}] ∩ new_inst = ∅ not theorem >
  grd4: card(run_peers(s))+card(new_inst) ≥ min_inst(s) not
theorem >
WITH
  new_run: new_run=card(run_peers(s))+card(new_inst) >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_CONFIG_4})
∪ {s ↦ DPL_4} >
  act2: run_peers(s) := run_peers(s) ∪ new_inst >
END

□EAL: extended ordinary >
REFINES
  HEAL
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ DPL_4 ∈ serviceState_4 not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {s ↦ DPL_4}) ∪ {s ↦
RUN_4} >
END

```



```
UNFAIL_PEER:  not extended ordinary >
  ANY
    s    >
    p    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  p ∈ PEERS not theorem >
    grd3:  s ↦ p ∈ fail_peers not theorem >
  THEN
    act1:  fail_peers := fail_peers \ {s ↦ p} >
  END
END
```

M07

```

MACHINE
  M07 >
REFINES
  M06
SEES
  C07
VARIABLES
  serviceState_4 >
  run_peers >
  susp_peers >
  fail_peers >
  dep_inst >
INVARIANTS
  inv1: dep_inst ∈ SERVICES ↔ PEERS not theorem >
  inv2: ∀ s · s ∈ SERVICES ⇒ dep_inst[{s}] ∩ fail_peers[{s}] = ∅ not
theorem >
  inv3: ∀ s, st · s ∈ SERVICES ∧ st ∈ STATES_4 ∧ s ↦ st ∈ serviceState_4
∧ st ≠ FAIL_CONFIG_4 ⇒ dep_inst[{s}] = ∅ not theorem >
  inv4: ∀ s · s ∈ SERVICES ⇒ finite(dep_inst[{s}]) not theorem >
  inv5: ∀ s · s ∈ SERVICES ⇒ dep_inst[{s}] ∩ run_peers(s) = ∅ not
theorem >
EVENTS
  INITIALISATION: extended ordinary >
  THEN
    act1: serviceState_4 := InitState_4 >
    act2: run_peers := InitSrvcPeers >
    act3: susp_peers := ∅ >
    act4: fail_peers := ∅ >
    act5: dep_inst := ∅ >
  END

  FAIL: extended ordinary >
  REFINES
    FAIL
  ANY
    s >
    fp >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: s ↦ RUN_4 ∈ serviceState_4 not theorem >
    grd5: fp ⊆ PEERS not theorem >
    grd3: fp ≠ ∅ not theorem >
    grd4: fp ⊂ run_peers(s) not theorem >
  THEN
    act1: serviceState_4 := (serviceState_4 \ {s ↦ RUN_4}) ∪ {s ↦
FAIL_4} >
    act2: susp_peers(s) := fp >
  END

```

```

FAIL_DETECT:    extended ordinary >
  REFINES
    FAIL_DETECT
  ANY
    s    >
    sf   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_4 ∈ serviceState_4 not theorem >
    grd5:  susp_peers(s) ≠ ∅ not theorem >
    grd6:  sf ⊆ PEERS not theorem >
    grd7:  sf ⊆ susp_peers(s) not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_4}) ∪ {s ↦
FAIL_DETECT_4} >
    act2:  susp_peers(s) := susp_peers(s) \ sf >
  END

IS_OK:    extended ordinary >
  REFINES
    IS_OK
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
    grd5:  susp_peers(s) = ∅ not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ RUN_4} >
  END

FAIL_ACTIV:    extended ordinary >
  REFINES
    FAIL_ACTIV
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
    grd5:  susp_peers(s) ≠ ∅ not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ FAIL_ACTIV_4} >
    act2:  run_peers(s) := run_peers(s) \ susp_peers(s) >
    act3:  susp_peers(s) := ∅ >
    act4:  fail_peers := fail_peers ∪ ({s} × susp_peers(s)) >

```

```

END

FAIL_CONFIGURE:    extended ordinary >
  REFINES
    FAIL_CONFIGURE
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
    grd3:  card(run_peers(s)) < min_inst(s) not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_CONFIG_4} >
  END

FAIL_IGNORE:      extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
    grd3:  card(run_peers(s)) ≥ min_inst(s) not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_IGN_4} >
  END

IGNORE:           extended ordinary >
  REFINES
    IGNORE
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_IGN_4 ∈ serviceState_4 not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_IGN_4}) ∪
{s ↦ RUN_4} >
  END

REDEPLO□INST :  not extended ordinary >
  ANY
    s    >
    dep  >
  WHERE

```

```

    grd1:  s ∈ SERVICES not theorem >
    grd2:  dep ⊆ PEERS not theorem >
    grd3:  finite(dep) not theorem >
    grd4:  dep n run_peers(s) = ∅ not theorem >
    grd5:  dep n fail_peers[{s}] = ∅ not theorem >
    grd6:  card(dep) = deplo_inst(s) not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd8:  s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
  THEN
    act1:  dep_inst := dep_inst ∪ ({s}×dep) >
  END

REDEPLOY□:  not extended ordinary >
  REFINES
    REDEPLOY
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
    grd6:  dep_inst[{s}] ≠ ∅ not theorem >
    grd4:  card(run_peers(s))+card(dep_inst[{s}]) ≥ min_inst(s) not
theorem >
  WITH
    new_inst:  new_inst=dep_inst[{s}] >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_CONFIG_4})
u {s ↦ DPL_4} >
    act2:  run_peers(s) := run_peers(s) ∪ dep_inst[{s}] >
    act3:  dep_inst := {s} ◀ dep_inst >
  END

□EAL:  extended ordinary >
  REFINES
    HEAL
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ DPL_4 ∈ serviceState_4 not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ DPL_4}) ∪ {s ↦
RUN_4} >
  END

UNFAIL_PEER:  extended ordinary >
  REFINES

```

M07

```
UNFAIL_PEER
ANY
  s   >
  p   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  p ∈ PEERS not theorem >
  grd3:  s ↦ p ∈ fail_peers not theorem >
THEN
  act1:  fail_peers := fail_peers \ {s ↦ p} >
END
```

END

```

MACHINE
  M08    >
REFINES
  M07
SEES
  C08
VARIABLES
  serviceState_4  >
  run_peers      >
  susp_peers     >
  fail_peers     >
  dep_inst       >
  token_owner    >
  unav_peers     >
  susp_inst      >
INVARIANTS
  inv1:  token_owner ∈ SERVICES → PEERS not theorem >
  inv2:  unav_peers ⊆ PEERS not theorem >
  inv3:  ∀ s · s ∈ SERVICES ⇒ token_owner(s) ∈ run_peers(s)\unav_peers
not theorem >
  inv4:  ∀ s · s ∈ SERVICES ∧ s ∈ dom(susp_peers) ⇒ token_owner(s) ∉
susp_peers(s) not theorem >
  inv5:  susp_inst ∈ PEERS ↔ (SERVICES×PEERS) not theorem >
  inv6:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(susp_inst[{\ld}])
⇒ ld = token_owner(s) not theorem >
  inv7:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(susp_inst[{\ld}]) ∧
ld = token_owner(s) ⇒ ld ∉ susp_inst[{\ld}][{s}] not theorem >
  inv8:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(susp_inst[{\ld}]) ∧
ld = token_owner(s) ⇒ susp_inst[{\ld}][{s}] ⊂ run_peers(s) not theorem >
  inv9:  ∀ ld, s, stt · ld ∈ PEERS ∧ s ∈ SERVICES ∧ stt ∈ STATES_4 ∧ s ⇨
stt ∈ serviceState_4 ∧ ld = token_owner(s) ∧ stt ≠ RUN_4 ⇒ susp_inst[{\ld}][{s}]
= ∅ not theorem >
EVENTS
  INITIALISATION:  extended ordinary >
  THEN
    act1:  serviceState_4 := initState_4 >
    act2:  run_peers := InitSrvcPeers >
    act3:  susp_peers := ∅ >
    act4:  fail_peers := ∅ >
    act5:  dep_inst := ∅ >
    act6:  token_owner := init_tok >
    act7:  unav_peers := ∅ >
    act8:  susp_inst := ∅ >
  END
  MAKE_PEER_UNAVAIL:  not extended ordinary >
  ANY
    prs >

```

```

E      >new values for token owner per service if needed
WHERE
  grd1:  prs ⊆ PEERS not theorem >
  grd2:  prs ⊄ unav_peers not theorem >
  grd3:  E ∈ SERVICES → PEERS not theorem >new value for token
owner per service if needed
  grd4:  ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs ⇒ E
(srv) = token_owner(srv) not theorem >If the token owner of a service srv does
not belong to prs, the token owner is not changed
  grd5:  ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧ srv ∉
dom(susp_peers) ⇒ E(srv) ∈ run_peers(srv) \ (unav_peers ∪ prs ∪ fail_peers
[ {srv} ]) not theorem >if the owner of the token for a service becomes
unavailable and the service is not suspicious,

then a new token owner among available peers is chosen
  grd6:  ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧ srv ∈
dom(susp_peers) ⇒ E(srv) ∈ run_peers(srv) \ (unav_peers ∪ prs ∪ susp_peers(srv) ∪
fail_peers[ {srv} ]) not theorem >if the owner of the token for a service becomes
unavailable, and the service

possess suspicious instances, then a new token owner among available and not

suspicious peers is chosen
  THEN
    act1:  unav_peers := unav_peers ∪ prs >the peers in prs become
unavailable
    act2:  token_owner := token_owner ◀ E >new value for token owner
per service is given if needed
    act3:  susp_inst := prs ◀ susp_inst >the peers in prs can not
suspect instances anymore
  END

SUSPECT_INST:  not extended ordinary >
  ANY
  s      >a service s
  susp   >suspicious instances
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  susp ⊆ PEERS not theorem >
    grd3:  susp = run_peers(s) ∩ unav_peers not theorem >instances
in susp are suspicious if the peers running them becomes unavailable
    grd4:  s ∉ dom(susp_inst[ {token_owner(s)} ]) not theorem >the
member of susp have not yet been suspected for s by the token owner of s
    grd5:  s ↦ RUN_4 ∈ serviceState_4 not theorem >the state of s
is OK

```



```

THEN
  act1:  susp_inst := susp_inst u ({token_owner(s)} × ({s}×susp))
  >the members of susp become suspected instances for s by the token owner of s
END

FAIL:  not extended ordinary >
REFINES
  FAIL
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ RUN_4 ∈ serviceState_4 not theorem >
  grd3:  susp_inst[{token_owner(s)}][{s}] ≠ ∅ not theorem >
WITH
  fp: fp=susp_inst[{token_owner(s)}][{s}] >
THEN
  act1:  serviceState_4 := (serviceState_4\{s ↦ RUN_4}) u {s ↦
FAIL_4} >
  act2:  susp_peers(s) := susp_inst[{token_owner(s)}][{s}] >
  act3:  susp_inst := susp_inst ▷ ({s} ◁ ran(susp_inst)) >
END

FAIL_DETECT:  extended ordinary >
REFINES
  FAIL_DETECT
ANY
  s >
  sf >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_4 ∈ serviceState_4 not theorem >
  grd5:  susp_peers(s) ≠ ∅ not theorem >
  grd6:  sf ⊆ PEERS not theorem >
  grd7:  sf ⊆ susp_peers(s) not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4\{s ↦ FAIL_4}) u {s ↦
FAIL_DETECT_4} >
  act2:  susp_peers(s) := susp_peers(s) \ sf >
END

IS_OK:  extended ordinary >
REFINES
  IS_OK
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >

```

M08

```

    grd2:  s  $\mapsto$  FAIL_DETECT_4  $\in$  serviceState_4 not theorem >
    grd5:  susp_peers(s) =  $\emptyset$  not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s  $\mapsto$  FAIL_DETECT_4})
u {s  $\mapsto$  RUN_4} >
  END

  FAIL_ACTIV:  extended ordinary >
  REFINES
    FAIL_ACTIV
  ANY
    s >
  WHERE
    grd1:  s  $\in$  SERVICES not theorem >
    grd2:  s  $\mapsto$  FAIL_DETECT_4  $\in$  serviceState_4 not theorem >
    grd5:  susp_peers(s)  $\neq$   $\emptyset$  not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s  $\mapsto$  FAIL_DETECT_4})
u {s  $\mapsto$  FAIL_ACTIV_4} >
    act2:  run_peers(s) := run_peers(s) \ susp_peers(s) >
    act3:  susp_peers(s) :=  $\emptyset$  >
    act4:  fail_peers := fail_peers  $\cup$  ({s} $\times$ susp_peers(s)) >
  END

  FAIL_CONFIGURE:  extended ordinary >
  REFINES
    FAIL_CONFIGURE
  ANY
    s >
  WHERE
    grd1:  s  $\in$  SERVICES not theorem >
    grd2:  s  $\mapsto$  FAIL_ACTIV_4  $\in$  serviceState_4 not theorem >
    grd3:  card(run_peers(s)) < min_inst(s) not theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s  $\mapsto$  FAIL_ACTIV_4})  $\cup$ 
{s  $\mapsto$  FAIL_CONFIG_4} >
  END

  FAIL_IGNORE:  extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s >
  WHERE
    grd1:  s  $\in$  SERVICES not theorem >
    grd2:  s  $\mapsto$  FAIL_ACTIV_4  $\in$  serviceState_4 not theorem >
    grd3:  card(run_peers(s))  $\geq$  min_inst(s) not theorem >
  THEN

```

```

act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_IGN_4} >
END

IGNORE: extended ordinary >
REFINES
  IGNORE
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_IGN_4 ∈ serviceState_4 not theorem >
THEN
act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_IGN_4}) ∪
{s ↦ RUN_4} >
END

REDEPLOY_INST: extended ordinary >
REFINES
  REDEPLOY_INST
ANY
  s >
  dep >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  dep ⊆ PEERS not theorem >
  grd3:  finite(dep) not theorem >
  grd4:  dep ∩ run_peers(s) = ∅ not theorem >
  grd5:  dep ∩ fail_peers[{s}] = ∅ not theorem >
  grd6:  card(dep) = depl_inst(s) not theorem >
  grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
  grd8:  s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
THEN
act1:  dep_inst := dep_inst ∪ ({s}×dep) >
END

REDEPLOY: extended ordinary >
REFINES
  REDEPLOY
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
  grd6:  dep_inst[{s}] ≠ ∅ not theorem >
  grd4:  card(run_peers(s))+card(dep_inst[{s}]) ≥ min_inst(s) not
theorem >

```

## M08

```

THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_CONFIG_4})
u {s ↦ DPL_4} >
  act2:  run_peers(s) := run_peers(s) u dep_inst[{s}] >
  act3:  dep_inst := {s} ◀ dep_inst >
END

HEAL:  extended ordinary >
REFINES
  HEAL
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ DPL_4 ∈ serviceState_4 not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ DPL_4}) u {s ↦
RUN_4} >
END

UNFAIL_PEER:  extended ordinary >
REFINES
  UNFAIL_PEER
ANY
  s >
  p >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  p ∈ PEERS not theorem >
  grd3:  s ↦ p ∈ fail_peers not theorem >
THEN
  act1:  fail_peers := fail_peers \ {s ↦ p} >
END

MAKE_PEER_AVAIL:  not extended ordinary >
ANY
  p >
WHERE
  grd1:  p ∈ PEERS not theorem >
  grd2:  p ∈ unav_peers not theorem >
THEN
  act1:  unav_peers := unav_peers \ {p} >
END

END

```

```

MACHINE
  M09    >
REFINES
  M08
SEES
  C08
VARIABLES
  serviceState_4  >
  run_peers       >
  susp_peers      >
  fail_peers      >
  dep_inst        >
  token_owner     >
  unav_peers      >
  susp_inst       >
  rec_inst        >instances that are tried to be recontacted
  rct_inst        >instances effectively recontacted after a try
INVARIANTS
  inv1:  rec_inst ∈ PEERS ↔ (SERVICES×PEERS) not theorem >
  inv2:  rct_inst ∈ PEERS ↔ (SERVICES×PEERS) not theorem >
  inv3:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ rct_inst[{ld}][{s}] ≠ ∅ ⇒
rec_inst[{ld}][{s}] ≠ ∅ not theorem >
  inv4:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ rct_inst[{ld}][{s}] ≠ ∅ ⇒
rct_inst[{ld}][{s}] ⊆ rec_inst[{ld}][{s}] not theorem >
  inv5:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(rec_inst[{ld}]) ⇒
ld = token_owner(s) not theorem >
  inv6:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(rec_inst[{ld}]) ∧
ld = token_owner(s) ⇒ ld ∉ rec_inst[{ld}][{s}] not theorem >
  inv7:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(rct_inst[{ld}]) ⇒
ld = token_owner(s) not theorem >
  inv8:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(rct_inst[{ld}]) ∧
ld = token_owner(s) ⇒ ld ∉ rct_inst[{ld}][{s}] not theorem >
  inv9:  dom(rct_inst) ⊆ dom(rec_inst) not theorem >
  inv10: ∀ ld · ld ∈ PEERS ∧ ld ∈ dom(rct_inst) ⇒ rct_inst[{ld}] ⊆
rec_inst[{ld}] theorem >
  inv11: ∀ s · s ∈ SERVICES ∧ s ∈ dom(susp_peers) ⇒ token_owner(s) ∉
susp_peers(s) not theorem >
EVENTS
  INITIALISATION:  extended ordinary >
  THEN
    act1:  serviceState_4 := InitState_4 >
    act2:  run_peers := InitSrvcPeers >
    act3:  susp_peers := ∅ >
    act4:  fail_peers := ∅ >
    act5:  dep_inst := ∅ >
    act6:  token_owner := init_tok >
    act7:  unav_peers := ∅ >
    act8:  susp_inst := ∅ >

```

```
act10: rec_inst := ∅ >
act11: rct_inst := ∅ >
```

END

MAKE\_PEER\_UNAVAIL: extended ordinary >

REFINES

MAKE\_PEER\_UNAVAIL

ANY

prs >

E >new values for token owner per service if needed

WHERE

grd1: prs ⊆ PEERS not theorem >

grd2: prs ⊄ unav\_peers not theorem >

grd3: E ∈ SERVICES → PEERS not theorem >new value for token owner per service if needed

grd4:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \notin \text{prs} \Rightarrow \text{E}(\text{srv}) = \text{token\_owner}(\text{srv})$  not theorem >If the token owner of a service srv does not belong to prs, the token owner is not changed

grd5:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs} \wedge \text{srv} \notin \text{dom}(\text{susp\_peers}) \Rightarrow \text{E}(\text{srv}) \in \text{run\_peers}(\text{srv}) \setminus (\text{unav\_peers} \cup \text{prs} \cup \text{fail\_peers}[\{\text{srv}\}])$  not theorem >if the owner of the token for a service becomes unavailable and the service is not suspicious,

then a new token owner among available peers is chosen

grd6:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs} \wedge \text{srv} \in \text{dom}(\text{susp\_peers}) \Rightarrow \text{E}(\text{srv}) \in \text{run\_peers}(\text{srv}) \setminus (\text{unav\_peers} \cup \text{prs} \cup \text{susp\_peers}(\text{srv}) \cup \text{fail\_peers}[\{\text{srv}\}])$  not theorem >if the owner of the token for a service becomes unavailable, and the service

possess suspicious instances, then a new token owner among available and not

suspicious peers is chosen

THEN

act1: unav\_peers := unav\_peers ∪ prs >the peers in prs become unavailable

act2: token\_owner := token\_owner ◀ E >new value for token owner per service is given if needed

act3: susp\_inst := prs ◀ susp\_inst >the peers in prs can not suspect instances anymore

act4: rec\_inst := prs ◀ rec\_inst >

act5: rct\_inst := prs ◀ rct\_inst >

END

SUSPECT\_INST: extended ordinary >

REFINES

```

    SUSPECT_INST
  ANY
    s    >a service s
    susp >suspicious instances
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: susp ⊆ PEERS not theorem >
    grd3: susp = run_peers(s) n unav_peers not theorem >instances
in susp are suspicious if the peers running them becomes unavailable
    grd4: s ∉ dom(susp_inst[{token_owner(s)}]) not theorem >the
member of susp have not yet been suspected for s by the token owner of s
    grd5: s ↦ RUN_4 ∈ serviceState_4 not theorem >the state of s
is OK
  THEN
    act1: susp_inst := susp_inst ∪ ({token_owner(s)} × ({s}×susp))
>the members of susp become suspected instances for s by the token owner of s
  END

  FAIL: extended ordinary >
  REFINES
    FAIL
  ANY
    s    >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: s ↦ RUN_4 ∈ serviceState_4 not theorem >
    grd3: susp_inst[{token_owner(s)}][{s}] ≠ ∅ not theorem >
  THEN
    act1: serviceState_4 := (serviceState_4 \ {s ↦ RUN_4}) ∪ {s ↦
FAIL_4} >
    act2: susp_peers(s) := susp_inst[{token_owner(s)}][{s}] >
    act3: susp_inst := susp_inst ▷ ({s} ◁ ran(susp_inst)) >
  END

  RECONTACT_INST_OK: not extended ordinary >
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: i ∈ PEERS not theorem >
    grd3: s ↦ FAIL_4 ∈ serviceState_4 not theorem >the state of s
is SUSPICIOUS
    grd4: susp_peers(s) ≠ ∅ not theorem >the set of suspicious
peers for s is not empty
    grd5: i ∈ susp_peers(s) \ unav_peers not theorem >i is a
suspicious instance of s and is available (can be contacted)
    grd6: token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the

```

```

token owner of s has not yet tried to recontact i
    grd7:  rec_inst[{{token_owner(s)}}][{s}]  $\subset$  susp_peers(s) not
theorem >the token owner of s has not yet tried to recontact all the suspicious
instances of s
    THEN
        act1:  rec_inst := rec_inst  $\cup$  {token_owner(s)  $\mapsto$  (s  $\mapsto$  i)} >the
token owner of s has tried to recontact i
        act2:  rct_inst := rct_inst  $\cup$  {token_owner(s)  $\mapsto$  (s  $\mapsto$  i)} >i is
recontacted by the token owner of s successfully
    END

RECONTACT_INST_K0:  not extended ordinary >
    ANY
        s      >a service s
        i      >an instance i
    WHERE
        grd1:  s  $\in$  SERVICES not theorem >
        grd2:  i  $\in$  PEERS not theorem >
        grd3:  s  $\mapsto$  FAIL_4  $\in$  serviceState_4 not theorem >the state of s
is SUSPICIOUS
        grd4:  susp_peers(s)  $\neq$   $\emptyset$  not theorem >the set of suspicious
peers for s is not empty
        grd5:  i  $\in$  susp_peers(s)nunav_peers not theorem >i is a
suspicious instance of s and is unavailable (can not be contacted)
        grd6:  token_owner(s)  $\mapsto$  (s  $\mapsto$  i)  $\notin$  rec_inst not theorem >the
token owner of s has not yet tried to recontact i
        grd7:  rec_inst[{{token_owner(s)}}][{s}]  $\subset$  susp_peers(s) not
theorem >the token owner of s has not yet tried to recontact all the suspicious
instances of s
    THEN
        act1:  rec_inst := rec_inst  $\cup$  {token_owner(s)  $\mapsto$  (s  $\mapsto$  i)} >the
token owner of s has tried to recontact i
    END

FAIL_DETECT:  not extended ordinary >
    REFINES
        FAIL_DETECT
    ANY
        s      >
    WHERE
        grd1:  s  $\in$  SERVICES not theorem >
        grd2:  s  $\mapsto$  FAIL_4  $\in$  serviceState_4 not theorem >
        grd5:  susp_peers(s)  $\neq$   $\emptyset$  not theorem >
        grd8:  rec_inst[{{token_owner(s)}}][{s}] = susp_peers(s) not
theorem >
    WITH
        sf: sf=rct_inst[{{token_owner(s)}}][{s}] >
    THEN

```



```

act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_4}) ∪ {s ↦
FAIL_DETECT_4}
>
act2:  susp_peers(s) := susp_peers(s) \ rct_inst[{token_owner
(s)}][{s}] >
act3:  rec_inst := rec_inst ▷ ({s} ◁ ran(rec_inst)) >
act4:  rct_inst := rct_inst ▷ ({s} ◁ ran(rct_inst)) >
END

IS_OK:  extended ordinary >
REFINES
  IS_OK
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
  grd5:  susp_peers(s) = ∅ not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ RUN_4} >
END

FAIL_ACTIV:  extended ordinary >
REFINES
  FAIL_ACTIV
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem >
  grd5:  susp_peers(s) ≠ ∅ not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ FAIL_ACTIV_4} >
  act2:  run_peers(s) := run_peers(s) \ susp_peers(s) >
  act3:  susp_peers(s) := ∅ >
  act4:  fail_peers := fail_peers ∪ ({s} × susp_peers(s)) >
END

FAIL_CONFIGURE:  extended ordinary >
REFINES
  FAIL_CONFIGURE
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
  grd3:  card(run_peers(s)) < min_inst(s) not theorem >

```

```

    THEN
      act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_CONFIG_4} >
    END

  FAIL_IGNORE: extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
    grd3: card(run_peers(s)) ≥ min_inst(s) not theorem >
  THEN
    act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_IGN_4} >
  END

  IGNORE: extended ordinary >
  REFINES
    IGNORE
  ANY
    s >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: s ↦ FAIL_IGN_4 ∈ serviceState_4 not theorem >
  THEN
    act1: serviceState_4 := (serviceState_4 \ {s ↦ FAIL_IGN_4}) ∪
{s ↦ RUN_4} >
  END

  REDEPLOY_INST: extended ordinary >
  REFINES
    REDEPLOY_INST
  ANY
    s >
    dep >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: dep ⊆ PEERS not theorem >
    grd3: finite(dep) not theorem >
    grd4: dep ∩ run_peers(s) = ∅ not theorem >
    grd5: dep ∩ fail_peers[{s}] = ∅ not theorem >
    grd6: card(dep) = depl_inst(s) not theorem >
    grd7: card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd8: s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >

```

```

THEN
  act1:  dep_inst := dep_inst u ({s}×dep) >
END

REDEPLOY:  extended ordinary >
REFINES
  REDEPLOY
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
  grd6:  dep_inst[{s}] ≠ ∅ not theorem >
  grd4:  card(run_peers(s))+card(dep_inst[{s}]) ≥ min_inst(s) not
theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_CONFIG_4})
u {s ↦ DPL_4} >
  act2:  run_peers(s) := run_peers(s) u dep_inst[{s}] >
  act3:  dep_inst := {s} ◀ dep_inst >
END

HEAL:  extended ordinary >
REFINES
  HEAL
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ DPL_4 ∈ serviceState_4 not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ DPL_4}) u {s ↦
RUN_4} >
END

UNFAIL_PEER:  extended ordinary >
REFINES
  UNFAIL_PEER
ANY
  s >
  p >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  p ∈ PEERS not theorem >
  grd3:  s ↦ p ∈ fail_peers not theorem >
THEN
  act1:  fail_peers := fail_peers\{s ↦ p} >
END

```

```
MAKE_PEER_AVAIL:  extended ordinary >
  REFINES
    MAKE_PEER_AVAIL
  ANY
    p >
  WHERE
    grd1:  p ∈ PEERS not theorem >
    grd2:  p ∈ unav_peers not theorem >
  THEN
    act1:  unav_peers := unav_peers \ {p} >
  END
END
```

```

MACHINE
  M10    >
REFINES
  M09
SEES
  C08
VARIABLES
  serviceState_4  >
  run_peers       >
  susp_peers      >
  fail_peers      >
  dep_inst        >
  token_owner     >
  unav_peers      >
  susp_inst       >
  rec_inst        >instances that are tried to be recontacted
  rct_inst        >instances effectively recontacted after a try
  actv_inst       >instances activated by token owne
INVARIANTS
  inv1:  actv_inst ∈ PEERS ↔ (SERVICES×PEERS) not theorem >
  inv2:  ∀ s, i · s ∈ SERVICES ∧ i ∈ PEERS ⇒ finite(actv_inst[i][s])
not theorem >
  inv3:  ∀ ld, s · ld ∈ PEERS ∧ s ∈ SERVICES ∧ s ∈ dom(actv_inst[ld])
⇒ ld = token_owner(s) not theorem >
  inv4:  ∀ s, i · s ∈ SERVICES ∧ i ∈ PEERS ⇒ actv_inst[i][s] ∩
run_peers(s) = ∅ not theorem >
  inv5:  ∀ s, i · s ∈ SERVICES ∧ i ∈ PEERS ⇒ actv_inst[i][s] ∩
dep_inst[s] = ∅ not theorem >
  inv6:  ∀ s, i · s ∈ SERVICES ∧ i ∈ PEERS ⇒ actv_inst[i][s] ∩
fail_peers[s] = ∅ not theorem >
  inv7:  ∀ ld, s, stt · ld ∈ PEERS ∧ s ∈ SERVICES ∧ stt ∈ STATES_4 ∧ s ⇨
stt ∈ serviceState_4 ∧ ld = token_owner(s) ∧ stt ≠ FAIL_CONFIG_4 ⇒ actv_inst
[ld][s] = ∅ not theorem >
  inv8:  finite(actv_inst) not theorem >
EVENTS
  INITIALISATION:  extended ordinary >
  THEN
    act1:  serviceState_4 := InitState_4 >
    act2:  run_peers := InitSrvcPeers >
    act3:  susp_peers := ∅ >
    act4:  fail_peers := ∅ >
    act5:  dep_inst := ∅ >
    act6:  token_owner := init_tok >
    act7:  unav_peers := ∅ >
    act8:  susp_inst := ∅ >
    act10: rec_inst := ∅ >
    act11: rct_inst := ∅ >
    act12: actv_inst := ∅ >

```

END

MAKE\_PEER\_UNAVAIL: extended ordinary >  
 REFINES  
 MAKE\_PEER\_UNAVAIL  
 ANY  
 prs >  
 E >new values for token owner per service if needed  
 WHERE  
 grd1:  $prs \subseteq PEERS$  not theorem >  
 grd2:  $prs \not\subseteq unav\_peers$  not theorem >  
 grd3:  $E \in SERVICES \rightarrow PEERS$  not theorem >new value for token  
 owner per service if needed  
 grd4:  $\forall srv \cdot srv \in SERVICES \wedge token\_owner(srv) \notin prs \Rightarrow E$   
 (srv) = token\_owner(srv) not theorem >If the token owner of a service srv does  
 not belong to prs, the token owner is not changed  
 grd5:  $\forall srv \cdot srv \in SERVICES \wedge token\_owner(srv) \in prs \wedge srv \notin$   
 dom(susp\_peers)  $\Rightarrow E(srv) \in run\_peers(srv) \setminus (unav\_peers \cup prs \cup fail\_peers$   
 [{srv}]) not theorem >if the owner of the token for a service becomes  
 unavailable and the service is not suspicious,  
 then a new token owner among available peers is chosen  
 grd6:  $\forall srv \cdot srv \in SERVICES \wedge token\_owner(srv) \in prs \wedge srv \in$   
 dom(susp\_peers)  $\Rightarrow E(srv) \in run\_peers(srv) \setminus (unav\_peers \cup prs \cup susp\_peers(srv) \cup$   
 fail\_peers[{{srv}}]) not theorem >if the owner of the token for a service becomes  
 unavailable, and the service

possess suspicious instances, then a new token owner among available and not

suspicious peers is chosen

THEN  
 act1:  $unav\_peers := unav\_peers \cup prs$  >the peers in prs become  
 unavailable  
 act2:  $token\_owner := token\_owner \leftarrow E$  >new value for token owner  
 per service is given if needed  
 act3:  $susp\_inst := prs \leftarrow susp\_inst$  >the peers in prs can not  
 suspect instances anymore  
 act4:  $rec\_inst := prs \leftarrow rec\_inst$  >  
 act5:  $rct\_inst := prs \leftarrow rct\_inst$  >  
 act6:  $actv\_inst := prs \leftarrow actv\_inst$  >  
 END

SUSPECT\_INST: extended ordinary >  
 REFINES  
 SUSPECT\_INST

M10

```

ANY
  s    >a service s
  susp >suspicious instances
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: susp ⊆ PEERS not theorem >
  grd3: susp = run_peers(s) n unav_peers not theorem >instances
in susp are suspicious if the peers running them becomes unavailable
  grd4: s ∉ dom(susp_inst[{{token_owner(s)}}]) not theorem >the
member of susp have not yet been suspected for s by the token owner of s
  grd5: s ↦ RUN_4 ∈ serviceState_4 not theorem >the state of s
is OK
THEN
  act1: susp_inst := susp_inst ∪ ({{token_owner(s)}} × ({{s}}×susp))
>the members of susp become suspected instances for s by the token owner of s
END

FAIL: extended ordinary >
REFINES
  FAIL
ANY
  s    >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: s ↦ RUN_4 ∈ serviceState_4 not theorem >
  grd3: susp_inst[{{token_owner(s)}}][{{s}}] ≠ ∅ not theorem >
THEN
  act1: serviceState_4 := (serviceState_4 \ {{s ↦ RUN_4}}) ∪ {{s ↦
FAIL_4}} >
  act2: susp_peers(s) := susp_inst[{{token_owner(s)}}][{{s}}] >
  act3: susp_inst := susp_inst ▷ ({{s}} ◁ ran(susp_inst)) >
END

RECONTACT_INST_OK: extended ordinary >
REFINES
  RECONTACT_INST_OK
ANY
  s    >a service s
  i    >an instance i
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: i ∈ PEERS not theorem >
  grd3: s ↦ FAIL_4 ∈ serviceState_4 not theorem >the state of s
is SUSPICIOUS
  grd4: susp_peers(s) ≠ ∅ not theorem >the set of suspicious
peers for s is not empty
  grd5: i ∈ susp_peers(s) \ unav_peers not theorem >i is a
suspicious instance of s and is available (can be contacted)

```

M10

```

        grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
        grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ susp_peers(s) not
theorem >the token owner of s has not yet tried to recontact all the suspicious
instances of s
    THEN
        act1:  rec_inst := rec_inst ∪ {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
        act2:  rct_inst := rct_inst ∪ {token_owner(s) ↦ (s ↦ i)} >i is
recontacted by the token owner of s successfully
    END

RECONTACT_INST_K0:  extended ordinary >
    REFINES
        RECONTACT_INST_K0
    ANY
        s    >a service s
        i    >an instance i
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  i ∈ PEERS not theorem >
        grd3:  s ↦ FAIL_4 ∈ serviceState_4 not theorem >the state of s
is SUSPICIOUS
        grd4:  susp_peers(s) ≠ ∅ not theorem >the set of suspicious
peers for s is not empty
        grd5:  i ∈ susp_peers(s)∪nnav_peers not theorem >i is a
suspicious instance of s and is unavailable (can not be contacted)
        grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
        grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ susp_peers(s) not
theorem >the token owner of s has not yet tried to recontact all the suspicious
instances of s
    THEN
        act1:  rec_inst := rec_inst ∪ {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
    END

FAIL_DETECT:  extended ordinary >
    REFINES
        FAIL_DETECT
    ANY
        s    >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  s ↦ FAIL_4 ∈ serviceState_4 not theorem >
        grd5:  susp_peers(s) ≠ ∅ not theorem >
        grd8:  rec_inst[{token_owner(s)}][{s}] = susp_peers(s) not
theorem >

```



M10

```

    THEN
      act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_4}) ∪ {s ↦
FAIL_DETECT_4}
      >
      act2:  susp_peers(s) := susp_peers(s) \ rct_inst[{token_owner
(s)}][{s}]
      >
      act3:  rec_inst := rec_inst ▷ ({s} < ran(rec_inst))
      >
      act4:  rct_inst := rct_inst ▷ ({s} < ran(rct_inst))
      >
    END

IS_OK:  extended ordinary
REFINES
  IS_OK
ANY
  s
  >
WHERE
  grd1:  s ∈ SERVICES not theorem
  >
  grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem
  >
  grd5:  susp_peers(s) = ∅ not theorem
  >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ RUN_4}
  >
END

FAIL_ACTIV:  extended ordinary
REFINES
  FAIL_ACTIV
ANY
  s
  >
WHERE
  grd1:  s ∈ SERVICES not theorem
  >
  grd2:  s ↦ FAIL_DETECT_4 ∈ serviceState_4 not theorem
  >
  grd5:  susp_peers(s) ≠ ∅ not theorem
  >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_DETECT_4})
∪ {s ↦ FAIL_ACTIV_4}
  >
  act2:  run_peers(s) := run_peers(s) \ susp_peers(s)
  >
  act3:  susp_peers(s) := ∅
  >
  act4:  fail_peers := fail_peers ∪ ({s} × susp_peers(s))
  >
END

FAIL_CONFIGURE:  extended ordinary
REFINES
  FAIL_CONFIGURE
ANY
  s
  >
WHERE
  grd1:  s ∈ SERVICES not theorem
  >
  grd2:  s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem
  >

```

M10

```

    grd3:   card(run_peers(s)) < min_inst(s) not theorem >
  THEN
    act1:   serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_CONFIG_4} >
  END

  FAIL_IGNORE:   extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s >
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   s ↦ FAIL_ACTIV_4 ∈ serviceState_4 not theorem >
    grd3:   card(run_peers(s)) ≥ min_inst(s) not theorem >
  THEN
    act1:   serviceState_4 := (serviceState_4 \ {s ↦ FAIL_ACTIV_4}) ∪
{s ↦ FAIL_IGN_4} >
  END

  IGNORE:   extended ordinary >
  REFINES
    IGNORE
  ANY
    s >
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   s ↦ FAIL_IGN_4 ∈ serviceState_4 not theorem >
  THEN
    act1:   serviceState_4 := (serviceState_4 \ {s ↦ FAIL_IGN_4}) ∪
{s ↦ RUN_4} >
  END

  REDEPLOY_INSTC:   not extended ordinary >
  ANY
    s >a service s
    i >an instance i
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   i ∈ PEERS not theorem >
    grd3:   i ∉ run_peers(s) ∪ fail_peers[{s}] ∪ unav_peers ∪
dep_inst[{s}] not theorem >i does not run s, is not failed for s, is not
unavailable and is not already activated for s
    grd4:   token_owner(s) ↦ (s ↦ i) ∉ actv_inst not theorem >
    grd5:   s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
    grd6:   card(actv_inst[{token_owner(s)}][{s}]) < deplo_inst(s)
not theorem >
    grd7:   card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)

```

```

not theorem >
  THEN
    act1:  actv_inst := actv_inst u {token_owner(s) ↦ (s ↦ i)} >
  END

  REDEPLOY_INSTS:  not extended ordinary >
  REFINES
    REDEPLOY_INST
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd6:  card(actv_inst[{token_owner(s)}][{s}]) = deplo_inst(s)
not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd8:  s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
  WITH
    dep:  dep:=actv_inst[{token_owner(s)}][{s}] >
  THEN
    act1:  dep_inst := dep_inst u ({s}×actv_inst[{token_owner(s)}]
[ {s} ]) >
    act2:  actv_inst := actv_inst ▷ ({s} ◁ ran(actv_inst)) >
  END

  REDEPLOY:  not extended ordinary >
  REFINES
    REDEPLOY
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  s ↦ FAIL_CONFIG_4 ∈ serviceState_4 not theorem >
    grd7:  actv_inst[{token_owner(s)}][{s}] = ∅ not theorem >
    grd6:  dep_inst[{s}] ≠ ∅ not theorem >
    grd4:  card(run_peers(s))+card(dep_inst[{s}]) ≥ min_inst(s) not
theorem >
  THEN
    act1:  serviceState_4 := (serviceState_4 \ {s ↦ FAIL_CONFIG_4})
u {s ↦ DPL_4} >
    act2:  run_peers(s) := run_peers(s) u dep_inst[{s}] >
    act3:  dep_inst := {s} ◁ dep_inst >
  END

  HEAL:  extended ordinary >
  REFINES
    HEAL
  ANY

```

M10

```

s      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  s ↦ DPL_4 ∈ serviceState_4 not theorem >
THEN
  act1:  serviceState_4 := (serviceState_4 \ {s ↦ DPL_4}) ∪ {s ↦
RUN_4} >
END

UNFAIL_PEER:  extended ordinary >
REFINES
  UNFAIL_PEER
ANY
  s      >
  p      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  p ∈ PEERS not theorem >
  grd3:  s ↦ p ∈ fail_peers not theorem >
THEN
  act1:  fail_peers := fail_peers \ {s ↦ p} >
END

MAKE_PEER_AVAIL:  extended ordinary >
REFINES
  MAKE_PEER_AVAIL
ANY
  p      >
WHERE
  grd1:  p ∈ PEERS not theorem >
  grd2:  p ∈ unav_peers not theorem >
THEN
  act1:  unav_peers := unav_peers \ {p} >
END

END

```

M11

```

MACHINE
  M11 >
REFINES
  M10
SEES
  C08
VARIABLES
  run_peers >
  susp_peers >
  fail_peers >
  dep_inst >
  token_owner >
  unav_peers >
  susp_inst >
  rec_inst >instances that are tried to be recontacted
  rct_inst >instances effectively recontacted after a try
  actv_inst >instances activated by token owne
  i_state >
INVARIANTS
  inv1: i_state ∈ (PEERS × SERVICES) ↔ STATES_4 not theorem >
  inv2: ∀ s · s ∈ SERVICES ⇒ token_owner(s) ↦ s ∈ dom(i_state) not
theorem >
  gluing_state1: ∀ s, stt · s ∈ SERVICES ∧ stt ∈ STATES_4 ∧ s ↦ stt ∈
serviceState_4 ⇒ (token_owner(s) ↦ s) ↦ stt ∈ i_state not theorem >
  gluing_state2: ∀ s, stt · s ∈ SERVICES ∧ stt ∈ STATES_4 ∧ (token_owner
(s) ↦ s) ↦ stt ∈ i_state ⇒ s ↦ stt ∈ serviceState_4 not theorem >
  inv3: ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈ dom(i_state) ⇒ p
= token_owner(s) not theorem >
EVENTS
  INITIALISATION: not extended ordinary >
  THEN
    act2: run_peers := InitSrvcPeers >
    act3: susp_peers := ∅ >
    act4: fail_peers := ∅ >
    act5: dep_inst := ∅ >
    act6: token_owner := init_tok >
    act7: unav_peers := ∅ >
    act8: susp_inst := ∅ >
    act10: rec_inst := ∅ >
    act11: rct_inst := ∅ >
    act12: actv_inst := ∅ >
    act13: i_state := InitStatus >
  END

  MAKE_PEER_UNAVAIL: not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY

```

M11

```

    prs >
    E >new values for token owner per service if needed
    i_s >
WHERE
    grd1: prs ⊆ PEERS not theorem >
    grd2: prs ⊄ unav_peers not theorem >
    grd3: E ∈ SERVICES → PEERS not theorem >new value for token
owner per service if needed
    grd4: i_s ∈ (PEERS×SERVICES) ↔ STATES_4 not theorem >
    grd5: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs ⇒ E
(srv) = token_owner(srv) not theorem >If the token owner of a service srv does
not belong to prs, the token owner is not changed
    grd6: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧ srv ∉
dom(susp_peers) ⇒ E(srv) ∈ run_peers(srv)\(unav_peers ∪ prs ∪ fail_peers
[{:srv}]) not theorem >if the owner of the token for a service becomes
unavailable and the service is not suspicious,

then a new token owner among available peers is chosen
    grd7: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧ srv ∈
dom(susp_peers) ⇒ E(srv) ∈ run_peers(srv)\(unav_peers ∪ prs ∪ fail_peers[{:srv}]
∪ susp_peers(srv)) not theorem >if the owner of the token for a service becomes
unavailable, and the service

possess suspicious instances, then a new token owner among available and not

suspicious peers is chosen
    grd8: ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ↦ s ∈ dom(i_s) ⇒
p = E(s) not theorem >
    grd9: ∀ srv · srv ∈ SERVICES ⇒ (E(srv) ↦ srv) ↦ i_state
(token_owner(srv) ↦ srv) ∈ i_s not theorem >
THEN
    act1: unav_peers := unav_peers ∪ prs >the peers in prs become
unavailable
    act2: token_owner := token_owner ◀ E >new value for token owner
per service is given if needed
    act3: susp_inst := prs ◀ susp_inst >the peers in prs can not
suspect instances anymore
    act4: rec_inst := prs ◀ rec_inst >the peers in prs can not try
to recontact instances anymore
    act5: rct_inst := prs ◀ rct_inst >the peers in prs can not
recontact instances anymore
    act6: actv_inst := prs ◀ actv_inst >
    act7: i_state := i_s >
END

```

M11

```

SUSPECT_INST:  not extended ordinary >
  REFINES
    SUSPECT_INST
  ANY
    s      >a service s
    susp   >suspicious instances
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  susp ⊆ PEERS not theorem >
    grd3:  susp = run_peers(s) n unav_peers not theorem >instances
in susp are suspicious if the peers running them becomes unavailable
    grd4:  s ∉ dom(susp_inst[{token_owner(s)}]) not theorem >the
member of susp have not yet been suspected for s by the token owner of s
    grd5:  i_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
  THEN
    act1:  susp_inst := susp_inst ∪ ({token_owner(s)} × ({s}×susp))
>the members of susp become suspected instances for s by the token owner of s
  END

FAIL:  not extended ordinary >
  REFINES
    FAIL
  ANY
    s      >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = RUN_4 not theorem >
    grd3:  susp_inst[{token_owner(s)}][{s}] ≠ ∅ not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_4 >
    act2:  susp_peers(s) := susp_inst[{token_owner(s)}][{s}] >
    act3:  susp_inst := susp_inst ▷ ({s} ◁ ran(susp_inst)) >
  END

RECONTACT_INST_OK:  not extended ordinary >
  REFINES
    RECONTACT_INST_OK
  ANY
    s      >a service s
    i      >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  susp_peers(s) ≠ ∅ not theorem >the set of suspicious
peers for s is not empty

```

M11

```

    grd5:  i ∈ susp_peers(s)\unav_peers not theorem >i is a
suspicious instance of s and is available (can be contacted)
    grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
    grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ susp_peers(s) not
theorem >the token owner of s has not yet tried to recontact all the suspicious
instances of s
    THEN
        act1:  rec_inst := rec_inst ∪ {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
        act2:  rct_inst := rct_inst ∪ {token_owner(s) ↦ (s ↦ i)} >i is
recontacted by the token owner of s successfully
    END

RECONTACT_INST_K0:  not extended ordinary >
REFINES
    RECONTACT_INST_K0
    ANY
    s      >a service s
    i      >an instance i
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  i ∈ PEERS not theorem >
        grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
        grd4:  susp_peers(s) ≠ ∅ not theorem >the set of suspicious
peers for s is not empty
        grd5:  i ∈ susp_peers(s)\unav_peers not theorem >i is a
suspicious instance of s and is unavailable (can not be contacted)
        grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
        grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ susp_peers(s) not
theorem >the token owner of s has not yet tried to recontact all the suspicious
instances of s
    THEN
        act1:  rec_inst := rec_inst ∪ {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
    END

FAIL_DETECT:  not extended ordinary >
REFINES
    FAIL_DETECT
    ANY
    s      >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
        grd5:  susp_peers(s) ≠ ∅ not theorem >

```



M11

```

theorem >
    grd8:  rec_inst[{token_owner(s)}][{s}] = susp_peers(s) not
    THEN
    act1:  i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 >
    act2:  susp_peers(s) = susp_peers(s) \ rct_inst[{token_owner
(s)}][{s}] >
    act3:  rec_inst = rec_inst ▷ ({s} < ran(rec_inst)) >
    act4:  rct_inst = rct_inst ▷ ({s} < ran(rct_inst)) >
    END

IS_OK:  not extended ordinary >
    REFINES
        IS_OK
    ANY
        s >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
        >
        grd5:  susp_peers(s) = ∅ not theorem >
    THEN
        act1:  i_state(token_owner(s) ↦ s) = RUN_4 >
    END

FAIL_ACTIV:  not extended ordinary >
    REFINES
        FAIL_ACTIV
    ANY
        s >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
        >
        grd5:  susp_peers(s) ≠ ∅ not theorem >
    THEN
        act1:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 >
        act2:  run_peers(s) = run_peers(s) \ susp_peers(s) >
        act3:  susp_peers(s) = ∅ >
        act4:  fail_peers = fail_peers ∪ ({s} × susp_peers(s)) >
    END

FAIL_CONFIGURE:  not extended ordinary >
    REFINES
        FAIL_CONFIGURE
    ANY
        s >
    WHERE
        grd1:  s ∈ SERVICES not theorem >

```

M11

```

    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) < min_inst(s) not theorem >
THEN
    act1:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 >
END

FAIL_IGNORE:  not extended ordinary >
REFINES
    FAIL_IGNORE
ANY
    s      >
WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) ≥ min_inst(s) not theorem >
THEN
    act1:  i_state(token_owner(s) ↦ s) = FAIL_IGN_4 >
END

IGNORE:  not extended ordinary >
REFINES
    IGNORE
ANY
    s      >
WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem >
THEN
    act1:  i_state(token_owner(s) ↦ s) = RUN_4 >
END

REDEPLOY_INSTC:  not extended ordinary >
REFINES
    REDEPLOY_INSTC
ANY
    s      >a service s
    i      >an instance i
WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i ∉ run_peers(s) ∪ fail_peers[{s}] ∪ unav_peers ∪
dep_inst[{s}] not theorem >i does not run s, is not failed for s, is not
unavailable and is not already activated for s
    grd4:  token_owner(s) ↦ (s ↦ i) ∉ actv_inst not theorem >
    grd5:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    grd6:  card(actv_inst[{token_owner(s)}][{s}]) < deplo_inst(s)
not theorem >

```

M11

```

not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
    THEN
    act1:  actv_inst := actv_inst u {token_owner(s) ↦ (s ↦ i)} >
    END

REDEPLOY_INSTS:  not extended ordinary >
    REFINES
    REDEPLOY_INSTS
    ANY
    s >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd6:  card(actv_inst[{token_owner(s)}][{s}]) = deplo_inst(s)
not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd8:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    THEN
    act1:  dep_inst := dep_inst u ({s} × actv_inst[{token_owner(s)}]
[{s}]) >
    act2:  actv_inst := actv_inst ▷ ({s} ◁ ran(actv_inst)) >
    END

REDEPLOY:  not extended ordinary >
    REFINES
    REDEPLOY
    ANY
    s >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    grd7:  actv_inst[{token_owner(s)}][{s}] = ∅ not theorem >
    grd6:  dep_inst[{s}] ≠ ∅ not theorem >
    grd4:  card(run_peers(s)) + card(dep_inst[{s}]) ≥ min_inst(s) not
theorem >
    THEN
    act1:  i_state(token_owner(s) ↦ s) := DPL_4 >
    act2:  run_peers(s) := run_peers(s) u dep_inst[{s}] >
    act3:  dep_inst := {s} ◁ dep_inst >
    END

HEAL:  not extended ordinary >
    REFINES
    HEAL
    ANY

```

## M11

```

    s      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i_state(token_owner(s) ↦ s) = DPL_4 not theorem >
THEN
  act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
END

UNFAIL_PEER:  extended ordinary >
REFINES
  UNFAIL_PEER
ANY
  s      >
  p      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  p ∈ PEERS not theorem >
  grd3:  s ↦ p ∈ fail_peers not theorem >
THEN
  act1:  fail_peers := fail_peers \ {s ↦ p} >
END

MAKE_PEER_AVAIL:  extended ordinary >
REFINES
  MAKE_PEER_AVAIL
ANY
  p      >
WHERE
  grd1:  p ∈ PEERS not theorem >
  grd2:  p ∈ unav_peers not theorem >
THEN
  act1:  unav_peers := unav_peers \ {p} >
END

END
```

## M12

```

MACHINE
  M12    >
REFINES
  M11
SEES
  C08
VARIABLES
  run_peers    >
  suspc_peers  >
  fail_peers   >
  dep_inst     >
  token_owner  >
  unav_peers   >
  susp_inst    >
  rec_inst     >instances that are tried to be recontacted
  rct_inst     >instances effectively recontacted after a try
  actv_inst    >instances activated by token owne
  i_state     >
INVARIANTS
  inv1:  suspc_peers ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈ dom(suspc_peers)
⇒ p = token_owner(s) not theorem >
  inv3:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p = token_owner(s) ⇒ (p ↦
s) ∈ dom(suspc_peers) not theorem >
  gluing_tok_own1:  ∀ s · s ∈ SERVICES ∧ s ∈ dom(susp_peers) ⇒
susp_peers(s) = suspc_peers(token_owner(s) ↦ s) not theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
  THEN
    act2:  run_peers := InitSrvcPeers >
    act3:  suspc_peers := InitSuspPeers >
    act4:  fail_peers := ∅ >
    act5:  dep_inst := ∅ >
    act6:  token_owner := init_tok >
    act7:  unav_peers := ∅ >
    act8:  susp_inst := ∅ >
    act10: rec_inst := ∅ >
    act11: rct_inst := ∅ >
    act12: actv_inst := ∅ >
    act13: i_state := InitStatus >
  END

  MAKE_PEER_UNAVAIL:  not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY
    prs    >
    E      >new values for token owner per service if needed

```

```

i_s >
p_s >
WHERE
  grd1: prs ⊆ PEERS not theorem >
  grd2: prs ⊄ unav_peers not theorem >
  grd3: E ∈ SERVICES → PEERS not theorem >new value for token
owner per service if needed
  grd4: i_s ∈ (PEERS×SERVICES) ↔ STATES_4 not theorem >
  grd5: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs ⇒ E
(srv) = token_owner(srv) not theorem >If the token owner of a service srv does
not belong to prs, the token owner is not changed
  grd6: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧
token_owner(srv) ↦ srv ∉ dom(suspc_peers) ⇒ E(srv) ∈ run_peers(srv)\(unav_peers
u prs u fail_peers[{srv}]) not theorem >if the owner of the token for a service
becomes unavailable and the service is not suspicious,

then a new token owner among available peers is chosen
  grd7: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧
token_owner(srv) ↦ srv ∈ dom(suspc_peers) ⇒ E(srv) ∈ run_peers(srv)\(unav_peers
u prs u fail_peers[{srv}] u suspc_peers(token_owner(srv) ↦ srv)) not theorem
>if the owner of the token for a service becomes unavailable, and the service

possess suspicious instances, then a new token owner among available and not

suspicious peers is chosen
  grd8: ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ↦ s ∈ dom(i_s) ⇒
p = E(s) not theorem >
  grd9: ∀ srv · srv ∈ SERVICES ⇒ (E(srv) ↦ srv) ↦ i_state
(token_owner(srv) ↦ srv) ∈ i_s not theorem >
  grd10: p_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
  grd11: ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ↦ s ∈ dom(p_s) ⇒
p = E(s) not theorem >
  grd12: ∀ srv · srv ∈ SERVICES ⇒ (E(srv) ↦ srv) ↦ suspc_peers
(token_owner(srv) ↦ srv) ∈ p_s not theorem >
THEN
  act1: unav_peers := unav_peers u prs >the peers in prs become
unavailable
  act2: token_owner := token_owner ◀ E >new value for token owner
per service is given if needed
  act3: susp_inst := prs ◀ susp_inst >the peers in prs can not
suspect instances anymore
  act4: rec_inst := prs ◀ rec_inst >the peers in prs can not try
to recontact instances anymore
  act5: rct_inst := prs ◀ rct_inst >the peers in prs can not
recontact instances anymore

```

M12

```

act6:  actv_inst := prs < actv_inst >
act7:  i_state := i_s >
act8:  suspc_peers := p_s >
END

SUSPECT_INST:  extended ordinary >
REFINES
  SUSPECT_INST
ANY
  s      >a service s
  susp   >suspicious instances
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  susp ⊆ PEERS not theorem >
  grd3:  susp = run_peers(s) n unav_peers not theorem >instances
in susp are suspicious if the peers running them becomes unavailable
  grd4:  s ∉ dom(susp_inst[{token_owner(s)}]) not theorem >the
member of susp have not yet been suspected for s by the token owner of s
  grd5:  i_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
THEN
  act1:  susp_inst := susp_inst ∪ ({token_owner(s)} × ({s}×susp))
>the members of susp become suspected instances for s by the token owner of s
END

FAIL:  not extended ordinary >
REFINES
  FAIL
ANY
  s      >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i_state(token_owner(s) ↦ s) = RUN_4 not theorem >
  grd3:  susp_inst[{token_owner(s)}][{s}] ≠ ∅ not theorem >
THEN
  act1:  i_state(token_owner(s) ↦ s) := FAIL_4 >
  act2:  suspc_peers(token_owner(s) ↦ s) := susp_inst[{token_owner
(s)}][{s}] >
  act3:  susp_inst := susp_inst ▷ ({s} < ran(susp_inst)) >
END

RECONTACT_INST_OK:  not extended ordinary >
REFINES
  RECONTACT_INST_OK
ANY
  s      >a service s
  i      >an instance i
WHERE

```

M12

```

    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
    grd5:  i ∈ suspc_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
    grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
    grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ suspc_peers
(token_owner(s) ↦ s) not theorem >the token owner of s has not yet tried to
recontact all the suspicious instances of s
    THEN
        act1:  rec_inst = rec_inst ∪ {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
        act2:  rct_inst = rct_inst ∪ {token_owner(s) ↦ (s ↦ i)} >i is
recontacted by the token owner of s successfully
    END

RECONTACT_INST_K0:  not extended ordinary >
REFINES
    RECONTACT_INST_K0
    ANY
    s  >a service s
    i  >an instance i
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  i ∈ PEERS not theorem >
        grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
        grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
        grd5:  i ∈ suspc_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
        grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
        grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ suspc_peers
(token_owner(s) ↦ s) not theorem >the token owner of s has not yet tried to
recontact all the suspicious instances of s
        THEN
            act1:  rec_inst = rec_inst ∪ {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
        END

FAIL_DETECT:  not extended ordinary >
REFINES

```



```

    FAIL_DETECT
  ANY
    s >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
    grd5: suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd8: rec_inst[{token_owner(s)}][{s}] = suspc_peers
(token_owner(s) ↦ s) not theorem >
  THEN
    act1: i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 >
    act2: suspc_peers(token_owner(s) ↦ s) = suspc_peers
(token_owner(s) ↦ s) \ rct_inst[{token_owner(s)}][{s}] >
    act3: rec_inst = rec_inst ▷ ({s} ◁ ran(rec_inst)) >
    act4: rct_inst = rct_inst ▷ ({s} ◁ ran(rct_inst)) >
  END

IS_OK: not extended ordinary >
  REFINES
    IS_OK
  ANY
    s >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
>
    grd5: suspc_peers(token_owner(s) ↦ s) = ∅ not theorem >
  THEN
    act1: i_state(token_owner(s) ↦ s) = RUN_4 >
  END

FAIL_ACTIV: not extended ordinary >
  REFINES
    FAIL_ACTIV
  ANY
    s >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
>
    grd5: suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
  THEN
    act1: i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 >
    act2: run_peers(s) = run_peers(s) \ suspc_peers(token_owner(s)
↦ s) >
    act3: fail_peers = fail_peers ∪ ({s} × suspc_peers(token_owner
(s) ↦ s)) >
    act4: suspc_peers(token_owner(s) ↦ s) = ∅ >

```

```

END

FAIL_CONFIGURE:    extended ordinary >
  REFINES
    FAIL_CONFIGURE
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) < min_inst(s) not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_CONFIG_4 >
  END

FAIL_IGNORE:      extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) ≥ min_inst(s) not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_IGN_4 >
  END

IGNORE:           extended ordinary >
  REFINES
    IGNORE
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
  END

REDEPLOY_INSTC:  extended ordinary >
  REFINES
    REDEPLOY_INSTC
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >

```

M12

```

    grd2:  i ∈ PEERS not theorem >
    grd3:  i ∉ run_peers(s) ∪ fail_peers[{s}] ∪ unav_peers ∪
dep_inst[{s}] not theorem > i does not run s, is not failed for s, is not
unavailable and is not already activated for s
    grd4:  token_owner(s) ↦ (s ↦ i) ∉ actv_inst not theorem >
    grd5:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    not theorem >
    grd6:  card(actv_inst[{token_owner(s)}][{s}]) < deplo_inst(s)
not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
    THEN
    act1:  actv_inst := actv_inst ∪ {token_owner(s) ↦ (s ↦ i)} >
    END

REDEPLOY_INSTS:  extended ordinary >
    REFINES
    REDEPLOY_INSTS
    ANY
    s >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    not theorem >
    grd6:  card(actv_inst[{token_owner(s)}][{s}]) = deplo_inst(s)
not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd8:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    THEN
    act1:  dep_inst := dep_inst ∪ ({s} × actv_inst[{token_owner(s)}]
[{s}]) >
    act2:  actv_inst := actv_inst ▷ ({s} < ran(actv_inst)) >
    END

REDEPLOY:  extended ordinary >
    REFINES
    REDEPLOY
    ANY
    s >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    grd7:  actv_inst[{token_owner(s)}][{s}] = ∅ not theorem >
    grd6:  dep_inst[{s}] ≠ ∅ not theorem >
    not theorem >
    grd4:  card(run_peers(s)) + card(dep_inst[{s}]) ≥ min_inst(s) not
theorem >
    THEN

```

## M12

```
act1: i_state(token_owner(s) ↦ s) := DPL_4 >
act2: run_peers(s) := run_peers(s) u dep_inst[{s}] >
act3: dep_inst := {s} ◀ dep_inst >
END

HEAL: extended ordinary >
REFINES
  HEAL
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: i_state(token_owner(s) ↦ s) = DPL_4 not theorem >
THEN
  act1: i_state(token_owner(s) ↦ s) := RUN_4 >
END

UNFAIL_PEER: extended ordinary >
REFINES
  UNFAIL_PEER
ANY
  s >
  p >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: p ∈ PEERS not theorem >
  grd3: s ↦ p ∈ fail_peers not theorem >
THEN
  act1: fail_peers := fail_peers \ {s ↦ p} >
END

MAKE_PEER_AVAIL: extended ordinary >
REFINES
  MAKE_PEER_AVAIL
ANY
  p >
WHERE
  grd1: p ∈ PEERS not theorem >
  grd2: p ∈ unav_peers not theorem >
THEN
  act1: unav_peers := unav_peers \ {p} >
END

END
```

M13

```

MACHINE
  M13    >
REFINES
  M12
SEES
  C08
VARIABLES
  run_peers    >
  suspc_peers  >
  fail_peers   >
  dep_inst     >
  token_owner  >
  unav_peers   >
  suspc_inst   >
  rec_inst     >instances that are tried to be recontacted
  rct_inst     >instances effectively recontacted after a try
  actv_inst    >instances activated by token owne
  i_state     >
INVARIANTS
  inv1:  suspc_inst ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈ dom(suspc_inst) ⇒
p = token_owner(s) not theorem >
  inv3:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p = token_owner(s) ⇒ (p ↦
s) ∈ dom(suspc_inst) not theorem >
  gluing_tok_own1:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈ dom
(suspc_inst) ⇒ susp_inst[{p}][{s}] = suspc_inst(p ↦ s) not theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
  THEN
    act2:  run_peers := InitSrvcPeers >
    act3:  suspc_peers := InitSuspPeers >
    act4:  fail_peers := ∅ >
    act5:  dep_inst := ∅ >
    act6:  token_owner := init_tok >
    act7:  unav_peers := ∅ >
    act8:  suspc_inst := InitSuspPeers >
    act10: rec_inst := ∅ >
    act11: rct_inst := ∅ >
    act12: actv_inst := ∅ >
    act13: i_state := InitStatus >
  END

  MAKE_PEER_UNAVAIL:  not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY
    prs    >
    E      >new values for token owner per service if needed

```

```

i_s >
p_s >
s_i >
WHERE
  grd1: prs ⊆ PEERS not theorem >
  grd2: prs ⊄ unav_peers not theorem >
  grd3: E ∈ SERVICES → PEERS not theorem >new value for token
owner per service if needed
  grd4: i_s ∈ (PEERS×SERVICES) ↔ STATES_4 not theorem >
  grd5: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs ⇒ E
(srv) = token_owner(srv) not theorem >If the token owner of a service srv does
not belong to prs, the token owner is not changed
  grd6: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧
token_owner(srv) ↦ srv ∉ dom(suspc_peers) ⇒ E(srv) ∈ run_peers(srv)\(unav_peers
u prs u fail_peers[{srv}]) not theorem >if the owner of the token for a service
becomes unavailable and the service is not suspicious,

then a new token owner among available peers is chosen
  grd7: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ∧
token_owner(srv) ↦ srv ∈ dom(suspc_peers) ⇒ E(srv) ∈ run_peers(srv)\(unav_peers
u prs u fail_peers[{srv}] u suspc_peers(token_owner(srv) ↦ srv)) not theorem
>if the owner of the token for a service becomes unavailable, and the service

possess suspicious instances, then a new token owner among available and not

suspicious peers is chosen
  grd8: ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ↦ s ∈ dom(i_s) ⇒
p = E(s) not theorem >
  grd9: ∀ srv · srv ∈ SERVICES ⇒ (E(srv) ↦ srv) ↦ i_state
(token_owner(srv) ↦ srv) ∈ i_s not theorem >
  grd10: p_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
  grd11: ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ↦ s ∈ dom(p_s) ⇒
p = E(s) not theorem >
  grd12: ∀ srv · srv ∈ SERVICES ⇒ (E(srv) ↦ srv) ↦ suspc_peers
(token_owner(srv) ↦ srv) ∈ p_s not theorem >
  grd13: s_i ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
  grd14: ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p ↦ s ∈ dom(s_i) ⇒
p = E(s) not theorem >
  grd15: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs ⇒ (E
(srv) ↦ srv) ↦ suspc_inst(E(srv) ↦ srv) ∈ s_i not theorem >
  grd16: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ⇒ (E
(srv) ↦ srv) ↦ ∅ ∈ s_i not theorem >
THEN
  act1: unav_peers = unav_peers u prs >the peers in prs become
unavailable

```

M13

act2: token\_owner := token\_owner ◀ E ›new value for token owner  
per service is given if needed  
act3: rec\_inst := prs ◀ rec\_inst ›the peers in prs can not try  
to recontact instances anymore  
act4: rct\_inst := prs ◀ rct\_inst ›the peers in prs can not  
recontact instances anymore  
act5: actv\_inst := prs ◀ actv\_inst ›  
act6: i\_state := i\_s ›  
act7: suspc\_peers := p\_s ›  
act8: suspc\_inst := s\_i ›  
END

SUSPECT\_INST: not extended ordinary ›  
REFINES  
SUSPECT\_INST  
ANY  
s ›a service s  
susp ›suspicious instances  
WHERE  
grd1: s ∈ SERVICES not theorem ›  
grd2: susp ⊆ PEERS not theorem ›  
grd3: susp = run\_peers(s) n unav\_peers not theorem ›instances  
in susp are suspicious if the peers running them becomes unavailable  
grd4: suspc\_inst(token\_owner(s) ↦ s) = ∅ not theorem ›the  
member of susp have not yet been suspected for s by the token owner of s  
grd5: i\_state(token\_owner(s) ↦ s) = RUN\_4 not theorem ›the  
state of s is OK  
grd6: susp ≠ ∅ not theorem ›  
THEN  
act1: suspc\_inst(token\_owner(s) ↦ s) := susp ›the members of  
susp become suspected instances for s by the token owner of s  
END

FAIL: not extended ordinary ›  
REFINES  
FAIL  
ANY  
s ›  
WHERE  
grd1: s ∈ SERVICES not theorem ›  
grd2: i\_state(token\_owner(s) ↦ s) = RUN\_4 not theorem ›  
grd3: suspc\_inst(token\_owner(s) ↦ s) ≠ ∅ not theorem ›  
THEN  
act1: i\_state(token\_owner(s) ↦ s) := FAIL\_4 ›  
act2: suspc\_peers(token\_owner(s) ↦ s) := suspc\_inst(token\_owner  
(s) ↦ s) ›  
act3: suspc\_inst(token\_owner(s) ↦ s) := ∅ ›  
END

```

RECONTACT_INST_OK:    extended ordinary >
  REFINES
    RECONTACT_INST_OK
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
    grd5:  i ∈ suspc_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
    grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
    grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ suspc_peers
(token_owner(s) ↦ s) not theorem >the token owner of s has not yet tried to
recontact all the suspicious instances of s
  THEN
    act1:  rec_inst := rec_inst ∪ {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
    act2:  rct_inst := rct_inst ∪ {token_owner(s) ↦ (s ↦ i)} >i is
recontacted by the token owner of s successfully
  END

RECONTACT_INST_K0:    extended ordinary >
  REFINES
    RECONTACT_INST_K0
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
    grd5:  i ∈ suspc_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
    grd6:  token_owner(s) ↦ (s ↦ i) ∉ rec_inst not theorem >the
token owner of s has not yet tried to recontact i
    grd7:  rec_inst[{token_owner(s)}][{s}] ⊂ suspc_peers
(token_owner(s) ↦ s) not theorem >the token owner of s has not yet tried to

```



M13

```

recontact all the suspicious instances of s
  THEN
    act1:  rec_inst := rec_inst u {token_owner(s) ↦ (s ↦ i)} >the
token owner of s has tried to recontact i
  END

FAIL_DETECT:  extended ordinary >
  REFINES
    FAIL_DETECT
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
    grd5:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd8:  rec_inst[{token_owner(s)}][{s}] = suspc_peers
(token_owner(s) ↦ s) not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_DETECT_4 >
    act2:  suspc_peers(token_owner(s) ↦ s) := suspc_peers
(token_owner(s) ↦ s) \ rct_inst[{token_owner(s)}][{s}] >
    act3:  rec_inst := rec_inst ▷ ({s} ◁ ran(rec_inst)) >
    act4:  rct_inst := rct_inst ▷ ({s} ◁ ran(rct_inst)) >
  END

IS_OK:  extended ordinary >
  REFINES
    IS_OK
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
>
    grd5:  suspc_peers(token_owner(s) ↦ s) = ∅ not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
  END

FAIL_ACTIV:  extended ordinary >
  REFINES
    FAIL_ACTIV
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
>

```

M13

```

    grd5:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_ACTIV_4 >
    act2:  run_peers(s) := run_peers(s) \ suspc_peers(token_owner(s)
↦ s) >
    act3:  fail_peers := fail_peers ∪ ({s}×suspc_peers(token_owner
(s) ↦ s)) >
    act4:  suspc_peers(token_owner(s) ↦ s) := ∅ >
  END

FAIL_CONFIGURE:  extended ordinary >
  REFINES
    FAIL_CONFIGURE
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) < min_inst(s) not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_CONFIG_4 >
  END

FAIL_IGNORE:  extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) ≥ min_inst(s) not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_IGN_4 >
  END

IGNORE:  extended ordinary >
  REFINES
    IGNORE
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
  END

```

M13

```

REDEPLOY_INSTC:    extended ordinary >
  REFINES
    REDEPLOY_INSTC
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i ∉ run_peers(s) ∪ fail_peers[{s}] ∪ unav_peers ∪
dep_inst[{s}] not theorem >i does not run s, is not failed for s, is not
unavailable and is not already activated for s
    grd4:  token_owner(s) ↦ (s ↦ i) ∉ actv_inst not theorem >
    grd5:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    grd6:  card(actv_inst[{token_owner(s)}][{s}]) < deplo_inst(s)
not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
  THEN
    act1:  actv_inst := actv_inst ∪ {token_owner(s) ↦ (s ↦ i)} >
  END

REDEPLOY_INSTS:    extended ordinary >
  REFINES
    REDEPLOY_INSTS
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd6:  card(actv_inst[{token_owner(s)}][{s}]) = deplo_inst(s)
not theorem >
    grd7:  card(dep_inst[{s}]) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd8:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
  THEN
    act1:  dep_inst := dep_inst ∪ ({s} × actv_inst[{token_owner(s)}]
[{s}]) >
    act2:  actv_inst := actv_inst ▷ ({s} ◁ ran(actv_inst)) >
  END

REDEPLOY:    extended ordinary >
  REFINES
    REDEPLOY
  ANY
    s    >
  WHERE

```

M13

```

    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    grd7:  actv_inst[{token_owner(s)}][{s}]=∅ not theorem >
    grd6:  dep_inst[{s}] ≠ ∅ not theorem >
    grd4:  card(run_peers(s))+card(dep_inst[{s}]) ≥ min_inst(s) not
theorem >
    THEN
    act1:  i_state(token_owner(s) ↦ s) := DPL_4 >
    act2:  run_peers(s) := run_peers(s) ∪ dep_inst[{s}] >
    act3:  dep_inst := {s} ◀ dep_inst >
    END

HEAL:    extended ordinary >
    REFINES
        HEAL
    ANY
        s >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = DPL_4 not theorem >
    THEN
    act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
    END

UNFAIL_PEER:    extended ordinary >
    REFINES
        UNFAIL_PEER
    ANY
        s >
        p >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  p ∈ PEERS not theorem >
    grd3:  s ↦ p ∈ fail_peers not theorem >
    THEN
    act1:  fail_peers := fail_peers \ {s ↦ p} >
    END

MAKE_PEER_AVAIL:    extended ordinary >
    REFINES
        MAKE_PEER_AVAIL
    ANY
        p >
    WHERE
    grd1:  p ∈ PEERS not theorem >
    grd2:  p ∈ unav_peers not theorem >
    THEN

```

M13

```
act1: unav_peers := unav_peers \ {p} >  
END
```

```
END
```

```

MACHINE
  M14    >
REFINES
  M13
SEES
  C08
VARIABLES
  run_peers    >
  suspc_peers  >
  failr_peers  >
  dep_instc    >
  token_owner  >
  unav_peers   >
  suspc_inst   >
  rect_inst    >instances that are tried to be recontacted
  rctt_inst    >instances effectively recontacted after a try
  actv_inst    >instances activated by token owne
  i_state     >
INVARIANTS
  inv1:  rect_inst ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈ dom(rect_inst) ⇒
p = token_owner(s) not theorem >
  inv3:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p = token_owner(s) ⇒ (p ↦
s) ∈ dom(rect_inst) not theorem >
  gluing_tok_own_recl:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈
dom(rect_inst) ⇒ rec_inst[{p}][{s}] = rect_inst(p ↦ s) not theorem >
  inv4:  rctt_inst ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv5:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈ dom(rctt_inst) ⇒
p = token_owner(s) not theorem >
  inv6:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p = token_owner(s) ⇒ (p ↦
s) ∈ dom(rctt_inst) not theorem >
  gluing_tok_own_rct1:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈
dom(rctt_inst) ⇒ rct_inst[{p}][{s}] = rctt_inst(p ↦ s) not theorem >
  inv7:  failr_peers ∈ SERVICES → ℙ(PEERS) not theorem >
  gluing_fail_1:  ∀ s · s ∈ SERVICES ⇒ fail_peers[{s}] = failr_peers(s)
not theorem >
  inv8:  dep_instc ∈ SERVICES → ℙ(PEERS) not theorem >
  gluing_act_1:  ∀ s · s ∈ SERVICES ⇒ dep_inst[{s}] = dep_instc(s) not
theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
  THEN
    act2:  run_peers = InitSrvcPeers >
    act3:  suspc_peers = InitSuspPeers >
    act4:  failr_peers = InitFail >
    act5:  dep_instc = InitFail >
    act6:  token_owner = init_tok >
    act7:  unav_peers = ∅ >

```

M14

```

act8:  suspc_inst := InitSuspPeers >
act10: rect_inst := InitSuspPeers >
act11: rctt_inst := InitSuspPeers >
act12: actv_inst := ∅ >
act13: i_state := InitStatus >

```

END

MAKE\_PEER\_UNAVAIL: not extended ordinary >

REFINES

MAKE\_PEER\_UNAVAIL

ANY

```

prs >
E >new values for token owner per service if needed
i_s >
p_s >
s_i >
rc_s >
rt_s >

```

WHERE

```

grd1: prs ⊆ PEERS not theorem >
grd2: prs ⊄ unav_peers not theorem >
grd3: E ∈ SERVICES → PEERS not theorem >new value for token
owner per service if needed
grd4: i_s ∈ (PEERS×SERVICES) ↔ STATES_4 not theorem >
grd5: p_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd6: s_i ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd7: rt_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd8: rc_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd9: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs ⇒ E
(srv) = token_owner(srv) not theorem >If the token owner of a service srv does
not belong to prs, the token owner is not changed
grd10: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs ⇒ E
(srv) ∈ run_peers(srv) \ (unav_peers ∪ prs ∪ failr_peers(srv) ∪ suspc_peers
(token_owner(srv) ↦ srv)) not theorem >if the owner of the token for a service
becomes unavailable, and the service

```

possess suspicious instances, then a new token owner among available and not

suspicious peers is chosen

```

grd11: dom(i_s) = E~ ∧ dom(p_s) = dom(i_s) ∧ dom(s_i) = dom
(i_s) ∧ dom(rc_s) = dom(i_s) ∧ dom(rt_s) = dom(i_s) not theorem >
grd12: ∀ srv · srv ∈ SERVICES ⇒ ((E(srv) ↦ srv) ↦ i_state
(token_owner(srv) ↦ srv) ∈ i_s ∧ (E(srv) ↦ srv) ↦ suspc_peers(token_owner(srv) ↦
srv) ∈ p_s) not theorem >
grd13: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs ⇒ ((E
(srv) ↦ srv) ↦ suspc_inst(E(srv) ↦ srv) ∈ s_i) ∧ ((E(srv) ↦ srv) ↦ rctt_inst(E

```

M14

$(\text{srv}) \mapsto \text{srv} \in \text{rt\_s} \wedge ((\text{E}(\text{srv}) \mapsto \text{srv}) \mapsto \text{rect\_inst}(\text{E}(\text{srv}) \mapsto \text{srv}) \in \text{rc\_s})$  not theorem >

grd14:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs} \Rightarrow ((\text{E}(\text{srv}) \mapsto \text{srv}) \mapsto \emptyset \in \text{s\_i}) \wedge ((\text{E}(\text{srv}) \mapsto \text{srv}) \mapsto \emptyset \in \text{rt\_s}) \wedge ((\text{E}(\text{srv}) \mapsto \text{srv}) \mapsto \emptyset \in \text{rc\_s})$  not theorem >

THEN

act1:  $\text{unav\_peers} := \text{unav\_peers} \cup \text{prs}$  >the peers in prs become unavailable

act2:  $\text{token\_owner} := \text{token\_owner} \leftarrow \text{E}$  >new value for token owner per service is given if needed

act3:  $\text{rect\_inst} := \text{rc\_s}$  >the peers in prs can not try to recontact instances anymore

act4:  $\text{rctt\_inst} := \text{rt\_s}$  >the peers in prs can not recontact instances anymore

act5:  $\text{actv\_inst} := \text{prs} \leftarrow \text{actv\_inst}$  >

act6:  $\text{i\_state} := \text{i\_s}$  >

act7:  $\text{suspc\_peers} := \text{p\_s}$  >

act8:  $\text{suspc\_inst} := \text{s\_i}$  >

END

SUSPECT\_INST: extended ordinary >

REFINES

SUSPECT\_INST

ANY

$s$  >a service  $s$

$\text{susp}$  >suspicious instances

WHERE

grd1:  $s \in \text{SERVICES}$  not theorem >

grd2:  $\text{susp} \subseteq \text{PEERS}$  not theorem >

grd3:  $\text{susp} = \text{run\_peers}(s) \cap \text{unav\_peers}$  not theorem >instances in  $\text{susp}$  are suspicious if the peers running them becomes unavailable

grd4:  $\text{suspc\_inst}(\text{token\_owner}(s) \mapsto s) = \emptyset$  not theorem >the member of  $\text{susp}$  have not yet been suspected for  $s$  by the token owner of  $s$

grd5:  $\text{i\_state}(\text{token\_owner}(s) \mapsto s) = \text{RUN\_4}$  not theorem >the state of  $s$  is OK

grd6:  $\text{susp} \neq \emptyset$  not theorem >

THEN

act1:  $\text{suspc\_inst}(\text{token\_owner}(s) \mapsto s) := \text{susp}$  >the members of  $\text{susp}$  become suspected instances for  $s$  by the token owner of  $s$

END

FAIL: extended ordinary >

REFINES

FAIL

ANY

$s$  >

WHERE

grd1:  $s \in \text{SERVICES}$  not theorem >



M14

```

    grd2:  i_state(token_owner(s) ↦ s) = RUN_4 not theorem >
    grd3:  suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_4 >
    act2:  suspc_peers(token_owner(s) ↦ s) := suspc_inst(token_owner
(s) ↦ s) >
    act3:  suspc_inst(token_owner(s) ↦ s) := ∅ >
  END

RECONTACT_INST_OK:  not extended ordinary >
  REFINES
    RECONTACT_INST_OK
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
    grd5:  i ∈ suspc_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspc_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
  THEN
    act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
  END

RECONTACT_INST_K0:  not extended ordinary >
  REFINES
    RECONTACT_INST_K0
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set

```

M14

```

of suspicious peers for s is not empty
  grd5: i ∈ suspc_peers(token_owner(s) ↦ s) ∧ i ∈ unav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
  grd6: i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
  grd7: rect_inst(token_owner(s) ↦ s) ⊂ suspc_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
  THEN
    act1: rect_inst(token_owner(s) ↦ s) = rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
  END

FAIL_DETECT: not extended ordinary >
  REFINES
    FAIL_DETECT
  ANY
    s >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
    grd5: suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    (s) ↦ s) not theorem >
    grd8: rect_inst(token_owner(s) ↦ s) = suspc_peers(token_owner
    THEN
      act1: i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 >
      act2: suspc_peers(token_owner(s) ↦ s) = suspc_peers
      (token_owner(s) ↦ s) \ rctt_inst(token_owner(s) ↦ s) >
      act3: rect_inst(token_owner(s) ↦ s) = ∅ >
      act4: rctt_inst(token_owner(s) ↦ s) = ∅ >
    END

IS_OK: extended ordinary >
  REFINES
    IS_OK
  ANY
    s >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
    >
    grd5: suspc_peers(token_owner(s) ↦ s) = ∅ not theorem >
  THEN
    act1: i_state(token_owner(s) ↦ s) = RUN_4 >
  END

FAIL_ACTIV: not extended ordinary >

```

```

REFINES
  FAIL_ACTIV
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
>
  grd5: suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
THEN
  act1: i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 >
  act2: run_peers(s) = run_peers(s) \ suspc_peers(token_owner(s)
↦ s) >
  act3: failr_peers(s) = failr_peers(s) ∪ suspc_peers
(token_owner(s) ↦ s) >
  act4: suspc_peers(token_owner(s) ↦ s) = ∅ >
END

FAIL_CONFIGURE: extended ordinary >
REFINES
  FAIL_CONFIGURE
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
  grd3: card(run_peers(s)) < min_inst(s) not theorem >
THEN
  act1: i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 >
END

FAIL_IGNORE: extended ordinary >
REFINES
  FAIL_IGNORE
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
  grd3: card(run_peers(s)) ≥ min_inst(s) not theorem >
THEN
  act1: i_state(token_owner(s) ↦ s) = FAIL_IGN_4 >
END

IGNORE: extended ordinary >
REFINES
  IGNORE
ANY

```

```

s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: i_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem >
THEN
  act1: i_state(token_owner(s) ↦ s) := RUN_4 >
END

REDEPLOY_INSTC: not extended ordinary >
REFINES
  REDEPLOY_INSTC
ANY
  s >a service s
  i >an instance i
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd2: i ∈ PEERS not theorem >
  grd3: i ∉ run_peers(s) ∪ failr_peers(s) ∪ unav_peers ∪
dep_instc(s) not theorem >i does not run s, is not failed for s, is not
unavailable and is not already activated for s
  grd4: token_owner(s) ↦ (s ↦ i) ∉ actv_inst not theorem >
  grd5: i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
  grd6: card(actv_inst[{{token_owner(s)}}][{s}]) < deplo_inst(s)
not theorem >
  grd7: card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
not theorem >
THEN
  act1: actv_inst := actv_inst ∪ {token_owner(s) ↦ (s ↦ i)} >
END

REDEPLOY_INSTS: not extended ordinary >
REFINES
  REDEPLOY_INSTS
ANY
  s >
WHERE
  grd1: s ∈ SERVICES not theorem >
  grd6: card(actv_inst[{{token_owner(s)}}][{s}]) = deplo_inst(s)
not theorem >
  grd7: card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
not theorem >
  grd8: i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
THEN
  act1: dep_instc(s) := dep_instc(s) ∪ actv_inst[{{token_owner
(s)}}][{s}] >
  act2: actv_inst := actv_inst ▷ ({s} ◁ ran(actv_inst)) >

```

```

END
REDEPLOY:  not extended ordinary >
  REFINES
    REDEPLOY
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    grd7:  actv_inst[{token_owner(s)}][{s}] = ∅ not theorem >
    grd6:  dep_instc(s) ≠ ∅ not theorem >
    grd4:  card(run_peers(s)) + card(dep_instc(s)) ≥ min_inst(s) not
theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) = DPL_4 >
    act2:  run_peers(s) = run_peers(s) ∪ dep_instc(s) >
    act3:  dep_instc(s) = ∅ >
  END

HEAL:  extended ordinary >
  REFINES
    HEAL
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = DPL_4 not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) = RUN_4 >
  END

UNFAIL_PEER:  not extended ordinary >
  REFINES
    UNFAIL_PEER
  ANY
    s >
    p >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  p ∈ PEERS not theorem >
    grd3:  p ∈ failr_peers(s) not theorem >
  THEN
    act1:  failr_peers(s) = failr_peers(s) \ {p} >
  END

MAKE_PEER_AVAIL:  extended ordinary >

```

M14

```
REFINES
  MAKE_PEER_AVAIL
ANY
  p >
WHERE
  grd1: p ∈ PEERS not theorem >
  grd2: p ∈ unav_peers not theorem >
THEN
  act1: unav_peers := unav_peers \ {p} >
END
```

END

M15

```

MACHINE
  M15    >
REFINES
  M14
SEES
  C08
VARIABLES
  run_peers    >
  suspc_peers  >
  failr_peers  >
  dep_instc    >
  token_owner  >
  unav_peers   >
  suspc_inst   >
  rect_inst    >instances that are tried to be recontacted
  rctt_inst    >instances effectively recontacted after a try
  actv_instc   >instances activated by token owne
  i_state     >
INVARIANTS
  inv1:  actv_instc ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈ dom(actv_instc) ⇒
p = token_owner(s) not theorem >
  inv3:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ p = token_owner(s) ⇒ (p ↦
s) ∈ dom(actv_instc) not theorem >
  gluing_tok_own_recl:  ∀ p, s · p ∈ PEERS ∧ s ∈ SERVICES ∧ (p ↦ s) ∈
dom(actv_instc) ⇒ actv_inst[{p}][{s}] = actv_instc(p ↦ s) not theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
  THEN
    act2:  run_peers := InitSrvcPeers >
    act3:  suspc_peers := InitSuspPeers >
    act4:  failr_peers := InitFail >
    act5:  dep_instc := InitFail >
    act6:  token_owner := init_tok >
    act7:  unav_peers := ∅ >
    act8:  suspc_inst := InitSuspPeers >
    act10: rect_inst := InitSuspPeers >
    act11: rctt_inst := InitSuspPeers >
    act12: actv_instc := InitSuspPeers >
    act13: i_state := InitStatus >
  END

MAKE_PEER_UNAVAIL:  not extended ordinary >
REFINES
  MAKE_PEER_UNAVAIL
ANY
  prs    >
  E      >new values for token owner per service if needed

```

```

i_s  >
p_s  >
s_i  >
rc_s  >
rt_s  >
ac_i  >
WHERE
grd1:  prs ⊆ PEERS not theorem >
grd2:  prs ⊄ unav_peers not theorem >
grd3:  E ∈ SERVICES → PEERS not theorem >new value for token
owner per service if needed
grd4:  i_s ∈ (PEERS×SERVICES) ↔ STATES_4 not theorem >
grd5:  p_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd6:  s_i ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd7:  rt_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd8:  rc_s ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd9:  ac_i ∈ (PEERS×SERVICES) ↔ P(PEERS) not theorem >
grd10: dom(i_s) = E~ ∧ dom(p_s) = dom(i_s) ∧ dom(s_i) = dom
(i_s) ∧ dom(rc_s) = dom(i_s) ∧ dom(rt_s) = dom(i_s) ∧ dom(ac_i) = dom(i_s) not
theorem >
grd11: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs
⇒
E(srv) = token_owner(srv) ∧
s_i(E(srv) ↦ srv) = suspc_inst(E(srv) ↦ srv) ∧
rt_s(E(srv) ↦ srv) = rctt_inst(E(srv) ↦ srv) ∧
rc_s(E(srv) ↦ srv) = rect_inst(E(srv) ↦ srv) ∧
ac_i(E(srv) ↦ srv) = actv_instc(E(srv) ↦ srv) not
theorem >If the token owner of a service srv does not belong to prs, the token
owner is not changed
grd12: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs
⇒
E(srv) ∈ run_peers(srv)\(unav_peers ∪ prs ∪ failr_peers
(srv) ∪ suspc_peers(token_owner(srv) ↦ srv)) ∧
s_i(E(srv) ↦ srv) = ∅ ∧
rt_s(E(srv) ↦ srv) = ∅ ∧
rc_s(E(srv) ↦ srv) = ∅ ∧
ac_i(E(srv) ↦ srv) = ∅ not theorem >if the owner of the
token for a service becomes unavailable, and the service
possess suspicious
instances, then a new token owner among available and not
suspicious peers is
chosen
grd13: ∀ srv · srv ∈ SERVICES ⇒ i_s(E(srv) ↦ srv) = i_state
(token_owner(srv) ↦ srv) ∧ p_s(E(srv) ↦ srv) = suspc_peers(token_owner(srv) ↦
srv) not theorem >
THEN
act1:  unav_peers = unav_peers ∪ prs >the peers in prs become
unavailable

```



M15

```

    act2: token_owner := token_owner ◀ E ›new value for token owner
per service is given if needed
    act3: rect_inst := rc_s ›the peers in prs can not try to
recontact instances anymore
    act4: rctt_inst := rt_s ›the peers in prs can not recontact
instances anymore
    act5: actv_instc := ac_i ›
    act6: i_state := i_s ›
    act7: suspc_peers := p_s ›
    act8: suspc_inst := s_i ›
END

```

```

SUSPECT_INST: extended ordinary ›
REFINES
    SUSPECT_INST
    ANY
    s ›a service s
    susp ›suspicious instances
    WHERE
    grd1: s ∈ SERVICES not theorem ›
    grd2: susp ⊆ PEERS not theorem ›
    grd3: susp = run_peers(s) n unav_peers not theorem ›instances
in susp are suspicious if the peers running them becomes unavailable
    grd4: suspc_inst(token_owner(s) ↦ s) = ∅ not theorem ›the
member of susp have not yet been suspected for s by the token owner of s
    grd5: i_state(token_owner(s) ↦ s) = RUN_4 not theorem ›the
state of s is OK
    grd6: susp ≠ ∅ not theorem ›
    THEN
    act1: suspc_inst(token_owner(s) ↦ s) := susp ›the members of
susp become suspected instances for s by the token owner of s
    END

```

```

FAIL: extended ordinary ›
REFINES
    FAIL
    ANY
    s ›
    WHERE
    grd1: s ∈ SERVICES not theorem ›
    grd2: i_state(token_owner(s) ↦ s) = RUN_4 not theorem ›
    grd3: suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem ›
    THEN
    act1: i_state(token_owner(s) ↦ s) := FAIL_4 ›
    act2: suspc_peers(token_owner(s) ↦ s) := suspc_inst(token_owner
(s) ↦ s) ›
    act3: suspc_inst(token_owner(s) ↦ s) := ∅ ›
    END

```

```

RECONTACT_INST_OK:    extended ordinary >
  REFINES
    RECONTACT_INST_OK
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
    grd5:  i ∈ suspc_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspc_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
  THEN
    act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
  END

RECONTACT_INST_K0:    extended ordinary >
  REFINES
    RECONTACT_INST_K0
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
    grd5:  i ∈ suspc_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspc_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all

```

the suspicious instances of s

```

    THEN
      act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    END

  FAIL_DETECT:  extended ordinary >
    REFINES
      FAIL_DETECT
    ANY
      s >
    WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  i_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
      grd5:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
      grd8:  rect_inst(token_owner(s) ↦ s) = suspc_peers(token_owner
(s) ↦ s) not theorem >
    THEN
      act1:  i_state(token_owner(s) ↦ s) := FAIL_DETECT_4 >
      act2:  suspc_peers(token_owner(s) ↦ s) := suspc_peers
(token_owner(s) ↦ s) \ rctt_inst(token_owner(s) ↦ s) >
      act3:  rect_inst(token_owner(s) ↦ s) := ∅ >
      act4:  rctt_inst(token_owner(s) ↦ s) := ∅ >
    END

  IS_OK:  extended ordinary >
    REFINES
      IS_OK
    ANY
      s >
    WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
>
      grd5:  suspc_peers(token_owner(s) ↦ s) = ∅ not theorem >
    THEN
      act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
    END

  FAIL_ACTIV:  extended ordinary >
    REFINES
      FAIL_ACTIV
    ANY
      s >
    WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  i_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not theorem
>

```

M15

```

    grd5:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_ACTIV_4 >
    act2:  run_peers(s) := run_peers(s) \ suspc_peers(token_owner(s)
↦ s) >
    act3:  failr_peers(s) := failr_peers(s) ∪ suspc_peers
(token_owner(s) ↦ s) >
    act4:  suspc_peers(token_owner(s) ↦ s) := ∅ >
  END

FAIL_CONFIGURE:  extended ordinary >
  REFINES
    FAIL_CONFIGURE
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) < min_inst(s) not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_CONFIG_4 >
  END

FAIL_IGNORE:  extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not theorem >
    grd3:  card(run_peers(s)) ≥ min_inst(s) not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := FAIL_IGN_4 >
  END

IGNORE:  extended ordinary >
  REFINES
    IGNORE
  ANY
    s >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem >
  THEN
    act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
  END

```

```

REDEPLOY_INSTC:    not extended ordinary >
  REFINES
    REDEPLOY_INSTC
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i ∉ run_peers(s) ∪ failr_peers(s) ∪ unav_peers ∪
dep_instc(s) not theorem >i does not run s, is not failed for s, is not
unavailable and is not already activated for s
    grd4:  i ∉ actv_instc(token_owner(s) ↦ s) not theorem >
    grd5:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
    grd6:  card(actv_instc(token_owner(s) ↦ s)) < deplo_inst(s) not
theorem >
    grd7:  card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
not theorem >
  THEN
    act1:  actv_instc(token_owner(s) ↦ s) = actv_instc(token_owner
(s) ↦ s) ∪ {i} >
  END

REDEPLOY_INSTS:    not extended ordinary >
  REFINES
    REDEPLOY_INSTS
  ANY
    s    >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd6:  card(actv_instc(token_owner(s) ↦ s)) = deplo_inst(s) not
theorem >
    grd7:  card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd8:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
  THEN
    act1:  dep_instc(s) = dep_instc(s) ∪ actv_instc(token_owner(s)
↦ s) >
    act2:  actv_instc(token_owner(s) ↦ s) = ∅ >
  END

REDEPLOY:    not extended ordinary >
  REFINES
    REDEPLOY
  ANY
    s    >

```

M15

```

WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not theorem
>
  grd7:  actv_instc(token_owner(s) ↦ s)=∅ not theorem >
  grd6:  dep_instc(s) ≠ ∅ not theorem >
  grd4:  card(run_peers(s))+card(dep_instc(s)) ≥ min_inst(s) not
theorem >
THEN
  act1:  i_state(token_owner(s) ↦ s) := DPL_4 >
  act2:  run_peers(s) := run_peers(s) ∪ dep_instc(s) >
  act3:  dep_instc(s) := ∅ >
END

HEAL:  extended ordinary >
REFINES
  HEAL
ANY
  s >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i_state(token_owner(s) ↦ s) = DPL_4 not theorem >
THEN
  act1:  i_state(token_owner(s) ↦ s) := RUN_4 >
END

UNFAIL_PEER:  extended ordinary >
REFINES
  UNFAIL_PEER
ANY
  s >
  p >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  p ∈ PEERS not theorem >
  grd3:  p ∈ failr_peers(s) not theorem >
THEN
  act1:  failr_peers(s) := failr_peers(s)\{p} >
END

MAKE_PEER_AVAIL:  extended ordinary >
REFINES
  MAKE_PEER_AVAIL
ANY
  p >
WHERE
  grd1:  p ∈ PEERS not theorem >
  grd2:  p ∈ unav_peers not theorem >

```

M15

```
THEN  
  act1: unav_peers := unav_peers \ {p} >  
END
```

END

```

MACHINE
  M16    >
REFINES
  M15
SEES
  C09
VARIABLES
  run_peers    >
  suspc_peers  >
  failr_peers  >
  dep_instc    >
  token_owner  >
  unav_peers   >
  suspc_inst   >
  rect_inst    >instances that are tried to be recontacted
  rctt_inst    >instances effectively recontacted after a try
  actv_instc   >instances activated by token owne
  inst_state   >
INVARIANTS
  inv1:  inst_state ∈ (PEERS×SERVICES) ↔ STATES_4 not theorem >
  inv2:  ∀ s · s ∈ SERVICES ⇒ token_owner(s) ↦ s ∈ dom(inst_state) not
theorem >
  gluing_state_1: ∀ s · s ∈ SERVICES ⇒ i_state(token_owner(s) ↦ s) =
inst_state(token_owner(s) ↦ s) not theorem >
  inv3:  ∀ s · s ∈ SERVICES ⇒ rctt_inst(token_owner(s) ↦ s) ⊆ run_peers
(s) not theorem >
  inv4:  ∀ s · s ∈ SERVICES ⇒ suspc_peers(token_owner(s) ↦ s) ⊆
run_peers(s) not theorem >
  inv5:  ∀ s · s ∈ SERVICES ⇒ suspc_inst(token_owner(s) ↦ s) ⊆ run_peers
(s) not theorem >
  inv6:  ∀ s · s ∈ SERVICES ⇒ rect_inst(token_owner(s) ↦ s) ⊆ run_peers
(s) not theorem >
  inv7:  ∀ s · s ∈ SERVICES ⇒ token_owner(s) ∉ suspc_inst(token_owner(s)
↦ s) not theorem >
  inv8:  ∀ s · s ∈ SERVICES ⇒ token_owner(s) ∉ suspc_peers(token_owner
(s) ↦ s) not theorem >
  inv9:  ∀ s · s ∈ SERVICES ⇒ token_owner(s) ∉ rctt_inst(token_owner(s)
↦ s) not theorem >
  inv10: ∀ s · s ∈ SERVICES ⇒ token_owner(s) ∉ rect_inst(token_owner(s)
↦ s) not theorem >
  inv11: ∀ s · s ∈ SERVICES ⇒ suspc_inst(token_owner(s) ↦ s) ∩
suspc_peers(token_owner(s) ↦ s) = ∅ not theorem >
  inv12: ∀ s · s ∈ SERVICES ∧ inst_state(token_owner(s) ↦ s) ∉
{FAIL_4, FAIL_DETECT_4} ⇒ suspc_peers(token_owner(s) ↦ s) = ∅ not theorem >
  inv13: ∀ s · s ∈ SERVICES ∧ inst_state(token_owner(s) ↦ s) ≠ FAIL_4 ⇒
rctt_inst(token_owner(s) ↦ s) = ∅ not theorem >
  inv14: ∀ s · s ∈ SERVICES ∧ inst_state(token_owner(s) ↦ s) ≠ FAIL_4 ⇒
rect_inst(token_owner(s) ↦ s) = ∅ not theorem >

```



## EVENTS

```

INITIALISATION:    not extended ordinary >
  THEN
    act1:  run_peers := InitSrvcPeers >
    act2:  suspc_peers := InitSuspPeers >
    act3:  failr_peers := InitFail >
    act4:  dep_instc := InitFail >
    act5:  token_owner := init_tok >
    act6:  unav_peers := ∅ >
    act7:  suspc_inst := InitSuspPeers >
    act8:  rect_inst := InitSuspPeers >
    act9:  rctt_inst := InitSuspPeers >
    act10: actv_instc := InitSuspPeers >
    act11: inst_state := InitStateSrv >
  END

MAKE_PEER_UNAVAIL:  not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY
    prs >
    E >new values for token owner per service if needed
    p_s >
    s_i >
    rc_s >
    rt_s >
    ac_i >
  WHERE
    grd1:  prs ⊆ PEERS not theorem >
    grd2:  prs ⊄ unav_peers not theorem >
    grd3:  ∀ srv · srv ∈ SERVICES ⇒ dom(dom(inst_state) ▷ {srv})
\prs ≠ ∅ not theorem >
    grd4:  E ∈ SERVICES → PEERS not theorem >new value for token
owner per service if needed
    grd5:  p_s ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
    grd6:  s_i ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
    grd7:  rt_s ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
    grd8:  rc_s ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
    grd9:  ac_i ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
    grd10: dom(p_s) = E~ ∧ dom(s_i) = E~ ∧ dom(rc_s) = E~ ∧ dom
(rt_s) = E~ ∧ dom(ac_i) = E~ not theorem >
    grd11: ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∉ prs
⇒
E(srv) = token_owner(srv) ∧
s_i(E(srv) ↦ srv) = suspc_inst(E(srv) ↦ srv) ∧
rt_s(E(srv) ↦ srv) = rctt_inst(E(srv) ↦ srv) ∧
rc_s(E(srv) ↦ srv) = rect_inst(E(srv) ↦ srv) ∧
ac_i(E(srv) ↦ srv) = actv_instc(E(srv) ↦ srv) not

```

M16

theorem >If the token owner of a service  $srv$  does not belong to  $prs$ , the token owner is not changed

```

grd12:  ∀ srv · srv ∈ SERVICES ∧ token_owner(srv) ∈ prs
        ⇒
        E(srv) ∈ run_peers(srv) \ (unav_peers ∪ prs ∪ failr_peers
(srv) ∪ suspc_peers(token_owner(srv) ↦ srv)) ∧
        E(srv) ↦ srv ∈ dom(inst_state) ∧
        inst_state(E(srv) ↦ srv) = inst_state(token_owner(srv) ↦
srv) ∧
        s_i(E(srv) ↦ srv) = ∅ ∧
        rt_s(E(srv) ↦ srv) = ∅ ∧
        rc_s(E(srv) ↦ srv) = ∅ ∧
        ac_i(E(srv) ↦ srv) = ∅

```

not theorem >if the owner of the token for a service becomes unavailable, and the service possess suspicious instances, then a new token owner among available and not suspicious peers is chosen

```

grd13:  ∀ srv · srv ∈ SERVICES ⇒ p_s(E(srv) ↦ srv) =
suspc_peers(token_owner(srv) ↦ srv)
WITH
  i_s:   i_s = E~ < inst_state >
THEN
  act1:  unav_peers := unav_peers ∪ prs >the peers in prs become
unavailable
  act2:  token_owner := token_owner ◀ E >new value for token owner
per service is given if needed
  act3:  rect_inst := rc_s >the peers in prs can not try to
recontact instances anymore
  act4:  rctt_inst := rt_s >the peers in prs can not recontact
instances anymore
  act5:  actv_instc := ac_i >
  act6:  suspc_peers := p_s >
  act7:  suspc_inst := s_i >
  act8:  inst_state := (prs×SERVICES) ◀ inst_state >
END

```

SUSPECT\_INST: not extended ordinary >

REFINES

SUSPECT\_INST

ANY

s >a service s

susp >suspicious instances

WHERE

grd1: s ∈ SERVICES not theorem >

grd2: susp ⊆ PEERS not theorem >

grd3: susp = run\_peers(s) n unav\_peers not theorem >instances

in susp are suspicious if the peers running them becomes unavailable

M16

```

    grd4:  suspc_inst(token_owner(s) ↦ s) = ∅ not theorem >the
member of susp have not yet been suspected for s by the token owner of s
    grd5:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
    grd6:  susp ≠ ∅ not theorem >
    THEN
    act1:  suspc_inst(token_owner(s) ↦ s) := susp >the members of
susp become suspected instances for s by the token owner of s
    END

FAIL:    not extended ordinary >
    REFINES
    FAIL
    ANY
    s      >
    prop   >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >
    grd4:  suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5:  prop = run_peers(s) \ (suspc_inst(token_owner(s) ↦ s) ∪
unav_peers) not theorem >
    THEN
    act1:  inst_state := inst_state ◀ ((prop × {s}) × {FAIL_4}) >
    act2:  suspc_peers(token_owner(s) ↦ s) := suspc_inst(token_owner
(s) ↦ s) >
    act3:  suspc_inst(token_owner(s) ↦ s) := ∅ >
    END

RECONTACT_INST_OK:    not extended ordinary >
    REFINES
    RECONTACT_INST_OK
    ANY
    s      >a service s
    i      >an instance i
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
    grd5:  i ∈ suspc_peers(token_owner(s) ↦ s) \ unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspc_peers(token_owner

```

```

(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
    THEN
        act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
        act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
    END

RECONTACT_INST_K0:  not extended ordinary >
    REFINES
        RECONTACT_INST_K0
    ANY
        s      >a service s
        i      >an instance i
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  i ∈ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
        grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the set
of suspicious peers for s is not empty
        grd5:  i ∈ suspc_peers(token_owner(s) ↦ s) ∧ !nunav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
        grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
        grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspc_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
    THEN
        act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    END

FAIL_DETECT:  not extended ordinary >
    REFINES
        FAIL_DETECT
    ANY
        s      >
        prop    >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
        grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
        grd5:  rect_inst(token_owner(s) ↦ s) = suspc_peers(token_owner
(s) ↦ s) not theorem >

```

M16

```

    grd6:  prop = ((run_peers(s) \ suspc_peers(token_owner(s) ↦ s))
u rctt_inst(token_owner(s)↦ s))\unav_peers not theorem >
    THEN
    act1:  inst_state = inst_state < ((prop×{s})×{FAIL_DETECT_4})
>
    act2:  suspc_peers(token_owner(s) ↦ s) = suspc_peers
(token_owner(s) ↦ s) \ rctt_inst(token_owner(s) ↦ s) >
    act3:  rctt_inst(token_owner(s) ↦ s) = ∅ >
    act4:  rctt_inst(token_owner(s) ↦ s) = ∅ >
    END

IS_OK:  not extended ordinary >
    REFINES
    IS_OK
    ANY
    s >
    prop >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
    grd4:  suspc_peers(token_owner(s) ↦ s) = ∅ not theorem >
    grd5:  prop = run_peers(s)\unav_peers not theorem >
    THEN
    act1:  inst_state = inst_state < ((prop×{s})×{RUN_4}) >
    END

FAIL_ACTIV:  not extended ordinary >
    REFINES
    FAIL_ACTIV
    ANY
    s >
    prop >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
    grd4:  suspc_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5:  prop = run_peers(s) \ (unav_peers ∪ suspc_peers
(token_owner(s) ↦ s)) not theorem >
    THEN
    act1:  inst_state = inst_state < ((prop×{s})×{FAIL_ACTIV_4}) >
    act2:  run_peers(s) = run_peers(s) \ suspc_peers(token_owner(s)
↦ s) >
    act3:  failr_peers(s) = failr_peers(s) ∪ suspc_peers
(token_owner(s) ↦ s) >

```

```

act4:  suspc_peers(token_owner(s) ↦ s) := ∅ >
END

FAIL_CONFIGURE:  not extended ordinary >
REFINES
  FAIL_CONFIGURE
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
  grd4:  card(run_peers(s)) < min_inst(s) not theorem >
  grd5:  prop = run_peers(s)\unav_peers not theorem >
THEN
  act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_CONFIG_4}) >
END

FAIL_IGNORE:  not extended ordinary >
REFINES
  FAIL_IGNORE
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
  grd4:  card(run_peers(s)) ≥ min_inst(s) not theorem >
  grd5:  prop = run_peers(s)\unav_peers not theorem >
THEN
  act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_IGN_4}) >
END

IGNORE: not extended ordinary >
REFINES
  IGNORE
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem
>

```

```

    grd4:   prop = run_peers(s)\unav_peers not theorem >
  THEN
    act1:   inst_state := inst_state << ((prop×{s})×{RUN_4}) >
  END

REDEPLOY_INSTC:   not extended ordinary >
  REFINES
    REDEPLOY_INSTC
  ANY
    s      >a service s
    i      >an instance i
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   i ∈ PEERS not theorem >
    grd3:   i ∉ run_peers(s) ∪ failr_peers(s) ∪ unav_peers ∪
dep_instc(s) not theorem >i does not run s, is not failed for s, is not
unavailable and is not already activated for s
    grd4:   i ∉ actv_instc(token_owner(s) ↦ s) not theorem >
    grd5:   inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
    grd6:   card(actv_instc(token_owner(s) ↦ s)) < deplo_inst(s) not
theorem >
    grd7:   card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
not theorem >
  THEN
    act1:   actv_instc(token_owner(s) ↦ s) := actv_instc(token_owner
(s) ↦ s) ∪ {i} >
  END

REDEPLOY_INSTS:   not extended ordinary >
  REFINES
    REDEPLOY_INSTS
  ANY
    s      >
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   card(actv_instc(token_owner(s) ↦ s)) = deplo_inst(s) not
theorem >
    grd3:   card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
not theorem >
    grd4:   inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
  THEN
    act1:   dep_instc(s) := dep_instc(s) ∪ actv_instc(token_owner(s)
↦ s) >
    act2:   actv_instc(token_owner(s) ↦ s) := ∅ >
  END

```

M16

```

REDEPLOY:  not extended ordinary >
  REFINES
    REDEPLOY
  ANY
    s      >
    prop   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
    grd4:  actv_instc(token_owner(s) ↦ s)=∅ not theorem >
    grd5:  dep_instc(s) ≠ ∅ not theorem >
    grd6:  card(run_peers(s))+card(dep_instc(s)) ≥ min_inst(s) not
theorem >
    grd7:  prop = run_peers(s)\unav_peers not theorem >
  THEN
    act1:  inst_state= inst_state ◀ ((prop×{s})×{DPL_4}) >
    act2:  run_peers(s) = run_peers(s) ∪ dep_instc(s) >
    act3:  dep_instc(s) = ∅ >
  END

HEAL:  not extended ordinary >
  REFINES
    HEAL
  ANY
    s      >
    prop   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = DPL_4 not theorem >
    grd4:  prop = run_peers(s)\unav_peers not theorem >
  THEN
    act1:  inst_state= inst_state ◀ ((prop×{s})×{RUN_4}) >
  END

UNFAIL_PEER:  extended ordinary >
  REFINES
    UNFAIL_PEER
  ANY
    s      >
    p      >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  p ∈ PEERS not theorem >
    grd3:  p ∈ failr_peers(s) not theorem >
  THEN

```



M16

```
act1: failr_peers(s) := failr_peers(s)\{p} >
END

MAKE_PEER_AVAIL: extended ordinary >
REFINES
  MAKE_PEER_AVAIL
ANY
  p >
WHERE
  grd1: p ∈ PEERS not theorem >
  grd2: p ∈ unav_peers not theorem >
THEN
  act1: unav_peers := unav_peers \ {p} >
END

END
```

## M17

```

MACHINE
  M17 >
REFINES
  M16
SEES
  C09
VARIABLES
  run_peers >
  suspct_peers >
  failr_peers >
  dep_instc >
  token_owner >
  unav_peers >
  suspc_inst >
  rect_inst >instances that are tried to be recontacted
  rctt_inst >instances effectively recontacted after a try
  actv_instc >instances activated by token owne
  inst_state >
INVARIANTS
  inv1: suspct_peers ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2: ∀ s · s ∈ SERVICES ⇒ token_owner(s) ↦ s ∈ dom(suspct_peers) not
theorem >
  gluing_susp_1: ∀ s · s ∈ SERVICES ⇒ suspc_peers(token_owner(s) ↦ s) =
suspct_peers(token_owner(s) ↦ s) not theorem >
EVENTS
  INITIALISATION: not extended ordinary >
  THEN
    act1: run_peers := InitSrvcPeers >
    act2: suspct_peers := InitSuspPrs >
    act3: failr_peers := InitFail >
    act4: dep_instc := InitFail >
    act5: token_owner := init_tok >
    act6: unav_peers := ∅ >
    act7: suspc_inst := InitSuspPeers >
    act8: rect_inst := InitSuspPeers >
    act9: rctt_inst := InitSuspPeers >
    act10: actv_instc := InitSuspPeers >
    act11: inst_state := InitStateSrv >
  END
  MAKE_PEER_UNAVAIL: not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY
    prs >Peers that will become unavailable
    E >Values for token owner per service
  WHERE
    grd1: prs ⊆ PEERS not theorem >

```

M17

yet unavaible  
 grd2:  $\text{prs} \not\subseteq \text{unav\_peers}$  not theorem >the peers in prs are not  
 \prs  $\neq \emptyset$  not theorem >for each service srv, there must always be at least 1  
 peer available  
 per service  
 grd4:  $E \in \text{SERVICES} \rightarrow \text{PEERS}$  not theorem >Value for token owner  
 owner of a service srv  
 grd5:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \notin \text{prs}$   
 $\Rightarrow$   
 $E(\text{srv}) = \text{token\_owner}(\text{srv})$  not theorem >If the token  
 does not belong to prs, the token owner is not changed  
 grd6:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs}$   
 $\Rightarrow$   
 $E(\text{srv}) \in \text{run\_peers}(\text{srv}) \setminus (\text{unav\_peers} \cup \text{prs} \cup \text{failr\_peers}$   
 $(\text{srv}) \cup \text{suspc\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv})) \wedge$   
 $E(\text{srv}) \mapsto \text{srv} \in \text{dom}(\text{inst\_state}) \wedge E(\text{srv}) \mapsto \text{srv} \in \text{dom}$   
 $(\text{suspc\_peers}) \wedge$   
 $\text{inst\_state}(E(\text{srv}) \mapsto \text{srv}) = \text{inst\_state}(\text{token\_owner}(\text{srv}) \mapsto$   
 $\text{srv}) \wedge$   
 $\text{suspc\_peers}(E(\text{srv}) \mapsto \text{srv}) = \text{suspc\_peers}(\text{token\_owner}$   
 $(\text{srv}) \mapsto \text{srv})$  not theorem >if the owner of the token for a service becomes  
 unavailable,

A new token owner is chosen: the new token owner must have same characteristics  
 as the previous one (state, list of suspicious neighbours, etc.), and it must  
 not be an unavailable, suspicious, failed peer or a member of prs

WITH  
 p\_s:  $\text{p\_s} = E \sim \triangleleft \text{suspc\_peers} \triangleright$   
 rc\_s:  $\text{rc\_s} = ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rect\_inst}) \triangleleft (((E \setminus \text{token\_owner})$   
 $\sim) \times \{\emptyset\}) \triangleright$   
 s\_i:  $\text{s\_i} = ((\text{prs} \times \text{SERVICES}) \triangleleft \text{suspc\_inst}) \triangleleft (((E \setminus \text{token\_owner})$   
 $\sim) \times \{\emptyset\}) \triangleright$   
 rt\_s:  $\text{rt\_s} = ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rctt\_inst}) \triangleleft (((E \setminus \text{token\_owner})$   
 $\sim) \times \{\emptyset\}) \triangleright$   
 ac\_i:  $\text{ac\_i} = ((\text{prs} \times \text{SERVICES}) \triangleleft \text{actv\_instc}) \triangleleft (((E \setminus \text{token\_owner})$   
 $\sim) \times \{\emptyset\}) \triangleright$   
 THEN  
 unavailable  
 act1:  $\text{unav\_peers} := \text{unav\_peers} \cup \text{prs}$  >the peers in prs become  
 owner per service  
 act2:  $\text{token\_owner} := \text{token\_owner} \triangleleft E$  >new values for token  
 act3:  $\text{rect\_inst} = ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rect\_inst}) \triangleleft$   
 $((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\})$  >the peers in prs can not try to recontact instances

```

anymore (1)
    act4:  rctt_inst := ((prs×SERVICES) ◀ rctt_inst) ◀
(((E\token_owner)~)×{∅}) >the peers in prs can not try to recontact instances
anymore (2)
    act5:  actv_instc := ((prs×SERVICES) ◀ actv_instc) ◀
(((E\token_owner)~)×{∅}) >the peers in prs can not activate instances anymore
    act6:  suspct_peers := (prs×SERVICES) ◀ suspct_peers >the peers
in prs can not suspect instances anymore (1)
    act7:  suspc_inst := ((prs×SERVICES) ◀ suspc_inst) ◀
(((E\token_owner)~)×{∅}) >the peers in prs can not suspect instances anymore (2)
    act8:  inst_state := (prs×SERVICES) ◀ inst_state >the peers in
prs can not monitor the state of the services provided anymore
    END

SUSPECT_INST:  extended ordinary >
    REFINES
        SUSPECT_INST
    ANY
        s      >a service s
        susp   >suspicious instances
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  susp ⊆ PEERS not theorem >
        grd3:  susp = run_peers(s) n unav_peers not theorem >instances
in susp are suspicious if the peers running them becomes unavailable
        grd4:  suspc_inst(token_owner(s) ↦ s) = ∅ not theorem >the
member of susp have not yet been suspected for s by the token owner of s
        grd5:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
        grd6:  susp ≠ ∅ not theorem >
    THEN
        act1:  suspc_inst(token_owner(s) ↦ s) := susp >the members of
susp become suspected instances for s by the token owner of s
    END

FAIL:  not extended ordinary >
    REFINES
        FAIL
    ANY
        s      >
        prop   >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >
        grd4:  suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem >
        grd5:  prop = run_peers(s)\(suspc_inst(token_owner(s) ↦ s) u
unav_peers) not theorem >

```

```

THEN
  act1:  inst_state := inst_state < ((prop×{s})×{FAIL_4}) >
  act2:  suspct_peers := suspct_peers < ((prop×{s})×{suspc_inst
(token_owner(s) ↦ s)}) >
  act3:  suspc_inst(token_owner(s) ↦ s) := ∅ >
END

RECONTACT_INST_OK:    not extended ordinary >
REFINES
  RECONTACT_INST_OK
ANY
  s    >a service s
  i    >an instance i
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
  grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
  grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
  grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
THEN
  act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
  act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
END

RECONTACT_INST_K0:    not extended ordinary >
REFINES
  RECONTACT_INST_K0
ANY
  s    >a service s
  i    >an instance i
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
  grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not

```

M17

```

theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊆ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
    THEN
        act1:  rect_inst(token_owner(s) ↦ s) = rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    END

FAIL_DETECT:  not extended ordinary >
    REFINES
        FAIL_DETECT
    ANY
        s >
        prop >
        susp >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >
        grd7:  susp ⊆ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
        grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
        grd5:  rect_inst(token_owner(s) ↦ s) = suspct_peers(token_owner
(s) ↦ s) not theorem >
        grd6:  prop = ((run_peers(s) \ suspct_peers(token_owner(s) ↦
s)) ∪ rctt_inst(token_owner(s) ↦ s)) \ unav_peers not theorem >
        grd8:  susp = suspct_peers(token_owner(s) ↦ s) \ rctt_inst
(token_owner(s) ↦ s) not theorem >
    THEN
        act1:  inst_state = inst_state ◀ ((prop×{s})×{FAIL_DETECT_4})
>
        act2:  suspct_peers = suspct_peers ◀ ((prop×{s})×{susp}) >
        act3:  rect_inst(token_owner(s) ↦ s) = ∅ >
        act4:  rctt_inst(token_owner(s) ↦ s) = ∅ >
    END

IS_OK:  not extended ordinary >
    REFINES
        IS_OK
    ANY
        s >
        prop >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >

```

M17

```

theorem >
    grd3: inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
    grd4: suspct_peers(token_owner(s) ↦ s) = ∅ not theorem >
    grd5: prop = run_peers(s)\unav_peers not theorem >
    THEN
    act1: inst_state := inst_state ◀ ((prop×{s})×{RUN_4}) >
    END

FAIL_ACTIV: not extended ordinary >
    REFINES
    FAIL_ACTIV
    ANY
    s >
    prop >
    WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    theorem >
    grd3: inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
    grd4: suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    (token_owner(s) ↦ s) not theorem >
    grd5: prop = run_peers(s) \ (unav_peers ∪ suspct_peers)
    THEN
    act1: inst_state := inst_state ◀ ((prop×{s})×{FAIL_ACTIV_4}) >
    (s) ↦ s) >
    act2: run_peers(s) := run_peers(s) \ suspct_peers(token_owner
    (token_owner(s) ↦ s) >
    act3: failr_peers(s) := failr_peers(s) ∪ suspct_peers
    act4: suspct_peers := suspct_peers ◀ ((prop×{s})×{∅}) >
    END

FAIL_CONFIGURE: extended ordinary >
    REFINES
    FAIL_CONFIGURE
    ANY
    s >
    prop >
    WHERE
    theorem >
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
    grd4: card(run_peers(s)) < min_inst(s) not theorem >
    grd5: prop = run_peers(s)\unav_peers not theorem >
    THEN
    act1: inst_state := inst_state ◀ ((prop×{s})×{FAIL_CONFIG_4}) >
    END

```

M17

```

FAIL_IGNORE:    extended ordinary >
  REFINES
    FAIL_IGNORE
  ANY
    s      >
    prop   >
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   prop ⊆ PEERS not theorem >
    grd3:   inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
    grd4:   card(run_peers(s)) ≥ min_inst(s) not theorem >
    grd5:   prop = run_peers(s)\unav_peers not theorem >
  THEN
    act1:   inst_state = inst_state ◀ ((prop×{s})×{FAIL_IGN_4}) >
  END

IGNORE:        extended ordinary >
  REFINES
    IGNORE
  ANY
    s      >
    prop   >
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   prop ⊆ PEERS not theorem >
    grd3:   inst_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem
>
    grd4:   prop = run_peers(s)\unav_peers not theorem >
  THEN
    act1:   inst_state = inst_state ◀ ((prop×{s})×{RUN_4}) >
  END

REDEPLOY_INSTC:  extended ordinary >
  REFINES
    REDEPLOY_INSTC
  ANY
    s      >a service s
    i      >an instance i
  WHERE
    grd1:   s ∈ SERVICES not theorem >
    grd2:   i ∈ PEERS not theorem >
    grd3:   i ∉ run_peers(s) ∪ failr_peers(s) ∪ unav_peers ∪
dep_instc(s) not theorem >i does not run s, is not failed for s, is not
unavailable and is not already activated for s
    grd4:   i ∉ actv_instc(token_owner(s) ↦ s) not theorem >
    grd5:   inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >

```



M17

```

theorem >          grd6:  card(activ_instc(token_owner(s) ↦ s)) < deplo_inst(s) not
not theorem >          grd7:  card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
THEN
(s) ↦ s) ∪ {i} >    act1:  activ_instc(token_owner(s) ↦ s) := activ_instc(token_owner
END
REDEPLOY_INSTS:    extended ordinary >
REFINES
REDEPLOY_INSTS
ANY
s >
WHERE
grd1:  s ∈ SERVICES not theorem >
grd2:  card(activ_instc(token_owner(s) ↦ s)) = deplo_inst(s) not
theorem >
grd3:  card(dep_instc(s)) + card(run_peers(s)) < min_inst(s)
not theorem >
grd4:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
THEN
act1:  dep_instc(s) := dep_instc(s) ∪ activ_instc(token_owner(s)
↦ s) >
act2:  activ_instc(token_owner(s) ↦ s) := ∅ >
END
REDEPLOY:    extended ordinary >
REFINES
REDEPLOY
ANY
s >
prop >
WHERE
grd1:  s ∈ SERVICES not theorem >
grd2:  prop ⊆ PEERS not theorem >
grd3:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
grd4:  activ_instc(token_owner(s) ↦ s) = ∅ not theorem >
grd5:  dep_instc(s) ≠ ∅ not theorem >
grd6:  card(run_peers(s)) + card(dep_instc(s)) ≥ min_inst(s) not
theorem >
grd7:  prop = run_peers(s) \ unav_peers not theorem >
THEN
act1:  inst_state := inst_state ◀ ((prop × {s}) × {DPL_4}) >
act2:  run_peers(s) := run_peers(s) ∪ dep_instc(s) >
act3:  dep_instc(s) := ∅ >

```

```

END

HEAL:   extended ordinary >
  REFINES
    HEAL
  ANY
    s   >
    prop >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = DPL_4 not theorem >
    grd4: prop = run_peers(s)\unav_peers not theorem >
  THEN
    act1: inst_state = inst_state ◀ ((prop×{s})×{RUN_4}) >
  END

UNFAIL_PEER:   extended ordinary >
  REFINES
    UNFAIL_PEER
  ANY
    s   >
    p   >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: p ∈ PEERS not theorem >
    grd3: p ∈ failr_peers(s) not theorem >
  THEN
    act1: failr_peers(s) := failr_peers(s)\{p} >
  END

MAKE_PEER_AVAIL:   extended ordinary >
  REFINES
    MAKE_PEER_AVAIL
  ANY
    p   >
  WHERE
    grd1: p ∈ PEERS not theorem >
    grd2: p ∈ unav_peers not theorem >
  THEN
    act1: unav_peers := unav_peers \ {p} >
  END

END

```

```

MACHINE
  M18 >
REFINES
  M17
SEES
  C09
VARIABLES
  run_inst >
  suspct_peers >
  failr_peers >
  dep_instc >
  token_owner >
  unav_peers >
  suspc_inst >
  rect_inst >instances that are tried to be recontacted
  rctt_inst >instances effectively recontacted after a try
  actv_instc >instances activated by token owne
  inst_state >
INVARIANTS
  inv1: run_inst ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2: ∀ s · s ∈ SERVICES ⇒ token_owner(s) ↦ s ∈ dom(run_inst) not
theorem >
  gluing_run_1: ∀ s · s ∈ SERVICES ⇒ run_inst(token_owner(s) ↦ s) =
run_peers(s) not theorem >
EVENTS
  INITIALISATION: not extended ordinary >
  THEN
    act1: run_inst := InitRunPeers >
    act2: suspct_peers := InitSuspPrs >
    act3: failr_peers := InitFail >
    act4: dep_instc := InitFail >
    act5: token_owner := init_tok >
    act6: unav_peers := ∅ >
    act7: suspc_inst := InitSuspPeers >
    act8: rect_inst := InitSuspPeers >
    act9: rctt_inst := InitSuspPeers >
    act10: actv_instc := InitSuspPeers >
    act11: inst_state := InitStateSrv >
  END
  MAKE_PEER_UNAVAIL: not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY
    prs >Peers that will become unavailable
    E >Values for token owner per service
  WHERE
    grd1: prs ⊆ PEERS not theorem >

```

grd2:  $\text{prs} \not\subseteq \text{unav\_peers}$  not theorem >the peers in prs are not yet unavailaible  
 grd3:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \Rightarrow \text{dom}(\text{dom}(\text{inst\_state}) \triangleright \{\text{srv}\}) \setminus \text{prs} \neq \emptyset$  not theorem >for each service srv, there must always be at least 1 peer available  
 grd4:  $E \in \text{SERVICES} \rightarrow \text{PEERS}$  not theorem >Value for token owner per service  
 grd5:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \notin \text{prs} \Rightarrow E(\text{srv}) = \text{token\_owner}(\text{srv})$  not theorem >If the token owner of a service srv does not belong to prs, the token owner is not changed  
 grd6:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs} \Rightarrow$   
 $E(\text{srv}) \in \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \setminus (\text{unav\_peers} \cup \text{prs} \cup \text{failr\_peers}(\text{srv}) \cup \text{suspct\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv})) \wedge$   
 $E(\text{srv}) \mapsto \text{srv} \in \text{dom}(\text{inst\_state}) \cap \text{dom}(\text{suspct\_peers}) \cap \text{dom}(\text{run\_inst}) \wedge$   
 $\text{run\_inst}(E(\text{srv}) \mapsto \text{srv}) = \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$   
 $\text{inst\_state}(E(\text{srv}) \mapsto \text{srv}) = \text{inst\_state}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$   
 $\text{suspct\_peers}(E(\text{srv}) \mapsto \text{srv}) = \text{suspct\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv})$  not theorem >if the owner of the token for a service becomes unavailable,

A new token owner is chosen: the new token owner must have same characteristics as the previous one (state, list of suspicious neighbours, etc.), and it must not be an unavailable, suspicious, failed peer or a member of prs

THEN

act1:  $\text{unav\_peers} := \text{unav\_peers} \cup \text{prs}$  >the peers in prs become unavailable  
 act2:  $\text{token\_owner} := \text{token\_owner} \triangleleft E$  >new values for token owner per service  
 act3:  $\text{rect\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rect\_inst}) \triangleleft (((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\})$  >the peers in prs can not try to recontact instances anymore (1)  
 act4:  $\text{rctt\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rctt\_inst}) \triangleleft (((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\})$  >the peers in prs can not try to recontact instances anymore (2)  
 act5:  $\text{actv\_instc} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{actv\_instc}) \triangleleft (((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\})$  >the peers in prs can not activate instances anymore  
 act6:  $\text{suspct\_peers} := (\text{prs} \times \text{SERVICES}) \triangleleft \text{suspct\_peers}$  >the peers in prs can not suspect instances anymore (1)  
 act7:  $\text{suspc\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{suspc\_inst}) \triangleleft$

```

(((E\token_owner)~)x{∅}) >the peers in prs can not suspect instances anymore (2)
  act8: inst_state := (prs×SERVICES) ◀ inst_state >the peers in
prs can not monitor the state of the services provided anymore
  act9: run_inst := (prs×SERVICES) ◀ run_inst >
  END

SUSPECT_INST: not extended ordinary >
  REFINES
    SUSPECT_INST
  ANY
    s >a service s
    susp >suspicious instances
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: susp ⊆ PEERS not theorem >
    grd3: susp = run_inst(token_owner(s) ↦ s) ∩ unav_peers not
theorem >instances in susp are suspicious if the peers running them becomes
unavailable
    grd4: suspc_inst(token_owner(s) ↦ s) = ∅ not theorem >the
member of susp have not yet been suspected for s by the token owner of s
    grd5: inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
    grd6: susp ≠ ∅ not theorem >
  THEN
    act1: suspc_inst(token_owner(s) ↦ s) := susp >the members of
susp become suspected instances for s by the token owner of s
  END

FAIL: not extended ordinary >
  REFINES
    FAIL
  ANY
    s >
    prop >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >
    grd4: suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5: prop = run_inst(token_owner(s) ↦ s) \ (suspc_inst
(token_owner(s) ↦ s) ∪ unav_peers) not theorem >
  THEN
    act1: inst_state := inst_state ◀ ((prop×{s})×{FAIL_4}) >
    act2: suspct_peers := suspct_peers ◀ ((prop×{s})×{suspc_inst
(token_owner(s) ↦ s)}) >
    act3: suspc_inst(token_owner(s) ↦ s) := ∅ >
  END

```

M18

```

RECONTACT_INST_OK:    extended ordinary >
  REFINES
    RECONTACT_INST_OK
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
    grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
  THEN
    act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
  END

RECONTACT_INST_K0:    extended ordinary >
  REFINES
    RECONTACT_INST_K0
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
    grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s

```

```

THEN
  act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
  END

FAIL_DETECT:  not extended ordinary >
  REFINES
    FAIL_DETECT
  ANY
    s      >
    prop   >
    susp   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd7:  susp ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
    grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5:  rect_inst(token_owner(s) ↦ s) = suspct_peers(token_owner
(s) ↦ s) not theorem >
    grd6:  prop = ((run_inst(token_owner(s) ↦ s) \ suspct_peers
(token_owner(s) ↦ s)) ∪ rctt_inst(token_owner(s) ↦ s)) \ unav_peers not theorem >
    grd8:  susp = suspct_peers(token_owner(s) ↦ s) \ rctt_inst
(token_owner(s) ↦ s) not theorem >
  THEN
    act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_DETECT_4})
>
    act2:  suspct_peers := suspct_peers ◀ ((prop×{s})×{susp}) >
    act3:  rect_inst(token_owner(s) ↦ s) := ∅ >
    act4:  rctt_inst(token_owner(s) ↦ s) := ∅ >
  END

IS_OK:  not extended ordinary >
  REFINES
    IS_OK
  ANY
    s      >
    prop   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
    grd4:  suspct_peers(token_owner(s) ↦ s) = ∅ not theorem >
    grd5:  prop = run_inst(token_owner(s) ↦ s) \ unav_peers not
theorem >
  THEN
    act1:  inst_state := inst_state ◀ ((prop×{s})×{RUN_4}) >

```

```

END

FAIL_ACTIV:      not extended ordinary >
REFINES
  FAIL_ACTIV
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s) \ (unav_peers ∪
suspct_peers(token_owner(s) ↦ s)) not theorem >
THEN
  act1:  inst_state = inst_state ◀ ((prop×{s})×{FAIL_ACTIV_4}) >
  act2:  run_inst = run_inst ◀ ((prop×{s})×{run_inst(token_owner
(s) ↦ s)\suspct_peers(token_owner(s) ↦ s)}) >
  act3:  failr_peers(s) = failr_peers(s) ∪ suspct_peers
(token_owner(s) ↦ s) >
  act4:  suspct_peers = suspct_peers ◀ ((prop×{s})×{∅}) >
END

FAIL_CONFIGURE:  not extended ordinary >
REFINES
  FAIL_CONFIGURE
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
  grd4:  card(run_inst(token_owner(s) ↦ s)) < min_inst(s) not
theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:  inst_state = inst_state ◀ ((prop×{s})×{FAIL_CONFIG_4}) >
END

FAIL_IGNORE:    not extended ordinary >
REFINES
  FAIL_IGNORE
ANY

```



```

s      >
prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
  grd4:  card(run_inst(token_owner(s) ↦ s)) ≥ min_inst(s) not
theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:  inst_state = inst_state ◀ ((prop×{s})×{FAIL_IGN_4}) >
END

IGNORE: not extended ordinary >
REFINES
  IGNORE
  ANY
  s      >
  prop   >
  WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem
>
  grd4:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:  inst_state = inst_state ◀ ((prop×{s})×{RUN_4}) >
END

REDEPLOY_INSTC: not extended ordinary >
REFINES
  REDEPLOY_INSTC
  ANY
  s      >a service s
  i      >an instance i
  WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  i ∉ run_inst(token_owner(s) ↦ s) ∪ failr_peers(s) ∪
unav_peers ∪ dep_instc(s) not theorem >i does not run s, is not failed for s, is
not unavailable and is not already activated for s
  grd4:  i ∉ actv_instc(token_owner(s) ↦ s) not theorem >
  grd5:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
  grd6:  card(actv_instc(token_owner(s) ↦ s)) < deplo_inst(s) not

```

```

theorem >
    grd7:  card(dep_instc(s)) + card(run_inst(token_owner(s) ↦ s))
< min_inst(s) not theorem >
    THEN
    act1:  actv_instc(token_owner(s) ↦ s) := actv_instc(token_owner
(s) ↦ s) ∪ {i} >
    END

REDEPLOY_INSTS:  not extended ordinary >
    REFINES
    REDEPLOY_INSTS
    ANY
    s >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  card(actv_instc(token_owner(s) ↦ s)) = deplo_inst(s) not
theorem >
    grd3:  card(dep_instc(s)) + card(run_inst(token_owner(s) ↦ s))
< min_inst(s) not theorem >
    grd4:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
    THEN
    act1:  dep_instc(s) := dep_instc(s) ∪ actv_instc(token_owner(s)
↦ s) >
    act2:  actv_instc(token_owner(s) ↦ s) := ∅ >
    END

REDEPLOY:  not extended ordinary >
    REFINES
    REDEPLOY
    ANY
    s >
    prop >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
    grd4:  actv_instc(token_owner(s) ↦ s)=∅ not theorem >
    grd5:  dep_instc(s) ≠ ∅ not theorem >
    grd6:  card(run_inst(token_owner(s) ↦ s))+card(dep_instc(s)) ≥
min_inst(s) not theorem >
    grd7:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
    THEN
    act1:  inst_state:= inst_state ◀ ((prop×{s})×{DPL_4}) >
    act2:  run_inst := run_inst ◀ ((prop×{s})× {run_inst(token_owner
(s) ↦ s) ∪ dep_instc(s)}) >

```

M18

```

act3:  dep_instc(s) := ∅ >
END

HEAL:  not extended ordinary >
REFINES
  HEAL
ANY
  s    >
  prop >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = DPL_4 not theorem >
  grd4:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:  inst_state= inst_state ◀ ((prop×{s})×{RUN_4}) >
END

UNFAIL_PEER:  extended ordinary >
REFINES
  UNFAIL_PEER
ANY
  s    >
  p    >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  p ∈ PEERS not theorem >
  grd3:  p ∈ failr_peers(s) not theorem >
THEN
  act1:  failr_peers(s) := failr_peers(s)\{p} >
END

MAKE_PEER_AVAIL:  extended ordinary >
REFINES
  MAKE_PEER_AVAIL
ANY
  p    >
WHERE
  grd1:  p ∈ PEERS not theorem >
  grd2:  p ∈ unav_peers not theorem >
THEN
  act1:  unav_peers := unav_peers \ {p} >
END

END

```

```

MACHINE
  M19 >
REFINES
  M18
SEES
  C09
VARIABLES
  run_inst >
  suspct_peers >
  failr_inst >
  dep_instc >
  token_owner >
  unav_peers >
  suspc_inst >
  rect_inst >instances that are tried to be recontacted
  rctt_inst >instances effectively recontacted after a try
  actv_instc >instances activated by token owne
  inst_state >
INVARIANTS
  inv1: failr_inst ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2: ∀ s · s ∈ SERVICES ⇒ token_owner(s) ↦ s ∈ dom(failr_inst) not
theorem >
  gluing_fail_1: ∀ s · s ∈ SERVICES ⇒ failr_inst(token_owner(s) ↦ s) =
failr_peers(s) not theorem >
EVENTS
  INITIALISATION: not extended ordinary >
  THEN
    act1: run_inst := InitRunPeers >
    act2: suspct_peers := InitSuspPrs >
    act3: failr_inst := InitSuspPeers >
    act4: dep_instc := InitFail >
    act5: token_owner := init_tok >
    act6: unav_peers := ∅ >
    act7: suspc_inst := InitSuspPeers >
    act8: rect_inst := InitSuspPeers >
    act9: rctt_inst := InitSuspPeers >
    act10: actv_instc := InitSuspPeers >
    act11: inst_state := InitStateSrv >
  END
  MAKE_PEER_UNAVAIL: not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY
    prs >Peers that will become unavailable
    E >Values for token owner per service
  WHERE
    grd1: prs ⊆ PEERS not theorem >

```

grd2:  $\text{prs} \not\subseteq \text{unav\_peers}$  not theorem >the peers in prs are not yet unavailaible  
 grd3:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \Rightarrow \text{dom}(\text{dom}(\text{inst\_state}) \triangleright \{\text{srv}\}) \setminus \text{prs} \neq \emptyset$  not theorem >for each service srv, there must always be at least 1 peer available  
 grd4:  $E \in \text{SERVICES} \rightarrow \text{PEERS}$  not theorem >Value for token owner per service  
 grd5:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \notin \text{prs} \Rightarrow E(\text{srv}) = \text{token\_owner}(\text{srv})$  not theorem >If the token owner of a service srv does not belong to prs, the token owner is not changed  
 grd6:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs} \Rightarrow$   
 $E(\text{srv}) \in \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \setminus (\text{unav\_peers} \cup \text{prs} \cup \text{failr\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \cup \text{suspct\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}))$   
 $\wedge$   
 $E(\text{srv}) \mapsto \text{srv} \in \text{dom}(\text{inst\_state}) \cap \text{dom}(\text{suspct\_peers}) \cap \text{dom}(\text{run\_inst}) \cap \text{dom}(\text{failr\_inst}) \wedge$   
 $\text{run\_inst}(E(\text{srv}) \mapsto \text{srv}) = \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$   
 $\text{inst\_state}(E(\text{srv}) \mapsto \text{srv}) = \text{inst\_state}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$   
 $\text{suspct\_peers}(E(\text{srv}) \mapsto \text{srv}) = \text{suspct\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$   
 $\text{failr\_inst}(E(\text{srv}) \mapsto \text{srv}) = \text{failr\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv})$  not theorem >if the owner of the token for a service becomes unavailable,

A new token owner is chosen: the new token owner must have same characteristics as the previous one (state, list of suspicious neighbours, etc.), and it must not be an unavailable, suspicious, failed peer or a member of prs

THEN

act1:  $\text{unav\_peers} := \text{unav\_peers} \cup \text{prs}$  >the peers in prs become unavailable  
 act2:  $\text{token\_owner} := \text{token\_owner} \triangleleft E$  >new values for token owner per service  
 act3:  $\text{rect\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rect\_inst}) \triangleleft ((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\}$  >the peers in prs can not try to recontact instances anymore (1)  
 act4:  $\text{rctt\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rctt\_inst}) \triangleleft ((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\}$  >the peers in prs can not try to recontact instances anymore (2)  
 act5:  $\text{actv\_instc} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{actv\_instc}) \triangleleft ((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\}$  >the peers in prs can not activate instances anymore  
 act6:  $\text{suspct\_peers} := (\text{prs} \times \text{SERVICES}) \triangleleft \text{suspct\_peers}$  >the peers

```

in prs can not suspect instances anymore (1)
  act7:  suspc_inst := ((prs×SERVICES) ◀ suspc_inst) ◀
(((E\token_owner)~)×{∅}) >the peers in prs can not suspect instances anymore (2)
  act8:  inst_state := (prs×SERVICES) ◀ inst_state >the peers in
prs can not monitor the state of the services provided anymore
  act9:  run_inst := (prs×SERVICES) ◀ run_inst >
  act10: failr_inst := (prs×SERVICES) ◀ failr_inst >
END

SUSPECT_INST:  extended ordinary >
REFINES
  SUSPECT_INST
ANY
  s      >a service s
  susp   >suspicious instances
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  susp ⊆ PEERS not theorem >
  grd3:  susp = run_inst(token_owner(s) ↦ s) ∩ unav_peers not
theorem >instances in susp are suspicious if the peers running them becomes
unavailable
  grd4:  suspc_inst(token_owner(s) ↦ s) = ∅ not theorem >the
member of susp have not yet been suspected for s by the token owner of s
  grd5:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
  grd6:  susp ≠ ∅ not theorem >
THEN
  act1:  suspc_inst(token_owner(s) ↦ s) := susp >the members of
susp become suspected instances for s by the token owner of s
END

FAIL:  extended ordinary >
REFINES
  FAIL
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >
  grd4:  suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s) \ (suspc_inst
(token_owner(s) ↦ s) ∪ unav_peers) not theorem >
THEN
  act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_4}) >
  act2:  suspct_peers := suspct_peers ◀ ((prop×{s})×{suspc_inst
(token_owner(s) ↦ s)}) >

```

```

act3:  suspc_inst(token_owner(s) ↦ s) := ∅ >
END

RECONTACT_INST_OK:  extended ordinary >
REFINES
  RECONTACT_INST_OK
ANY
  s    >a service s
  i    >an instance i
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
  grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
  grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
  grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
THEN
  act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
  act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
END

RECONTACT_INST_K0:  extended ordinary >
REFINES
  RECONTACT_INST_K0
ANY
  s    >a service s
  i    >an instance i
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
  grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\nunav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
  grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i

```

```

    grd7:  rect_inst(token_owner(s) ↦ s) ⊆ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
    THEN
        act1:  rect_inst(token_owner(s) ↦ s) = rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    END

FAIL_DETECT:  extended ordinary >
    REFINES
        FAIL_DETECT
    ANY
        s      >
        prop    >
        susp    >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >
        grd7:  susp ⊆ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
        grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
        grd5:  rect_inst(token_owner(s) ↦ s) = suspct_peers(token_owner
(s) ↦ s) not theorem >
        grd6:  prop = ((run_inst(token_owner(s) ↦ s) \ suspct_peers
(token_owner(s) ↦ s)) ∪ rctt_inst(token_owner(s) ↦ s)) \ unav_peers not theorem >
        grd8:  susp = suspct_peers(token_owner(s) ↦ s) \ rctt_inst
(token_owner(s) ↦ s) not theorem >
    THEN
        act1:  inst_state = inst_state ◀ ((prop×{s})×{FAIL_DETECT_4})
>
        act2:  suspct_peers = suspct_peers ◀ ((prop×{s})×{susp}) >
        act3:  rect_inst(token_owner(s) ↦ s) = ∅ >
        act4:  rctt_inst(token_owner(s) ↦ s) = ∅ >
    END

IS_OK:  extended ordinary >
    REFINES
        IS_OK
    ANY
        s      >
        prop    >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
        grd4:  suspct_peers(token_owner(s) ↦ s) = ∅ not theorem >
        grd5:  prop = run_inst(token_owner(s) ↦ s) \ unav_peers not

```



```

theorem >
  THEN
    act1: inst_state := inst_state << ((prop×{s})×{RUN_4}) >
  END

  FAIL_ACTIV: not extended ordinary >
  REFINES
    FAIL_ACTIV
  ANY
    s >
    prop >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
    grd4: suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5: prop = run_inst(token_owner(s) ↦ s) \ (unav_peers ∪
suspct_peers(token_owner(s) ↦ s)) not theorem >
  THEN
    act1: inst_state := inst_state << ((prop×{s})×{FAIL_ACTIV_4}) >
    act2: run_inst := run_inst << ((prop×{s})×{run_inst(token_owner
(s) ↦ s)\suspct_peers(token_owner(s) ↦ s)}) >
    act3: failr_inst := failr_inst << ((prop×{s})× {failr_inst
(token_owner(s) ↦ s) ∪ suspct_peers(token_owner(s) ↦ s)}) >
    act4: suspct_peers := suspct_peers << ((prop×{s})×{∅}) >
  END

  FAIL_CONFIGURE: extended ordinary >
  REFINES
    FAIL_CONFIGURE
  ANY
    s >
    prop >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
    grd4: card(run_inst(token_owner(s) ↦ s)) < min_inst(s) not
theorem >
    grd5: prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1: inst_state := inst_state << ((prop×{s})×{FAIL_CONFIG_4}) >
  END

  FAIL_IGNORE: extended ordinary >

```

```

REFINES
  FAIL_IGNORE
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
  grd4:  card(run_inst(token_owner(s) ↦ s)) ≥ min_inst(s) not
theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_IGN_4}) >
END

IGNORE: extended ordinary >
REFINES
  IGNORE
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem
>
  grd4:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:  inst_state := inst_state ◀ ((prop×{s})×{RUN_4}) >
END

REDEPLOY_INSTC: not extended ordinary >
REFINES
  REDEPLOY_INSTC
ANY
  s      >a service s
  i      >an instance i
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  i ∉ run_inst(token_owner(s) ↦ s) ∪ failr_inst
(token_owner(s) ↦ s) ∪ unav_peers ∪ dep_instc(s) not theorem >i does not run s,
is not failed for s, is not unavailable and is not already activated for s
  grd4:  i ∉ actv_instc(token_owner(s) ↦ s) not theorem >

```

M19

```

theorem >         grd5:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >         grd6:  card(activ_instc(token_owner(s) ↦ s)) < depl_inst(s) not
theorem >         grd7:  card(dep_instc(s)) + card(run_inst(token_owner(s) ↦ s))
< min_inst(s) not theorem >
      THEN
      act1:  activ_instc(token_owner(s) ↦ s) := activ_instc(token_owner
(s) ↦ s) ∪ {i} >
      END

REDEPLOY_INSTS:  extended ordinary >
      REFINES
      REDEPLOY_INSTS
      ANY
      s >
      WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  card(activ_instc(token_owner(s) ↦ s)) = depl_inst(s) not
theorem >
      grd3:  card(dep_instc(s)) + card(run_inst(token_owner(s) ↦ s))
< min_inst(s) not theorem >
      grd4:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
      THEN
      act1:  dep_instc(s) := dep_instc(s) ∪ activ_instc(token_owner(s)
↦ s) >
      act2:  activ_instc(token_owner(s) ↦ s) := ∅ >
      END

REDEPLOY:  extended ordinary >
      REFINES
      REDEPLOY
      ANY
      s >
      prop >
      WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  prop ⊆ PEERS not theorem >
      grd3:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
      grd4:  activ_instc(token_owner(s) ↦ s)=∅ not theorem >
      grd5:  dep_instc(s) ≠ ∅ not theorem >
      grd6:  card(run_inst(token_owner(s) ↦ s))+card(dep_instc(s)) ≥
min_inst(s) not theorem >
      grd7:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
      THEN

```

M19

```

    act1:  inst_state:= inst_state < ((prop×{s})×{DPL_4}) >
    act2:  run_inst := run_inst < ((prop×{s})× {run_inst(token_owner
(s) ↦ s) ∪ dep_instc(s)}) >
    act3:  dep_instc(s) := ∅ >
  END

HEAL:  extended ordinary >
  REFINES
    HEAL
  ANY
    s    >
    prop >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = DPL_4 not theorem >
    grd4:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1:  inst_state:= inst_state < ((prop×{s})×{RUN_4}) >
  END

UNFAIL_PEER:  not extended ordinary >
  REFINES
    UNFAIL_PEER
  ANY
    s    >
    p    >
    prop >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  p ∈ PEERS not theorem >
    grd4:  p ∈ failr_inst(token_owner(s) ↦ s) not theorem >
    grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1:  failr_inst := failr_inst < ((prop×{s})×{failr_inst
(token_owner(s) ↦ s) \ {p}}) >
  END

MAKE_PEER_AVAIL:  extended ordinary >
  REFINES
    MAKE_PEER_AVAIL
  ANY
    p    >
  WHERE
    grd1:  p ∈ PEERS not theorem >

```

M19

```
    grd2:  p ∈ unav_peers not theorem >  
THEN  
    act1:  unav_peers := unav_peers \ {p} >  
END
```

END

```

MACHINE
  M20    >
REFINES
  M19
SEES
  C09
VARIABLES
  run_inst      >
  suspct_peers  >
  failr_inst    >
  dep_instcs    >
  token_owner   >
  unav_peers    >
  suspc_inst    >
  rect_inst     >instances that are tried to be recontacted
  rctt_inst     >instances effectively recontacted after a try
  actv_instc    >instances activated by token owne
  inst_state    >
INVARIANTS
  inv1:  dep_instcs ∈ (PEERS×SERVICES) ↔ ℙ(PEERS) not theorem >
  inv2:  ∀ s · s ∈ SERVICES ⇒ token_owner(s) ↦ s ∈ dom(dep_instcs) not
theorem >
  gluing_act_1:  ∀ s · s ∈ SERVICES ⇒ dep_instcs(token_owner(s) ↦ s) =
dep_instc(s) not theorem >
EVENTS
  INITIALISATION:  not extended ordinary >
  THEN
    act1:  run_inst := InitRunPeers >
    act2:  suspct_peers := InitSuspPrs >
    act3:  failr_inst := InitSuspPeers >
    act4:  dep_instcs := InitSuspPeers >
    act5:  token_owner := init_tok >
    act6:  unav_peers := ∅ >
    act7:  suspc_inst := InitSuspPeers >
    act8:  rect_inst := InitSuspPeers >
    act9:  rctt_inst := InitSuspPeers >
    act10: actv_instc := InitSuspPeers >
    act11: inst_state := InitStateSrv >
  END
  MAKE_PEER_UNAVAIL:  not extended ordinary >
  REFINES
    MAKE_PEER_UNAVAIL
  ANY
    prs >Peers that will become unavailable
    E   >Values for token owner per service
  WHERE
    grd1:  prs ⊆ PEERS not theorem >

```

grd2:  $\text{prs} \not\subseteq \text{unav\_peers}$  not theorem >the peers in prs are not yet unavailaible  
 grd3:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \Rightarrow \text{dom}(\text{dom}(\text{inst\_state}) \triangleright \{\text{srv}\}) \setminus \text{prs} \neq \emptyset$  not theorem >for each service srv, there must always be at least 1 peer available  
 grd4:  $E \in \text{SERVICES} \rightarrow \text{PEERS}$  not theorem >Value for token owner per service  
 grd5:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \notin \text{prs} \Rightarrow E(\text{srv}) = \text{token\_owner}(\text{srv})$  not theorem >If the token owner of a service srv does not belong to prs, the token owner is not changed  
 grd6:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs} \Rightarrow E(\text{srv}) \in \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \setminus (\text{unav\_peers} \cup \text{prs} \cup \text{failr\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \cup \text{suspct\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}))$   
 $\wedge E(\text{srv}) \mapsto \text{srv} \in \text{dom}(\text{inst\_state}) \cap \text{dom}(\text{suspct\_peers}) \cap \text{dom}(\text{run\_inst}) \cap \text{dom}(\text{failr\_inst}) \cap \text{dom}(\text{dep\_instcs}) \wedge \text{run\_inst}(E(\text{srv}) \mapsto \text{srv}) = \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge \text{inst\_state}(E(\text{srv}) \mapsto \text{srv}) = \text{inst\_state}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge \text{suspct\_peers}(E(\text{srv}) \mapsto \text{srv}) = \text{suspct\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge \text{failr\_inst}(E(\text{srv}) \mapsto \text{srv}) = \text{failr\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge \text{dep\_instcs}(E(\text{srv}) \mapsto \text{srv}) = \text{dep\_instcs}(\text{token\_owner}(\text{srv}) \mapsto \text{srv})$  not theorem >if the owner of the token for a service becomes unavailable,

A new token owner is chosen: the new token owner must have same characteristics as the previous one (state, list of suspicious neighbours, etc.), and it must not be an unavailable, suspicious, failed peer or a member of prs

THEN

act1:  $\text{unav\_peers} := \text{unav\_peers} \cup \text{prs}$  >the peers in prs become unavailable  
 act2:  $\text{token\_owner} := \text{token\_owner} \triangleleft E$  >new values for token owner per service  
 act3:  $\text{rect\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rect\_inst}) \triangleleft (((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\})$  >the peers in prs can not try to recontact instances anymore (1)  
 act4:  $\text{rctt\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rctt\_inst}) \triangleleft (((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\})$  >the peers in prs can not try to recontact instances anymore (2)  
 act5:  $\text{actv\_instc} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{actv\_instc}) \triangleleft$

```

(((E\token_owner)~)×{∅}) >the peers in prs can not activate instances anymore
act6:  suspc_peers := (prs×SERVICES) ◀ suspc_peers >the peers
in prs can not suspect instances anymore (1)
act7:  suspc_inst := ((prs×SERVICES) ◀ suspc_inst) ◀
(((E\token_owner)~)×{∅}) >the peers in prs can not suspect instances anymore (2)
act8:  inst_state := (prs×SERVICES) ◀ inst_state >the peers in
prs can not monitor the state of the services provided anymore
act9:  run_inst := (prs×SERVICES) ◀ run_inst >
act10: failr_inst := (prs×SERVICES) ◀ failr_inst >
act11: dep_instcs := (prs×SERVICES) ◀ dep_instcs >
END

SUSPECT_INST:  extended ordinary >
REFINES
  SUSPECT_INST
ANY
  s      >a service s
  susp   >suspicious instances
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  susp ⊆ PEERS not theorem >
  grd3:  susp = run_inst(token_owner(s) ↦ s) ∩ unav_peers not
theorem >instances in susp are suspicious if the peers running them becomes
unavailable
  grd4:  suspc_inst(token_owner(s) ↦ s) = ∅ not theorem >the
member of susp have not yet been suspected for s by the token owner of s
  grd5:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
  grd6:  susp ≠ ∅ not theorem >
THEN
  act1:  suspc_inst(token_owner(s) ↦ s) := susp >the members of
susp become suspected instances for s by the token owner of s
END

FAIL:  extended ordinary >
REFINES
  FAIL
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >
  grd4:  suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s) \ (suspc_inst
(token_owner(s) ↦ s) ∪ unav_peers) not theorem >
THEN

```



```

act1:  inst_state := inst_state < ((prop×{s})×{FAIL_4}) >
act2:  suspct_peers := suspct_peers < ((prop×{s})×{suspc_inst
(token_owner(s) ↦ s)}) >
act3:  suspc_inst(token_owner(s) ↦ s) := ∅ >
END

RECONTACT_INST_OK:    extended ordinary >
REFINES
  RECONTACT_INST_OK
ANY
  s    >a service s
  i    >an instance i
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
  grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
  grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
  grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
THEN
  act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
  act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
END

RECONTACT_INST_K0:    extended ordinary >
REFINES
  RECONTACT_INST_K0
ANY
  s    >a service s
  i    >an instance i
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  i ∈ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
  grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be

```

contacted)

```

      grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
      grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
    THEN
      act1:  rect_inst(token_owner(s) ↦ s) = rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    END

```

```

FAIL_DETECT:  extended ordinary >
  REFINES
    FAIL_DETECT
  ANY
    s      >
    prop   >
    susp   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd7:  susp ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
    grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5:  rect_inst(token_owner(s) ↦ s) = suspct_peers(token_owner
(s) ↦ s) not theorem >
    grd6:  prop = ((run_inst(token_owner(s) ↦ s) \ suspct_peers
(token_owner(s) ↦ s)) ∪ rctt_inst(token_owner(s) ↦ s)) \ unav_peers not theorem >
    grd8:  susp = suspct_peers(token_owner(s) ↦ s) \ rctt_inst
(token_owner(s) ↦ s) not theorem >
  THEN
    act1:  inst_state = inst_state ◀ ((prop×{s})×{FAIL_DETECT_4})
>
    act2:  suspct_peers = suspct_peers ◀ ((prop×{s})×{susp}) >
    act3:  rect_inst(token_owner(s) ↦ s) = ∅ >
    act4:  rctt_inst(token_owner(s) ↦ s) = ∅ >
  END

```

```

IS_OK:  extended ordinary >
  REFINES
    IS_OK
  ANY
    s      >
    prop   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not

```

```

theorem >
    grd4:  suspct_peers(token_owner(s) ↦ s) = ∅ not theorem >
    grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
    THEN
    act1:  inst_state := inst_state ◀ ((prop×{s})×{RUN_4}) >
    END

FAIL_ACTIV:  extended ordinary >
    REFINES
    FAIL_ACTIV
    ANY
    s      >
    prop   >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
    grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5:  prop = run_inst(token_owner(s) ↦ s) \ (unav_peers ∪
suspct_peers(token_owner(s) ↦ s)) not theorem >
    THEN
    act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_ACTIV_4}) >
    act2:  run_inst := run_inst ◀ ((prop×{s})×{run_inst(token_owner
(s) ↦ s)\suspct_peers(token_owner(s) ↦ s)}) >
    act3:  failr_inst := failr_inst ◀ ((prop×{s})× {failr_inst
(token_owner(s) ↦ s) ∪ suspct_peers(token_owner(s) ↦ s)}) >
    act4:  suspct_peers := suspct_peers ◀ ((prop×{s})×{∅}) >
    END

FAIL_CONFIGURE:  extended ordinary >
    REFINES
    FAIL_CONFIGURE
    ANY
    s      >
    prop   >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
    grd4:  card(run_inst(token_owner(s) ↦ s)) < min_inst(s) not
theorem >
    grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
    THEN
    act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_CONFIG_4}) >

```

```

END

FAIL_IGNORE:    extended ordinary >
REFINES
  FAIL_IGNORE
ANY
  s      >
  prop   >
WHERE
  grd1:   s ∈ SERVICES not theorem >
  grd2:   prop ⊆ PEERS not theorem >
  grd3:   inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
  grd4:   card(run_inst(token_owner(s) ↦ s)) ≥ min_inst(s) not
theorem >
  grd5:   prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:   inst_state = inst_state ◀ ((prop×{s})×{FAIL_IGN_4}) >
END

IGNORE:    extended ordinary >
REFINES
  IGNORE
ANY
  s      >
  prop   >
WHERE
  grd1:   s ∈ SERVICES not theorem >
  grd2:   prop ⊆ PEERS not theorem >
  grd3:   inst_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem
>
  grd4:   prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:   inst_state = inst_state ◀ ((prop×{s})×{RUN_4}) >
END

REDEPLOY_INSTC:    not extended ordinary >
REFINES
  REDEPLOY_INSTC
ANY
  s      >a service s
  i      >an instance i
WHERE
  grd1:   s ∈ SERVICES not theorem >
  grd2:   i ∈ PEERS not theorem >
  grd3:   i ∉ run_inst(token_owner(s) ↦ s) ∪ failr_inst

```

```

(token_owner(s) ↦ s) ∪ unav_peers ∪ dep_instcs(token_owner(s) ↦ s) not theorem
>i does not run s, is not failed for s, is not unavailable and is not already
activated for s
    grd4: i ∉ actv_instc(token_owner(s) ↦ s) not theorem >
    grd5: inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
    grd6: card(actv_instc(token_owner(s) ↦ s)) < deplo_inst(s) not
theorem >
    grd7: card(dep_instcs(token_owner(s) ↦ s)) + card(run_inst
(token_owner(s) ↦ s)) < min_inst(s) not theorem >
    THEN
    act1: actv_instc(token_owner(s) ↦ s) = actv_instc(token_owner
(s) ↦ s) ∪ {i} >
    END

REDEPLOY_INSTS: not extended ordinary >
    REFINES
    REDEPLOY_INSTS
    ANY
    s >
    prop >
    WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: card(actv_instc(token_owner(s) ↦ s)) = deplo_inst(s) not
theorem >
    grd4: card(dep_instcs(token_owner(s) ↦ s)) + card(run_inst
(token_owner(s) ↦ s)) < min_inst(s) not theorem >
    grd5: inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
    grd6: prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
    THEN
    act1: dep_instcs = dep_instcs ◀ ((prop×{s})× {dep_instcs
(token_owner(s)↦s) ∪ actv_instc(token_owner(s)↦s)}) >
    act2: actv_instc(token_owner(s) ↦ s) = ∅ >
    END

REDEPLOY: not extended ordinary >
    REFINES
    REDEPLOY
    ANY
    s >
    prop >
    WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not

```

```

theorem >
    grd4:  actv_instc(token_owner(s) ↦ s)=∅ not theorem >
    grd5:  dep_instcs(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd6:  card(run_inst(token_owner(s) ↦ s))+card(dep_instcs
(token_owner(s) ↦ s)) ≥ min_inst(s) not theorem >
    grd7:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
    THEN
    act1:  inst_state:= inst_state ◀ ((prop×{s})×{DPL_4}) >
    act2:  run_inst := run_inst ◀ ((prop×{s})× {run_inst(token_owner
(s) ↦ s) ∪ dep_instcs(token_owner(s) ↦ s)}) >
    act3:  dep_instcs := dep_instcs ◀ ((prop×{s})×{∅}) >
    END

HEAL:  extended ordinary >
    REFINES
    HEAL
    ANY
    s >
    prop >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = DPL_4 not theorem >
    grd4:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
    THEN
    act1:  inst_state:= inst_state ◀ ((prop×{s})×{RUN_4}) >
    END

UNFAIL_PEER:  extended ordinary >
    REFINES
    UNFAIL_PEER
    ANY
    s >
    p >
    prop >
    WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  p ∈ PEERS not theorem >
    grd4:  p ∈ failr_inst(token_owner(s) ↦ s) not theorem >
    grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
    THEN
    act1:  failr_inst := failr_inst ◀ ((prop×{s})×{failr_inst
(token_owner(s) ↦ s) \ {p}}) >
    END

```

```
MAKE_PEER_AVAIL:  extended ordinary >
  REFINES
    MAKE_PEER_AVAIL
  ANY
    p >
  WHERE
    grd1:  p ∈ PEERS not theorem >
    grd2:  p ∈ unav_peers not theorem >
  THEN
    act1:  unav_peers := unav_peers \ {p} >
  END
END
```

M21

```

MACHINE
  M21 >
REFINES
  M20
SEES
  C09
VARIABLES
  run_inst >
  suspct_peers >
  failr_inst >
  dep_instcs >
  token_owner >
  unav_peers >
  suspc_inst >
  rect_inst >instances that are tried to be recontacted
  rctt_inst >instances effectively recontacted after a try
  actv_instc >instances activated by token owne
  inst_state >
INVARIANTS
  inv1: dom(run_inst)  $\subseteq$  dom(inst_state) not theorem >
EVENTS
  INITIALISATION: extended ordinary >
    THEN
      act1: run_inst := InitRunPeers >
      act2: suspct_peers := InitSuspPrs >
      act3: failr_inst := InitSuspPeers >
      act4: dep_instcs := InitSuspPeers >
      act5: token_owner := init_tok >
      act6: unav_peers :=  $\emptyset$  >
      act7: suspc_inst := InitSuspPeers >
      act8: rect_inst := InitSuspPeers >
      act9: rctt_inst := InitSuspPeers >
      act10: actv_instc := InitSuspPeers >
      act11: inst_state := InitStateSrv >
    END
  MAKE_PEER_UNAVAIL: not extended ordinary >
    REFINES
      MAKE_PEER_UNAVAIL
    ANY
      prs >Peers that will become unavailable
      E >Values for token owner per service
    WHERE
      grd1: prs  $\subseteq$  PEERS not theorem >
      grd2: prs  $\not\subseteq$  unav_peers not theorem >the peers in prs are not
yet unavailaible
      grd3:  $\forall$  srv  $\cdot$  srv  $\in$  SERVICES  $\Rightarrow$  dom(dom(inst_state)  $\triangleright$  {srv})
\pr  $\neq$   $\emptyset$  not theorem >for each service srv, there must always be at least 1

```



peer available

grd4:  $E \in \text{SERVICES} \rightarrow \text{PEERS}$  not theorem >Value for token owner

per service

grd5:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \notin \text{prs} \Rightarrow E(\text{srv}) = \text{token\_owner}(\text{srv})$  not theorem >If the token owner of a service  $\text{srv}$  does not belong to  $\text{prs}$ , the token owner is not changed

grd6:  $\forall \text{srv} \cdot \text{srv} \in \text{SERVICES} \wedge \text{token\_owner}(\text{srv}) \in \text{prs} \Rightarrow$   
 $E(\text{srv}) \in \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \setminus (\text{unav\_peers} \cup \text{prs} \cup \text{failr\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \cup \text{suspc\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}))$   
 $\wedge$

$E(\text{srv}) \mapsto \text{srv} \in \text{dom}(\text{run\_inst}) \cap \text{dom}(\text{suspc\_peers}) \cap \text{dom}(\text{failr\_inst}) \cap \text{dom}(\text{dep\_instcs}) \wedge$

$\text{run\_inst}(E(\text{srv}) \mapsto \text{srv}) = \text{run\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$

$\text{inst\_state}(E(\text{srv}) \mapsto \text{srv}) = \text{inst\_state}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$

$\text{suspc\_peers}(E(\text{srv}) \mapsto \text{srv}) = \text{suspc\_peers}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$

$\text{failr\_inst}(E(\text{srv}) \mapsto \text{srv}) = \text{failr\_inst}(\text{token\_owner}(\text{srv}) \mapsto \text{srv}) \wedge$

$\text{dep\_instcs}(E(\text{srv}) \mapsto \text{srv}) = \text{dep\_instcs}(\text{token\_owner}(\text{srv}) \mapsto \text{srv})$  not theorem >if the owner of the token for a service becomes unavailable,

A new token owner is chosen: the new token owner must have same characteristics

as the previous one (state, list of suspicious neighbours, etc.), and it must

not be an unavailable, suspicious, failed peer or a member of  $\text{prs}$

THEN

act1:  $\text{unav\_peers} := \text{unav\_peers} \cup \text{prs}$  >the peers in  $\text{prs}$  become unavailable

act2:  $\text{token\_owner} := \text{token\_owner} \triangleleft E$  >new values for token owner per service

act3:  $\text{rect\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rect\_inst}) \triangleleft ((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\}$  >the peers in  $\text{prs}$  can not try to recontact instances anymore (1)

act4:  $\text{rctt\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{rctt\_inst}) \triangleleft ((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\}$  >the peers in  $\text{prs}$  can not try to recontact instances anymore (2)

act5:  $\text{actv\_instc} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{actv\_instc}) \triangleleft ((E \setminus \text{token\_owner}) \sim) \times \{\emptyset\}$  >the peers in  $\text{prs}$  can not activate instances anymore

act6:  $\text{suspc\_peers} := (\text{prs} \times \text{SERVICES}) \triangleleft \text{suspc\_peers}$  >the peers in  $\text{prs}$  can not suspect instances anymore (1)

act7:  $\text{suspc\_inst} := ((\text{prs} \times \text{SERVICES}) \triangleleft \text{suspc\_inst}) \triangleleft$

M21

```

(((E\token_owner)~)x{∅}) >the peers in prs can not suspect instances anymore (2)
  act8: inst_state := (prs×SERVICES) ◀ inst_state >the peers in
prs can not monitor the state of the services provided anymore
  act9: run_inst := (prs×SERVICES) ◀ run_inst >
  act10: failr_inst := (prs×SERVICES) ◀ failr_inst >
  act11: dep_instcs := (prs×SERVICES) ◀ dep_instcs >
END

SUSPECT_INST: extended ordinary >
  REFINES
    SUSPECT_INST
  ANY
    s >a service s
    susp >suspicious instances
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: susp ⊆ PEERS not theorem >
    grd3: susp = run_inst(token_owner(s) ↦ s) ∩ unav_peers not
theorem >instances in susp are suspicious if the peers running them becomes
unavailable
    grd4: suspc_inst(token_owner(s) ↦ s) = ∅ not theorem >the
member of susp have not yet been suspected for s by the token owner of s
    grd5: inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >the
state of s is OK
    grd6: susp ≠ ∅ not theorem >
  THEN
    act1: suspc_inst(token_owner(s) ↦ s) := susp >the members of
susp become suspected instances for s by the token owner of s
  END

FAIL: extended ordinary >
  REFINES
    FAIL
  ANY
    s >
    prop >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = RUN_4 not theorem >
    grd4: suspc_inst(token_owner(s) ↦ s) ≠ ∅ not theorem >
    grd5: prop = run_inst(token_owner(s) ↦ s) \ (suspc_inst
(token_owner(s) ↦ s) ∪ unav_peers) not theorem >
  THEN
    act1: inst_state := inst_state ◀ ((prop×{s})×{FAIL_4}) >
    act2: suspct_peers := suspct_peers ◀ ((prop×{s})×{suspc_inst
(token_owner(s) ↦ s)}) >
    act3: suspc_inst(token_owner(s) ↦ s) := ∅ >

```

```

END

RECONTACT_INST_OK:    extended ordinary >
  REFINES
    RECONTACT_INST_OK
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
    grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is available (can be contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner
(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
  THEN
    act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    act2:  rctt_inst(token_owner(s) ↦ s) := rctt_inst(token_owner(s)
↦ s) ∪ {i} >i is recontacted by the token owner of s successfully
  END

RECONTACT_INST_K0:    extended ordinary >
  REFINES
    RECONTACT_INST_K0
  ANY
    s    >a service s
    i    >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >the
state of s is SUSPICIOUS
    grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >the
set of suspicious peers for s is not empty
    grd5:  i ∈ suspct_peers(token_owner(s) ↦ s)\unav_peers not
theorem >i is a suspicious instance of s and is unavailable (can not be
contacted)
    grd6:  i ∉ rect_inst(token_owner(s) ↦ s) not theorem >the token
owner of s has not yet tried to recontact i
    grd7:  rect_inst(token_owner(s) ↦ s) ⊂ suspct_peers(token_owner

```

M21

```

(s) ↦ s) not theorem >the token owner of s has not yet tried to recontact all
the suspicious instances of s
    THEN
        act1:  rect_inst(token_owner(s) ↦ s) := rect_inst(token_owner(s)
↦ s) ∪ {i} >the token owner of s has tried to recontact i
    END

FAIL_DETECT:  extended ordinary >
    REFINES
        FAIL_DETECT
    ANY
        s      >
        prop   >
        susp   >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >
        grd7:  susp ⊆ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = FAIL_4 not theorem >
        grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
        grd5:  rect_inst(token_owner(s) ↦ s) = suspct_peers(token_owner
(s) ↦ s) not theorem >
        grd6:  prop = ((run_inst(token_owner(s) ↦ s) \ suspct_peers
(token_owner(s) ↦ s)) ∪ rctt_inst(token_owner(s) ↦ s)) \ unav_peers not theorem >
        grd8:  susp = suspct_peers(token_owner(s) ↦ s) \ rctt_inst
(token_owner(s) ↦ s) not theorem >
    THEN
        act1:  inst_state := inst_state ◀ ((prop×{s})×{FAIL_DETECT_4})
>
        act2:  suspct_peers := suspct_peers ◀ ((prop×{s})×{susp}) >
        act3:  rect_inst(token_owner(s) ↦ s) := ∅ >
        act4:  rctt_inst(token_owner(s) ↦ s) := ∅ >
    END

IS_OK:  extended ordinary >
    REFINES
        IS_OK
    ANY
        s      >
        prop   >
    WHERE
        grd1:  s ∈ SERVICES not theorem >
        grd2:  prop ⊆ PEERS not theorem >
        grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
        grd4:  suspct_peers(token_owner(s) ↦ s) = ∅ not theorem >
        grd5:  prop = run_inst(token_owner(s) ↦ s) \ unav_peers not
theorem >

```

```

THEN
  act1:  inst_state := inst_state << ((prop×{s})×{RUN_4}) >
END

FAIL_ACTIV:  extended ordinary >
REFINES
  FAIL_ACTIV
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_DETECT_4 not
theorem >
  grd4:  suspct_peers(token_owner(s) ↦ s) ≠ ∅ not theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s) \ (unav_peers ∪
suspct_peers(token_owner(s) ↦ s)) not theorem >
THEN
  act1:  inst_state := inst_state << ((prop×{s})×{FAIL_ACTIV_4}) >
  act2:  run_inst := run_inst << ((prop×{s})×{run_inst(token_owner
(s) ↦ s)\suspct_peers(token_owner(s) ↦ s)}) >
  act3:  failr_inst := failr_inst << ((prop×{s})× {failr_inst
(token_owner(s) ↦ s) ∪ suspct_peers(token_owner(s) ↦ s)}) >
  act4:  suspct_peers := suspct_peers << ((prop×{s})×{∅}) >
END

FAIL_CONFIGURE:  extended ordinary >
REFINES
  FAIL_CONFIGURE
ANY
  s      >
  prop   >
WHERE
  grd1:  s ∈ SERVICES not theorem >
  grd2:  prop ⊆ PEERS not theorem >
  grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
  grd4:  card(run_inst(token_owner(s) ↦ s)) < min_inst(s) not
theorem >
  grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
THEN
  act1:  inst_state := inst_state << ((prop×{s})×{FAIL_CONFIG_4}) >
END

FAIL_IGNORE:  extended ordinary >
REFINES

```

```

    FAIL_IGNORE
  ANY
    s      >
    prop   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_ACTIV_4 not
theorem >
    grd4:  card(run_inst(token_owner(s) ↦ s)) ≥ min_inst(s) not
theorem >
    grd5:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1:  inst_state = inst_state ◀ ((prop×{s})×{FAIL_IGN_4}) >
  END

IGNORE: extended ordinary >
  REFINES
    IGNORE
  ANY
    s      >
    prop   >
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  prop ⊆ PEERS not theorem >
    grd3:  inst_state(token_owner(s) ↦ s) = FAIL_IGN_4 not theorem
>
    grd4:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1:  inst_state = inst_state ◀ ((prop×{s})×{RUN_4}) >
  END

REDEPLOY_INSTC: extended ordinary >
  REFINES
    REDEPLOY_INSTC
  ANY
    s      >a service s
    i      >an instance i
  WHERE
    grd1:  s ∈ SERVICES not theorem >
    grd2:  i ∈ PEERS not theorem >
    grd3:  i ∉ run_inst(token_owner(s) ↦ s) ∪ failr_inst
(token_owner(s) ↦ s) ∪ unav_peers ∪ dep_instcs(token_owner(s) ↦ s) not theorem
>i does not run s, is not failed for s, is not unavailable and is not already
activated for s
    grd4:  i ∉ actv_instc(token_owner(s) ↦ s) not theorem >

```

M21

```

theorem >          grd5:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >          grd6:  card(activ_instc(token_owner(s) ↦ s)) < depl_inst(s) not
theorem >          grd7:  card(dep_instcs(token_owner(s) ↦ s)) + card(run_inst
(token_owner(s) ↦ s)) < min_inst(s) not theorem >
      THEN
      act1:  activ_instc(token_owner(s) ↦ s) := activ_instc(token_owner
(s) ↦ s) ∪ {i} >
      END

REDEPLOY_INSTS:  extended ordinary >
      REFINES
      REDEPLOY_INSTS
      ANY
      s      >
      prop   >
      WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  prop ⊆ PEERS not theorem >
      grd3:  card(activ_instc(token_owner(s) ↦ s)) = depl_inst(s) not
theorem >
      grd4:  card(dep_instcs(token_owner(s) ↦ s)) + card(run_inst
(token_owner(s) ↦ s)) < min_inst(s) not theorem >
      grd5:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
      grd6:  prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
      THEN
      act1:  dep_instcs := dep_instcs ◀ ((prop×{s})× {dep_instcs
(token_owner(s)↦s) ∪ activ_instc(token_owner(s)↦s)}) >
      act2:  activ_instc(token_owner(s) ↦ s) := ∅ >
      END

REDEPLOY:  extended ordinary >
      REFINES
      REDEPLOY
      ANY
      s      >
      prop   >
      WHERE
      grd1:  s ∈ SERVICES not theorem >
      grd2:  prop ⊆ PEERS not theorem >
      grd3:  inst_state(token_owner(s) ↦ s) = FAIL_CONFIG_4 not
theorem >
      grd4:  activ_instc(token_owner(s) ↦ s)=∅ not theorem >
      grd5:  dep_instcs(token_owner(s) ↦ s) ≠ ∅ not theorem >
      grd6:  card(run_inst(token_owner(s) ↦ s))+card(dep_instcs

```

M21

```

(token_owner(s) ↦ s) ≥ min_inst(s) not theorem >
  grd7: prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1: inst_state = inst_state < ((prop×{s})×{DPL_4}) >
    act2: run_inst = run_inst < ((prop×{s})× {run_inst(token_owner
(s) ↦ s) ∪ dep_instcs(token_owner(s) ↦ s)}) >
    act3: dep_instcs = dep_instcs < ((prop×{s})×{∅}) >
  END

HEAL: extended ordinary >
  REFINES
    HEAL
  ANY
    s >
    prop >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: inst_state(token_owner(s) ↦ s) = DPL_4 not theorem >
    grd4: prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1: inst_state = inst_state < ((prop×{s})×{RUN_4}) >
  END

UNFAIL_PEER: extended ordinary >
  REFINES
    UNFAIL_PEER
  ANY
    s >
    p >
    prop >
  WHERE
    grd1: s ∈ SERVICES not theorem >
    grd2: prop ⊆ PEERS not theorem >
    grd3: p ∈ PEERS not theorem >
    grd4: p ∈ failr_inst(token_owner(s) ↦ s) not theorem >
    grd5: prop = run_inst(token_owner(s) ↦ s)\unav_peers not
theorem >
  THEN
    act1: failr_inst = failr_inst < ((prop×{s})×{failr_inst
(token_owner(s) ↦ s) \ {p}}) >
  END

MAKE_PEER_AVAIL: extended ordinary >
  REFINES
    MAKE_PEER_AVAIL

```



M21

```
ANY
  p >
WHERE
  grd1: p ∈ PEERS not theorem >
  grd2: p ∈ unav_peers not theorem >
THEN
  act1: unav_peers := unav_peers \ {p} >
END
```

END