Maryam Kamali | Linas Laibinis | Luigia Petre | Kaisa Sere

Reconstructing Coordination Links in Sensor-Actor Networks

Turku Centre for Computer Science

TUCS Technical Report
No 967, February 2010

Reconstructing Coordination Links in Sensor-Actor Networks

Maryam Kamali

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, FIN-20520 Turku, Finland
maryam.kamali@abo.fi

Linas Laibinis

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, FIN-20520 Turku, Finland
linas.laibinis@abo.fi

Luigia Petre

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, FIN-20520 Turku, Finland
luigia.petre@abo.fi

Kaisa Sere

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, FIN-20520 Turku, Finland
kaisa.sere@abo.fi

TUCS Technical Report

No 967, February 2010

Abstract

Wireless sensor-actor networks are a recent development of wireless networks where both ordinary sensor nodes and more sophisticated and powerful nodes, called *actors*, are present. The role of the actors is to take various decisions relevant for the network based on the data retrieved and transmitted by the sensors. In order to fulfill their role, actor nodes, independently of the sensor nodes, coordinate with each other via their own communication links. However, when an actor node fails, it may hinder the overall actor coordination. Hence, some backup mechanisms need to be enforced. In this paper we present a novel method on how to *always* enforce a reconstruction of the failed coordination links among the remaining active actors in a manner that aims to achieve various optimality properties. We argue that our method promotes a reusable coordination model of employing existing infrastructure as a fault-tolerance mechanism. Moreover, various forms of the coordination links are emphasized.

Keywords: Wireless Sensor Actor Networks (WSANs); Coordination links; Coordination recovery; Refinement; Event-B; Rodin Tool

TUCS Laboratory
Distributed Systems Lab

1 Introduction

Wireless Sensor Actor Networks (WSANs) are a rather new generation of sensor networks [8], made of two kinds of nodes: sensors and actors. In a WSAN, sensors detect the events that occur in the field, gather them and transmit the collected data to actors. The actors react to the events in the environment based on the received information. The sensor nodes are low-cost, low-power devices equipped with limited communication capabilities, while the actor nodes are usually mobile, more sophisticated and powerful devices compared to the sensor nodes. In addition, the density of sensor nodes in WSANs is much bigger than that of actor nodes.

WSANs are dynamic networks where the network topology continuously changes because some new links or nodes are added, or are removed due to their failure. A failure can occur due to hardware crashes, lack of energy, malfunctions, etc. Two central research topics concerning WSANs are *coordination* and *real-time requirements*. As there is no centralized control in a WSAN, sensors and actors need to coordinate with each other in order to collect information and take decisions on the next actions [8]. Also, depending on the application, it might be essential to respond to sensor inputs within predefined time limits, e.g., in critical applications such as forest fire detection.

There are three main types of WSAN coordination [11]: sensor-sensor, sensor-actor and actor-actor coordination, out of which we are here concerned with the latter. The sensor-sensor coordination in WSANs is similar to the Wireless Sensor Network (WSN) coordination, i.e., it defines how sensors route information, how information aggregates among them and which sensors are responsible for which tasks. The sensor-actor coordination prescribes which sensors should send certain data to certain actors. The actor-actor coordination focuses on the actor decisions and the division of tasks among different actors. To achieve the actor-actor coordination in WSANs, actors need reliable connection links for communicating with each other, which are established upon initializing the WSAN. However, actor nodes may fail during operation of the network. As a result, a WSAN may transform into several, disconnected WSAN sub-networks. This separation is called *network partitioning* and is illustrated in Figure 1, where the actor nodes $A_1 - A_{15}$ are shown to produce a network partitioning if actor node A_1 fails.

Due to the real-time requirements of WSANs, the failure of an actor node should not impact the whole actor network for too long. The problem of actor failing in the actor-actor coordination has been already addressed in [7, 1] by proposing the physical movement of actor nodes toward each other so that they can re-establish connectivity. However, during this movement, nodes in different partitions that have been created by the actor failure cannot coordinate. To shorten the time of recovery, Kamali et all [10] have previously proposed an algorithm for establishing new routes between non-failed actors via sensor nodes. This algorithm allows us to quickly connect the separated partitions, *before* moving actor

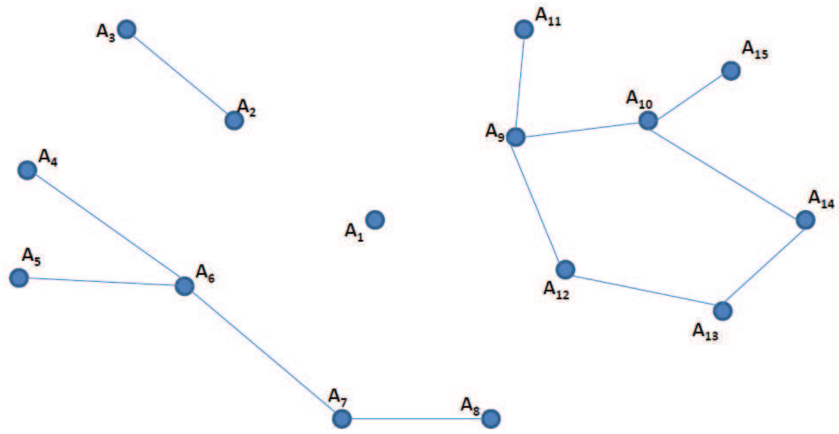

Figure 1: Three partitions created by a failed actor (A_1)

nodes as proposed in [7, 1]. In this paper, we further employ this recovery mechanism that alleviates the coordination failure.

There are several properties that are desirable to verify for this algorithm. First, we need to show that there is always a path via sensor nodes that can be established *by* the partitioned actor nodes. Second, it is desirable to guarantee that this path is the shortest, in order not to overload the power-limited sensor nodes. Third, to shorten the time of recovery as much as possible, it is desirable to establish the connection as soon as possible. In this paper we focus on ensuring the first property of the algorithm.

The novelty of our contribution is twofold. First, we formally prove that there is always a possibility to reconstruct a coordination link via sensor nodes between two non-failed actor nodes. Second, we guarantee that the coordination links are formed in a distributed manner and are (temporarily) delegated to sensors. Hence, the reconstruction is based only on local knowledge that actors have of their neighbor actors and the nearby sensors. These properties stress the WSAN strength as an innovative coordination model. Specifically, actor nodes which are more sophisticated devices are used to implement varied, possibly quite complex decision making behavior, based on the information provided by the sensors. In their turn, the sensor nodes not only dutifully collect and transmit data but also act as fault tolerance support for the coordination links between actor nodes. Therefore, the sensor nodes provide the backup infrastructure on which the actor coordination can rely while applying a simple distributed algorithm that essentially computes the best alternatives for these links. Thus, our contribution puts forward an additional role for the data gathering infrastructure.

In order to prove the local path existence property, we employ the Event-B formal method. Event-B [4, 2, 3] is an extension of the B formalism [5] for specifying distributed and reactive systems. A system model is gradually specified on several levels of abstraction, always ensuring that a more concrete model is a *correct implementation* of an abstract model. The language and proof theory of Event-B are based on logic and set theory. The correctness of the stepwise construction of formal models is ensured by discharging a set of proof obligations: if these obligations hold, then the development is mathematically shown to be correct. Event-B comes with the associated tool Rodin [2, 12], which automatically discharges part of the proof obligations and also provides the means for the user to discharge interactively the remaining proofs.

This paper is organized as follows. In Section 2 we briefly overview the Event-B formalism and present the recovery algorithm. In Section 3 we present the recovery mechanism at four levels of abstraction in Event-B. In Section 4 we conclude with some final remarks.

2 Preliminaries

This section briefly overviews our modeling formalism Event-B and also describes the recovery algorithm to be modeled in this paper.

Event-B Each Event-B model consists of two components called *context* and *machine*. A context describes the static part of the model, i.e., it introduces new types and constants. The properties of these types and constants are gathered as a list of axioms. An example of an Event-B context is shown in the appendix. A machine represents the dynamic part of the model, consisting of model variables and operations called *events*. The structure of an Event-B machine is given in Figure 2. The system properties that should be preserved during the execution are formulated as a list of *invariant* predicates over the state of the model.

An event, modeling state changes, is composed of a *guard* and an *action*. The guard is the necessary condition under which an event might occur; if the guard holds, we call the event *enabled*. The action determines the way in which the state variables change when the event occurs. For initializing the system, a sequence of actions is defined. When the guards of several events hold at the same time, then only one event is non-deterministically chosen for execution. If some events have no variables in common and are enabled at the same time, then they can be considered to be executed in parallel since their sequential execution in any order gives the same result.

A model is developed by a number of correctness preserving steps called *refinements*. One form of model refinement can add new data and new behavior events on top of the already existing data and behavior but in such a way that the introduced behavior does not contradict or take over the abstract machine behavior. In addition to this *superposition* refinement [9] we may also use other refinement forms, such as *algorithmic refinement* [6]. In this case, an event of an abstract machine can be refined by several corresponding events in a refined machine. This will model different branches of execution, that can for instance

```
MACHINE machine-name
  VARIABLES list of variables
  INVARIANTS list of invariants/predicates
  EVENTS
    INITIALIZATION
    BEGIN
      list of actions
    END
    event-name
    WHEN
      list of guards
    THEN
      list of actions
    END
END
```

Figure 2: MACHINE definition in Event-B

take place in parallel and thus can improve the algorithmic efficiency.

The recovery algorithm In this algorithm, the detection of a failed node leads to the communication links among non-failed actor nodes to be reconstructed via sensor nodes. The mechanism has three parts: detecting a failed actor, selecting the shortest path, and establishing the selected path through sensor nodes. When actor neighbors of an actor node do not receive any acknowledgment from that actor node, they detect it as failed. At this time, the neighbors of the failed node have to investigate whether this failure has produced separated partitions. If there is no partitioning, then nothing is done except updating the neighbor lists in nodes. However, if there are some separated partitions, a new path should be selected and established.

In our algorithm, we refer to paths at two levels, one at the actor level and the other at the sensor level. The *length* of a path refers to the number of edges making up the path. To connect all the separated partitions, we need at least one path of length equal to the number of partitions minus one. For instance, in order to connect three partitions, we need at least one path among actors of length two. We assume that each actor node has information about its immediate neighbors (1-hop neighbors) and 2-hop neighbors (the neighbors of the neighbors).

Upon detecting a failed actor node, the actor neighbors of the failed actor node need to re-establish their connections. These connections are formed based on the node *degree* information (the number of immediate neighbors) and on the relative distance between actor nodes.

3 Four levels of abstraction for the Recovery Algorithm

In this section we formally develop the algorithm for reconstructing coordination links among actor nodes as explained in Section 2. Our purpose is to show that, given a network of sensors and a network of actors above them, the actors can *always* reconstruct coordination links between themselves, by using local information and sensors as intermediate nodes. In order to prove this property, we first model the network at three increasing levels of detail so that each model is a refinement of the previous one. In the initial model, we very abstractly specify a network of generic nodes and the recovery mechanism. In the second model, we add new data and events to model the list of 1-hop and 2-hop neighbors for every node. In the third model, we distinguish among sensor and actor nodes and their corresponding networks. Moreover, the non-failed actor nodes aim to establish the shortest path among themselves. We therefore have a fourth abstraction level where details about the physical distance between actor nodes is taken into consideration for establishing the path. In the following, we describe these models.

3.1 The Initial Model

The context of our initial model contains the definition of sets and constants as well as our model assumptions as axioms. A finite (axiom 6) and non-empty (axiom 7), generic set $NODE$ describes all the network nodes. We assume at this point that all the nodes are homogeneous, that is, we do not distinguish between actors and sensors yet. We also define two constants, $FAIL$ and $closure$. The constant $FAIL$ denotes the set $\{0, 1\}$, where 1 stands for a failed node and 0 for a non-failed node (axiom 1). The constant $closure$ models the transitive closure of a binary relation on the set $NODE$ (axioms 2-5).

axioms:
- **@axm1** $FAIL = \{0, 1\}$
- **@axm2** $closure \in (NODE \leftrightarrow NODE) \rightarrow$
  $(NODE \leftrightarrow NODE)$
- **@axm3** $\forall r \cdot r \subseteq closure(r)$
- **@axm4** $\forall r \cdot closure(r); r \subseteq closure(r)$
- **@axm5** $\forall r, s \cdot r \subseteq s \land s; r \subseteq s \Rightarrow closure(r) \subseteq s$
- **@axm6** $finite(NODE)$
- **@axm7** $NODE \neq \varnothing$

In the machine part of our initial model we have five events and five invariants as shown below. The status of each node (non-failed or failed) is modeled with the function N mapping each node in $NODE$ to 0 or 1 (invariant 1). The relation NET denotes the bidirectional links that are non-failed (invariant 2 and 5). This relation is non-reflexive (invariant 3) and symmetric (invariant 4). This means that, a NET link from a node to itself is prohibited. Moreover, if node A has a link with node B, then node B also has a link with the node A. We also model that the network is active continuously with theorem **THM1** that ensures that always, at least one event is enabled (i.e., the disjunction of all the events guards is true). An invariant has to be checked every time an event is chosen and executed, even if it has been proven to hold at a previous execution of that event. In contrast, once we prove a theorem for a model, we need not prove it again. We chose to model the continuous activity of the network with a theorem to avoid reproving this property at each execution.

INVARIANTS
@inv1 $N \in NODE \rightarrow FAIL$
@inv2 $NET \in dom(N) \leftrightarrow dom(N)$
@inv3 $dom(N) \lhd id \cap NET = \varnothing$
@inv4 $NET = NET \sim$
@inv5 $\forall n, m \cdot n \mapsto m \in NET \Rightarrow n \mapsto 0 \in N \land m \mapsto 0 \in N$
theorem @THM1$(\exists l \cdot l \mapsto 1 \in N)$
$\lor(\exists n, m \cdot n \mapsto 0 \in N \land m \mapsto 0 \in N \land n \mapsto m \notin NET$
$\land\, m \mapsto n \notin NET \land n \neq m)$
$\lor(\exists p \cdot p \mapsto 1 \in N)$
$\lor(\exists i, j, k \cdot i \mapsto 0 \in N \land j \mapsto 1 \in N \land k \mapsto 0 \in N \land$
$i \mapsto j \notin NET \land k \mapsto j \notin NET \land$
$i \neq j \land j \neq k \land k \neq i \land i \mapsto k \notin closure(NET))$

The initialization event sets all the nodes to 1, i.e., failed (action 1); therefore, the NET relation should be empty (action 2), based on invariant 5. Except initialization, the events in the initial model add nodes (**AddNode**) and links (**AddLink**), remove nodes and their corresponding links (**RemoveNode**) and also abstractly recovery connections when a node fails(**FaultDetRec**).

INITIALIZATION
then
@act1 $N := NODE \times \{1\}$
@act2 $NET := \varnothing$
AddNode
any n **where**
@grd1 $n \mapsto 1 \in N$
then
@act1 $N := N \Leftarrow \{n \mapsto 0\}$
end

In the **AddNode** event, every node that is added overwrites the function N, using the overwriting operator $\Leftarrow$ in Event-B.

In the **AddLink** event, we add a link in both directions to meet invariant 4.

AddLink
  any n m **where**
    @grd1 $n \mapsto 0 \in N \wedge m \mapsto 0 \in N$
    @grd2 $n \mapsto m \notin NET \wedge m \mapsto n \notin NET$
    @grd3 $n \neq m$
  then
    @act1 $NET := NET \cup \{n \mapsto m\} \cup \{m \mapsto n\}$
  end

The **RemoveNode** event changes the status of a node from 0 to 1; also, all the links of that node are removed from NET, expressed with the domain substraction operator $\lhd$ and the range substraction operator $\rhd$.

RemoveNode
  any n **where**
    @grd1 $n \mapsto 0 \in N$
  then
    @act1 $N := N \Leftarrow \{n \mapsto 1\}$
    @act2 $NET := \{n\} \lhd NET \rhd \{n\}$
  end

Removing a node from the network can lead to some separated network partitions. The event **FaultDetRec** detects whether a removed node has created separated partitions or not. If two nodes had no connection through other nodes (i.e., there was no path from one node to the other, expressed by guard 4 of **FaultDetRec**), then a partition is formed and, at this abstract level, simply a new path is established.

FaultDetRec
  any n m k **where**
    @grd1 $n \mapsto 0 \in N \wedge m \mapsto 1 \in N \wedge k \mapsto 0 \in N$
    @grd2 $n \mapsto m \notin NET \wedge k \mapsto m \notin NET$
    @grd3 $m \neq n \wedge m \neq k \wedge n \neq k$
    @grd4 $n \mapsto k \notin closure(NET)$
  then
    @act1 $NET := NET \cup \{n \mapsto k, k \mapsto n\}$
  end

Overall, our initial model abstractly describes the non-deterministic addition and removal of nodes and links in a dynamic (wireless sensor-actor) network for whom the network partitioning problem can be detected and recovered from. At this level we only model that a failed node is detected and new links among remaining nodes are established, without discussing the details of how these links can be added.

3.2 The Second Model

In the initial model we have considered the network having knowledge about itself while in our algorithm we assume that each node has access only to information of its 1-hop neighbors and 2-hop neighbors. We now refine the initial model and define a new relation l_net that, for each node, keeps track of the 1-hop and 2-hop neighbors. The relation l_net relates three nodes as defined by invariant 1 below and is non-reflexive, modeled by invariant 2 below. The meaning of this relation is that a 1-hop neighbor m of a node n is denoted by $n \mapsto m \mapsto m \in l\_net$ and a 2-hop neighbor m of a node n is denoted by $n \mapsto m \mapsto k \in l\_net$. In the first example, m is locally related to n via m (itself, i.e., via a direct link) and in the second example m is locally related to n via k (i.e., m is a 2-hop neighbor of n, while k is a 1-hop neighbor of n). The relation l_net describes all these *localized* links between nodes. The goal of this refinement step is to supplement the global knowledge of the network in the initial model with a localized knowledge formalized with the relation l_net.

@inv1 $l\_net \in NODE \times NODE \leftrightarrow NODE$
@inv2 $dom(N) \lhd id \cap dom(l\_net) = \varnothing$

When a new link is added between two nodes the l_net relation also needs to be updated. Therefore, the **AddLink** event is refined to also add links to l_net. For every two nodes n and m which have a direct link, $n \mapsto m \mapsto m$ and $m \mapsto n \mapsto n$ are added, meaning that n has a link with m through m (m is a 1-hop neighbor n) and m has a link with n through n (n is a 1-hop neighbor of m).

AddLink
  extends AddLink
  then
    @act2 $l\_net := l\_net \cup \{n \mapsto m \mapsto m, m \mapsto n \mapsto n\}$
  end

The **Addl_net2hopLink** event is a newly introduced event that handles the addition of 2-hop neighbor links for nodes. If a node has a direct link with two nodes, then these nodes will be 2-hop neighbors of each other:

Addl_net2hopLink
  any n m k **where**
    @grd1 $n \mapsto 0 \in N \wedge m \mapsto 0 \in N \wedge k \mapsto 0 \in N$
    @grd2 $m \mapsto k \mapsto k \in l\_net \wedge n \mapsto m \mapsto m \in l\_net \wedge$
      $n \mapsto k \mapsto m \notin l\_net \wedge k \mapsto n \mapsto m \notin l\_net$
    @grd3 $m \neq n \wedge n \neq k \wedge m \neq k$
  then
    @act1 $l\_net := l\_net \cup \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}$
  end

When removing a node, all its connections should be removed. Thus, in the **RemoveNode** event a new action is added which removes all the immediate links with the failed node in the l_net relation. The expression $\{n\} \times NODE \times NODE$ describes all the links of n, either direct connections (1-hop neighbors) or indirect connections (2-hop neighbors) and the expression $dom(NET) \times \{n\} \times \{n\}$ describes all the links between immediate neighbors of n and n.

> **RemoveNode**
>   **extends** RemoveNode
>   **then**
>   **@act3** $l\_net :\mid l\_net' \subseteq l\_net \setminus ((\{n\} \times NODE \times NODE) \cup$
> $(dom(NET) \times \{n\} \times \{n\}))$
>   **end**

Detecting failed nodes and recovering links should be managed locally instead of being based on all the network topology described by NET. We now use l_net information in addition to NET for detecting an actor failure (guard 5) and recovering links in the **FaultDetRec** event.

> **FaultDetRec**
>   **refines** FaultDetRec
>   **any** n m k **where**
>   **@grd5** $n \mapsto k \mapsto m \in l\_net \land n \mapsto m \mapsto m \notin l\_net \land$
> $k \mapsto n \mapsto m \in l\_net \land k \mapsto m \mapsto m \notin l\_net$
>   **then**
>   **@act2** $l\_net :\mid l\_net' \subseteq (l\_net \setminus (\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\} \cup$
> $(NET[\{n\}] \times \{m\} \times \{n\}) \cup (NET[\{k\}] \times \{m\} \times \{k\})))$
> $\cup (NET[\{k\}] \times \{n\} \times \{k\}) \cup (\{n\} \times NET[\{k\}] \times \{k\})$
> $\cup (NET[\{n\}] \times \{k\} \times \{n\}) \cup (\{k\} \times NET[\{n\}] \times \{n\}) \cup$
> $(\{k\} \times \{n\} \times (NODE \setminus \{m\})) \cup (\{n\} \times \{k\} \times (NODE \setminus \{m\}))$

When node m is detected as a failed node, neighbors of m (n and k) that have a connection with each other through m ($n \mapsto k \mapsto m$ and $k \mapsto n \mapsto m$) need to find an alternative path toward each other. If there is no other route in NET ($n \mapsto k \notin closure(NET)$), then l_net should be updated by removing expired links and addding new routes. Since m is failed, links between n and k through m are not valid, so $n \mapsto k \mapsto m$ and $k \mapsto n \mapsto m$ is removed from l_net. In addition, links describing the immediate neighbors of n ($NET[\{n\}]$) and of k ($NET[\{k\}]$) to m via n and k, respectively are removed from l_net. The second phase of the updating process is adding new links to connect n and k. In this refinement, since we still have no information about sensors, we define that node n can establish a link with k through any node except m which is failed: $\{n\} \times \{k\} \times (NODE \setminus \{m\})$ and similarly for node k to establish a new link with n: $\{k\} \times \{n\} \times (NODE \setminus \{m\})$. When node n establishes a link with k, neighbors of n also need to add node k to their 2-hop neighbors list ($NET[\{n\}] \times \{k\} \times \{n\}$). Moreover, neighbors of k need to add n to their 2-hop neighbors list ($NET[\{k\}] \times \{n\} \times \{k\}$). The updating process of l_net is described by action 2 in the **FaultDetRec** event.

We also add another new event **FaultDetRec2**. This event treats the situation when a failure is detected but an alternative path already exists between the neighbors of the failed node ($n \mapsto k \in closure(NET)$). In this case, l_net is simply updated by removing all the links with the failed node or through it.

FaultDecRec2
  any n m k **where**
    @grd1 $n \mapsto 0 \in N \wedge m \mapsto 1 \in N \wedge k \mapsto 0 \in N$
    @grd2 $n \neq m \wedge m \neq k \wedge n \neq k$
    @grd3 $n \mapsto k \mapsto m \in l\_net \wedge n \mapsto m \mapsto m \notin l\_net \wedge$
        $k \mapsto n \mapsto m \in l\_net \wedge k \mapsto m \mapsto m \notin l\_net$
    @grd4 $n \mapsto k \in closure(NET)$
    @grd5 $n \mapsto m \notin NET \wedge k \mapsto m \notin NET$
  then
    @act1 $l\_net := l\_net \setminus (\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\} \cup$
        $(NET[\{n\}] \times \{m\} \times \{n\}) \cup (NET[\{k\}] \times \{m\} \times \{k\}))$

We observe that l_net is an elegant data structure relating two nodes in its domain via a third node in its range. The following model further employs this data structure.

3.3 The Third Model

In this model, we distinguish sensor and actor nodes and specify more concretely how replacement links are added after detecting an actor failure. We introduce two new relations on $dom(N)$, $SNET$ and $SANET$ (invariant 1 and invariant 2), the former representing links among sensor nodes and the latter depicting links between sensor and actor nodes.

@inv1 $SNET \in dom(N) \leftrightarrow dom(N)$
@inv2 $SANET \in dom(N) \leftrightarrow dom(N)$
@inv3 $SNET \cap NET = \varnothing$
@inv4 $NET \cap SANET = \varnothing$
@inv5 $SNET \cap SANET = \varnothing$
@inv6 $SNET = SNET \sim$
@inv7 $SANET = SANET \sim$
@inv8 $dom(N) \lhd id \cap SNET = \varnothing$
@inv9 $dom(N) \lhd id \cap SANET = \varnothing$
@inv10 $\forall n, m \cdot n \mapsto m \in SANET \Rightarrow (K(n) = 0 \land K(m) = 1)$
$\quad \lor (K(m) = 0 \land K(n) = 1)$
@inv11 $\forall n, m \cdot n \mapsto m \in SNET \Rightarrow n \mapsto 0 \in N \land m \mapsto 0 \in N$
@inv12 $\forall n, m \cdot n \mapsto m \in SANET \Rightarrow n \mapsto 0 \in N \land m \mapsto 0 \in N$
@inv13 $NET \in (K \sim)[\{0\}] \leftrightarrow (K \sim)[\{0\}]$
@inv14 $SNET \in (K \sim)[\{1\}] \leftrightarrow (K \sim)[\{1\}]$
@inv15 $dom(l\_net) \in (K \sim)[\{0\}] \leftrightarrow (K \sim)[\{0\}]$
@inv16 $\forall n, k, x, y \cdot n \mapsto k \mapsto x \in l\_net \land k \mapsto n \mapsto y \in l\_net \land$
$\quad x \mapsto 1 \in K \land y \mapsto 1 \in K \Rightarrow x \in SANET[\{n\}]$
$\quad \land y \in SANET[\{k\}] \land x \mapsto y \in closure(SNET)$

These relations describe links between nodes at a different level, hence they are disjoint from the actor links modeled by NET (invariant 3 and invariant 4). $SNET$ and $SANET$ are also disjoint sets (invariant 5). Moreover, they are symmetric and non-reflexive sets as shown by invariants 6-9. To differentiate between sensor and actor nodes, we define a constant K with the following axiom: 0 represents actor nodes and 1 represents sensor nodes.

@axm1 $K \in NODE \rightarrow \{0, 1\}$

We also formalize that for each link $n \mapsto m$ in $SANET$ one of these nodes should be a sensor node and the other one should be an actor node (invariant 10). The next two invariants (invariant 11 and 12) model that every node of a link in either $SNET$ or $SANET$ should be non-failed. Invariant 13 and invariant 14 show that the nodes of every link in NET and $SNET$ should belong to actor nodes and sensor nodes, respectively. We have defined the l_net relation to connect 1-hop and 2-hop neighbors via a third part. We now restrict the domain of l_net to only actors (invariant 15) as we are interested in re-establishing connections between actors. At the same time, the range of l_net is free: this models that actors that are 2-hop neighbors can be connected via an actor or via a sensor. Invariant 16 models that if there is a link between two actor nodes via sensor nodes in l_net, the involved sensor nodes are within the range of l_net, the respective actor-sensor links belong to $SANET$ and the sensors themselves have at least one path toward each other within $closure(SNET)$.

In the previous model, removing a node and all its connections was modeled by the **RemoveNode** event. In this model we refine **RemoveNode** by adding a new action for updating $SANET$ after omitting an actor node (action 4). Also, all connections through sensor nodes towards a failed node should be removed from l_net (action 3). In addition, we add a guard restricting n to being an actor (guard 2).

RemoveNode
  refines RemoveNode
  any n **where**
    @grd2 $n \mapsto 0 \in K$
    then
    @act3 $l\_net := l\_net \setminus ((\{n\} \times NODE \times NODE) \cup$
      $(dom(NET) \times \{n\} \times \{n\}) \cup (dom(NET) \times \{n\} \times dom(SNET)))$
    @act4 $SANET := \{n\} \lhd SANET \rhd \{n\}$
  end

The only differences in **AddLink** and **Addl_net2hopLink** events with respect to the previous model are their new guards. These guards prevent the addition of links between two nodes which are not actor nodes because NET and l_net sets depict the links between just actor nodes.

AddLink
  extends AddLink
  where
    @grd4 $n \mapsto 0 \in K$
    @grd5 $m \mapsto 0 \in K$
  end

Addl_net2hopLink
  extends Addl_net2hopLink
  where
    @grd4 $n \mapsto 0 \in K \wedge m \mapsto 0 \in K \wedge k \mapsto 0 \in K$
  end

In this model we have two new events for adding links between sensor nodes in $SNET$ and links between sensor and actor nodes in $SANET$: **AddSLink** and **AddSALink**.

AddSLink
  any n m **where**
    @grd1 $n \mapsto 0 \in N \wedge m \mapsto 0 \in N$
    @grd2 $n \notin dom(NET) \wedge m \notin dom(NET)$
    @grd3 $n \mapsto m \notin SNET$
    @grd4 $n \neq m$
    @grd5 $n \mapsto 1 \in K \wedge m \mapsto 1 \in K$
  then
    @act1 $SNET := SNET \cup \{n \mapsto m, m \mapsto n\}$
  end

AddSALink
  any n m **where**
    @grd1 $n \mapsto 0 \in N \wedge m \mapsto 0 \in N$
    @grd2 $(K(n) = 0 \wedge K(m) = 1) \vee (K(n) = 1 \wedge K(m) = 0)$
    @grd3 $n \mapsto m \notin SANET$
    @grd4 $n \neq m$
  then
    @act1 $SANET := SANET \cup \{n \mapsto m, m \mapsto n\}$
  end

The **AddSLink** event is similar to **AddLink** with a different guard that models that, for every map $n \mapsto m$ added in **AddSLink**, n and m should be sensor nodes. The **AddSALink** event is for adding links between sensor and actors.

The event **FaultDetRec** which models the recovery mechanism after an actor failure is refined using information of $SNET$ and $SANET$. Compared to the previous version of the event, there are two additional parameters x, y as sensor nodes, that have connections with actor nodes n and k, respectively. Also, x and y have either a direct link or an indirect one towards each other (retrieved in $closure(SNET)$). Moreover, the actors n and k have no connection with each other (guard 9)

FaultDetRec
  refines FaultDetRec
  any n m k x y **where**
    @grd6 $x \in SANET[\{n\}] \wedge y \in SANET[\{k\}]$
    @grd7 $x \mapsto y \in closure(SNET)$
    @grd8 $m \mapsto 0 \in K$
    @grd9 $n \mapsto k \notin dom(l\_net \setminus \{n \mapsto k \mapsto m\})$
  then
    @act2 $l\_net := (l\_net \setminus (\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}$
      $\cup(NET[\{n\}] \times \{m\} \times \{n\}) \cup (NET[\{k\}] \times \{m\} \times \{k\})))$
      $\cup(NET[\{k\}] \times \{n\} \times \{k\}) \cup (\{n\} \times NET[\{k\}] \times \{k\})$
      $\cup(NET[\{n\}] \times \{k\} \times \{n\}) \cup (\{k\} \times NET[\{n\}] \times \{n\})$
      $\cup\{n \mapsto k \mapsto x, k \mapsto n \mapsto y\}$
  end

The action 2 in **FaultDetRec** was non-deterministic in the previous model. We now refine this assignment to a deterministic one. We replace $\{k\} \times \{n\} \times NODE \setminus \{m\}$ with $k \mapsto n \mapsto y$ and similarly $\{n\} \times \{k\} \times NODE \setminus \{m\}$ is replaced with $n \mapsto k \mapsto x$.

The action in the **FaultDetRec2** event is unchanged. However, we strengthen the guard of the event by adding guard 6 that guarantees the existence of a link between two direct neighbors of a failed node via other nodes than the failed one.

FaultDetRec2
  refines FaultDetRec2
  where
    @grd6 $n \mapsto k \in dom(l\_net \setminus \{n \mapsto k \mapsto m\})$
  end

3.4 The Fourth Model

Our machine in the second model re-establishes connections through sensor nodes between pairs of actor nodes which were direct neighbors of a failed actor node. However, this is not an optimal mechanism since actor nodes can be far from each other and involve numerous sensor nodes to re-establish the connection, while there might be a shorter path for this. To determine the shortest path between these actor nodes we need to introduce information about the physical location of the nodes. In this model we add two new (function) variables $locX$ and $locY$ that store the cartesian coordinates (x, y) of each node (invariant 21 and invariant 22).

As explained in Section 2, one of the direct neighbors of a failed node with highest degree starts to calculate its distance with other direct neighbors of the failed node and select this sub-path as replacement alternative if a path can be established through sensor nodes. Next, node with the second highest degree starts to calculate its distance with others. This process continues till all sub-partitions connect together.

We define the $degree$ function to calculate the degree of each neighbor of the failed node (invariant 20). To calculate the degree of such a neighbor of the failed node, the list of the failed node neighbors is modeled by the **failedNodeNeigh** variable (invariant 19). In order to establish a complete path between the partitions, we need to disable all the events which are not involved in the recovery procedure. We define the boolean variable $flag$ (invariant 17) for this, with the meaning that $flag = TRUE$ enables all the events not involved in the recovery mechanism and $flag = FALSE$ disables them and enables the recovery mechanism events. Invariant 24 models that when there is no needed recovery ($flag = TRUE$), then the degree variable is also empty (we are not interested in the degree function when there is no recovery model).

Invariants

@inv17 $flag \in BOOL$
@inv19 $failedNodeNeigh \subseteq dom(N)$
@inv20 $degree \in dom(N) \nrightarrow 0..card(dom(N))$
@inv21 $locX \in dom(N) \rightarrow 0..1000$
@inv22 $locY \in dom(N) \rightarrow 0..1000$
@inv23 $failedNodeNeigh \cap dom(degree) = \varnothing$
@inv24 $flag = TRUE \Rightarrow degree = \varnothing$

The refined **AddNode** event also stores the location of the node when a node is added to the network.

AddNode

extends AddNode
any i j
where
@grd3 $flag = TRUE$
@grd4 $i \in 1..1000$
@grd5 $j \in 1..1000$
then
@act2 $locX := locX \Leftarrow \{n \mapsto i\}$
@act3 $locY := locY \Leftarrow \{n \mapsto j\}$
end

When a node fails in **RemoveNode** event, a flag sets to enable events to recover this failure (action 5) in the network and the $FailedNodeNeigh$ is filled by neighbors of the failed node. Assume $m \mapsto k \mapsto n \in l\_net$. Then, $l\_net \sim$ is the inverse of l_net, hence $l\_net \sim [\{n\}]$ denotes the neighbors (1 hop and 2 hop) having either via n, in our example it denotes $m \mapsto k$. Hence, $dom(l\_net \sim [\{n\}])$ denotes the 1-hop neighbors of n, in our example which is modeled by action 6.

RemoveNode

extends RemoveNode
where
@grd3 $flag = TRUE$
then
@act5 $flag := FALSE$
@act6 $failedNodeNeigh := dom(l\_net \sim [\{n\}])$
end

In this refinement, a new event is added to calculate the degree of nodes in $failedNodeNeigh$. For each member n of this list, n adds to the $degree$ variable the map $n \mapsto card(NET[\{n\}]$ (action 1). We also remove that node from the list $failedNodeNeigh$. This ensures invariant 24. The $degree$ and $failedNodeNeigh$ are temporary variables needed in the recovering process. When the **Degree** event is enabled, the rest of events are disabled and when $failedNodeNeigh$ becomes empty then **FaultDetRec** becomes enabled. To model this sequentiality, in the same time when we add elements to the $degree$ variable (action 1), we remove them from the $failedNodeNeigh$ variable (action 2). When we have finished the degree calculation, then $failedNodeNeigh$ is empty and **FailedDetRec** becomes enabled, in order to start the recovery mechanism.

Degree
  any n
  where
    @grd1 $flag = FALSE$
    @grd2 $failedNodeNeigh \neq \varnothing$
    @grd3 $n \in failedNodeNeigh$
  then
  @act1 $degree := degree \cup \{n \mapsto card(NET[\{n\}])\}$
  @act2 $failedNodeNeigh := failedNodeNeigh \setminus \{n\}$
  end

FaultDetRec

extends FaultDetRec

where

@grd10 $flag = FALSE$

@grd11 $failedNodeNeigh = \varnothing$

@grd12 $n \in dom(degree) \land k \in dom(degree)$

@grd13 $degree(n) > min(dom(degree \sim))$

@grd14 $\forall i \cdot i \in dom(\{n, k\} \mathbin{⩥} degree) \Rightarrow (locX(n) - locX(k))$
$*(locX(n) - locX(k)) + (locY(n) - locY(k))*$
$(locY(n) - locY(k)) < (locX(n) - locX(i))*$
$(locX(n) - locX(i)) + (locY(n) - locY(i))$
$*(locY(n) - locY(i))$

@grd15 $degree(k) > degree(n) \Rightarrow (\exists i \cdot i \in dom(\{n, k\} \lhd degree)$
$\land(locX(k) - locX(i)) * (locX(k) - locX(i))+$
$(locY(k) - locY(i)) * (locY(k) - locY(i))$
$< (locX(k) - locX(n)) * (locX(k) - locX(n))$
$+(locY(k) - locY(n)) * (locY(k) - locY(n)))$

then

@act3 $degree := \{n\} \lhd degree$

end

As explained before, **FaultDetRec** is enabled when the $flag$ variable evaluates to FALSE (guard 10) and the $failedNodeNeigh$ variable evaluates to $\varnothing$ (guard 11). Guard 12 ensures that nodes n and k are neighbors of the failed node m. As we intended to create a path as short as possible its length should be the number of partitions minus one. This means that the node with the lowest degree does not need to calculate anything (guard 13). For any other neighbor of m except the node with the minimum degree, the shortest distance to other nodes in degree is selected (guard 14). Gaurd 14 ensures that the distance between n and k is the smallest, i.e., any other neighbor i of m is further away from n than k. We also need to ensure that the path between n and k has not already been chosen by k. This situation can not occur if $degree(k) < degree(n)$, but if $degree(k) > degree(n)$ then we need to ensure that k already established a path to another neighbor i of m. This is modeled by guard 15. The square distance d between nodes i and j is calculated as $pow(d) = (locX(i) - locX(j)) * (locX(i) - locX(j)) + (locY(i) - locY(j)) * (locY(i) - locY(j))$.

When a node selects its path, it is removed from the $degree$ (action 3). When all needed sub-paths are established, then the $flag$ should be changed to enable all the other events relating to the recovery mechanism. This flag update is shown in the **Flag** event.

Table 1: Proof Statistics

Model	Number of Proof Obligations	Automatically Discharged	Interactively Discharged
Context	4	4(100%)	0(0%)
Initial Model	26	15(58%)	11(42%)
1st Refinement	19	13(68%)	6(32%)
2nd Refinement	95	34(36%)	61(64%)
3rd Refinement	35	32(91%)	3(9%)
Total	179	98(54%)	81(46%)

Flag
  where
    @grd1 $flag = FALSE$
    @grd2 $card(dom(degree)) = 1$
    @grd3 $failedNodeNeigh = \varnothing$
  then
    @act1 $flag := TRUE$
    @act2 $degree := \varnothing$
  end

3.5 Proof Statistics

The proof statistics of our development are shown in Table 1. These figures express the number of proof obligations generated by the Rodin Platform as well as the number of obligations automatically discharged by the platform and those interactively proved.

4 Conclusion

In this paper, we have formalized a distributed recovery algorithm in Event-B. The algorithm addresses the network partitioning problem in WSANs generated by actor failures. We have modeled the algorithm and the correspondent actor coordination links at four increasing levels of abstraction that refine each other. We have proved the refinement formally using the theorem prover tool Rodin [12]. The most interesting aspect put forward with our refinement modeling is the development of an actor coordination link that can be seen in three forms: a direct actor-actor link, an indirect, not further specified path, or an indirect path through sensor nodes. We have developed this link as a refinement with the precise purpose of replacing the first (failed) form with the third one. However, the refinement shows that all the three forms can be present in a network and thus provide various coordination alternatives for actors. In this respect, one can define coordination classes, e.g., for delegating the most security sensitive coordination to the direct actor-actor coordination links, the least real-time constrained coordination to indirect links, and the safety critical coordination to both direct actor links and indirect sensor paths between actors. This observation can prove very useful in practice.

Using the sensor infrastructure as temporary backup for actor coordination also aligns with the growing *sustainability* research of using resources without depleting them. Upon detecting a direct actor-actor coordination link between two actor nodes, all sensor nodes contributing to a communication link between these actor nodes should be released of their backup task, a feature outside the scope of this paper.

Our formal WSAN model is the first attempt at formalizing WSAN algorithms in Event-B and hence the WSAN model can be much extended. For instance, non-deterministically adding and removing nodes is a useful feature for these networks as it models their dynamic scalability mechanism as well as their uncontrollable failures. However, non-deterministically adding links is just an abstraction for nodes detecting each other in wireless range and connecting via various protocols. Hence, the WSAN formal modeling space is quite generous and we intend to investigate it further, e.g., by modeling various temporal properties as well as real-time aspects and verifying various other algorithms too.

References

[1] A. Abbasi, K. Akkaya, M. Younis, *A Distributed Connectivity Restoration Algorithm in Wireless Sensor and Actor Networks*, 32nd IEEE Conference on Local Computer Networks (LCN), Dublin, Ireland, March 2007.

[2] J. R. Abrial, *A system development process with Event-B and the Rodin platform*, Butler, M., Hinchey, M. G. and Larrondo-Petrie, M. M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 1-3. Springer, 2007.

[3] J. R. Abrial. Event Driven Distributed Program Construction. http://www.atelierb.societe.com/ressources/articles/dis.pdf, 2001.

[4] J. R. Abrial, *Modeling in Event-B: System and Software Design*, Cambridge University Press, Cambridge(to appear 2010).

[5] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[6] R. J. Back and K. Sere. Stepwise Refinement of Action Systems. In *J. L. A. van de Snepscheut (ed), Proceedings of MPC'89 – Mathematics of Program Construction*, pp. 115-138, 1989.

[7] K. Akkaya and M. Younis, *COLA: A Coverage and Latency Aware Actor Placement for Wireless Sensor and Actor Networks*, IEEE Vehicular Technology Conference (VTC-Fall06), Montreal, Canada, September 2006.

[8] I. F. Akyildiz, and I. H. Kasimoglu,*Wireless Sensor and Actor Networks: Research Challenges*, Vol. 2, No. 4, pp. 351-367. Elsevier Ad hoc Network Journal, 2004.

[9] S. Katz. A Superimposition Control Construct for Distributed Systems. In *ACM Transactions on Programming Languages and Systems*, Vol. 15, No. 2, pp. 337-356, 1993.

[10] M. Kamali, S. Sedighian and M. Sharifi, *A Distributed Recovery Mechanism for Actor-Actor Connectivity in Wireless Sensor Actor Networks*, pp. 183-188, IEEE ISSNIP, Australia, 2008.

[11] T. Melodia, D. Pompili, V. C. Gungor and I. F. Akyildiz *Communication and Coordination in Wireless Sensor and Actor Networks*, Vol. 6 No. 10, pp. 1116-1129 IEEE Transactions on Mobile Computing, 2007.

[12] RODIN tool platform, http://www.event-b.org/platform.html.

5 Appendix

An Event-B Specification of Model_ctx
Creation Date: 28 Jan 2010 @ 01:28:41 PM

CONTEXT Model_ctx

SETS

NODE

CONSTANTS

FAIL

closure

AXIOMS

axm1 : $FAIL = \{0, 1\}$

axm2 : $closure \in (NODE \leftrightarrow NODE) \rightarrow (NODE \leftrightarrow NODE)$

axm3 : $\forall r \cdot r \subseteq closure(r)$

axm4 : $\forall r \cdot closure(r); r \subseteq closure(r)$

axm5 : $\forall r, s \cdot r \subseteq s \land s; r \subseteq s \Rightarrow closure(r) \subseteq s$

axm6 : $\forall r \cdot r = r^{-1} \Rightarrow closure(r) = (closure(r))^{-1}$

axm7 : $finite(NODE)$

axm8 : $NODE \neq \varnothing$

END

An Event-B Specification of Model
Creation Date: 28 Jan 2010 @ 01:32:03 PM

MACHINE Model

SEES Model_ctx

VARIABLES

N

NET

INVARIANTS

inv1 : $N \in NODE \rightarrow FAIL$

each node has just one state failed or non-failed

inv2 : $NET \in dom(N) \leftrightarrow dom(N)$

inv3 : $dom(N) \lhd id \cap NET = \varnothing$

non-reflexive NET

inv4 : $NET = NET^{-1}$

symmetric NET

theoremthm1 : $(\exists l \cdot l \mapsto 1 \in N) \lor (\exists p \cdot p \mapsto 0 \in N) \lor (\exists n, m \cdot n \mapsto 0 \in N \land m \mapsto 0 \in N \land n \mapsto m \notin NET \land m \mapsto n \notin NET \land n \neq m) \lor (\exists i, j, k \cdot i \mapsto 0 \in N \land j \mapsto 0 \in N \land k \mapsto 0 \in N \land i \mapsto j \notin NET \land k \mapsto j \notin NET \land i \neq j \land i \neq k \land k \neq j \land i \mapsto k \notin closure(NET))$

inv6 : $\forall n, m \cdot n \mapsto m \in NET \Rightarrow n \mapsto 0 \in N \land m \mapsto 0 \in N$

EVENTS

Initialisation

begin

act1 : $N := NODE \times \{1\}$

all nodes are failed

act2 : $NET := \varnothing$

NET without any link

end

Event $AddNode \widehat{=}$

any

n

where

grd1 : $n \mapsto 1 \in N$

then

act1 : $N := N \Leftarrow \{n \mapsto 0\}$

a node becomes non-failed

end

Event *RemoveNode* $\widehat{=}$

remove (node+ all its connections)

any

n

where

grd1 : $n \mapsto 0 \in N$

then

act1 : $N := N \Leftarrow \{n \mapsto 1\}$

a node fails

act2 : $NET := \{n\} \lhd NET \rhd \{n\}$

end

Event *AddLink* $\widehat{=}$

any

n

m

where

grd1 : $n \mapsto 0 \in N \land m \mapsto 0 \in N$

nodes n,m are non-failed

grd2 : $n \mapsto m \notin NET \land m \mapsto n \notin NET$

grd3 : $n \neq m$

then

act1 : $NET := NET \cup \{n \mapsto m\} \cup \{m \mapsto n\}$

end

Event *FaultDetRec* $\widehat{=}$

if partitioning happened in the NET it recovers the problem

any

n

m

k

where

grd1 : $n \mapsto 0 \in N \land m \mapsto 1 \in N \land k \mapsto 0 \in N$

grd2 : $n \mapsto m \notin NET \land k \mapsto m \notin NET$

grd3 : $m \neq n \land m \neq k \land n \neq k$

grd4 : $n \mapsto k \notin closure(NET)$

then

act1 : $NET := NET \cup \{n \mapsto k, k \mapsto n\}$
force to keep the NET connected (establish again the missing link
)

end

END

An Event-B Specification of Model_r
Creation Date: 28 Jan 2010 @ 07:43:54 PM

MACHINE Model_r

REFINES Model

SEES Model_ctx

VARIABLES

N

NET

l_net

INVARIANTS

inv1 : $l\_net \in NODE \times NODE \leftrightarrow NODE$

inv2 : $dom(N) \lhd id \cap dom(l\_net) = \varnothing$

EVENTS

Initialisation

begin

act3 : $N := NODE \times \{1\}$

act2 : $NET := \varnothing$

act1 : $l\_net := \varnothing$

end

Event *AddNode* $\widehat{=}$

extends *AddNode*

any

n

where

grd1 : $n \mapsto 1 \in N$

then

act1 : $N := N \lhd\!\!- \{n \mapsto 0\}$
a node becomes non-failed

end

Event *RemoveNode* $\widehat{=}$

extends *RemoveNode*

any

$\mathtt{n}$

where

grd1 : $\mathtt{n} \mapsto 0 \in \mathtt{N}$

then

act1 : $\mathtt{N} := \mathtt{N} \lhd\!\!\!- \{\mathtt{n} \mapsto 1\}$

a node fails

act2 : $\mathtt{NET} := \{\mathtt{n}\} \lhd\!\!\!\!- \mathtt{NET} \rhd\!\!\!\!- \{\mathtt{n}\}$

act3 : $l\_net : |l\_net' \subseteq l\_net \setminus ((\{n\} \times NODE \times NODE) \cup (dom(NET) \times \{n\} \times \{n\}))$

immediate neighbors of a failed node delete their links with it

end

Event *AddLink* $\widehat{=}$

extends *AddLink*

any

$\mathtt{n}$

$\mathtt{m}$

where

grd1 : $\mathtt{n} \mapsto 0 \in \mathtt{N} \land \mathtt{m} \mapsto 0 \in \mathtt{N}$

nodes n,m are non-failed

grd2 : $\mathtt{n} \mapsto \mathtt{m} \notin \mathtt{NET} \land \mathtt{m} \mapsto \mathtt{n} \notin \mathtt{NET}$

grd3 : $\mathtt{n} \neq \mathtt{m}$

then

act1 : $\mathtt{NET} := \mathtt{NET} \cup \{\mathtt{n} \mapsto \mathtt{m}\} \cup \{\mathtt{m} \mapsto \mathtt{n}\}$

act2 : $l\_net := l\_net \cup \{n \mapsto m \mapsto m, m \mapsto n \mapsto n\}$

end

Event *Addl_net2hoplink* $\widehat{=}$

any

n

m

k

where

grd1 : $n \mapsto 0 \in N \land m \mapsto 0 \in N \land k \mapsto 0 \in N$

grd2 : $m \mapsto k \mapsto k \in l\_net \land n \mapsto m \mapsto m \in l\_net \land n \mapsto k \mapsto m \notin l\_net \land k \mapsto n \mapsto m \notin l\_net$

grd3 : $m \neq n \land n \neq k \land m \neq k$

then

act1 : $l\_net := l\_net \cup \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}$

adding 2hop neighbors of each node

end

Event *FaultDetRec* $\widehat{=}$

refines *FaultDetRec*

any

n on node

m failed node

k on node

where

grd1 : $n \mapsto 0 \in N \land m \mapsto 1 \in N \land k \mapsto 0 \in N$

grd2 : $n \neq m \land m \neq k \land n \neq k$

grd3 : $n \mapsto k \mapsto m \in l\_net \land n \mapsto m \mapsto m \notin l\_net \land k \mapsto n \mapsto m \in l\_net \land k \mapsto m \mapsto m \notin l\_net$

grd4 : $n \mapsto k \notin closure(NET)$

grd5 : $n \mapsto m \notin NET \land k \mapsto m \notin NET$

then

act1 : $NET := NET \cup \{n \mapsto k, k \mapsto n\}$

act2 : $l\_net : |l\_net' \subseteq (l\_net \setminus ((\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}) \cup (NET[\{n\}] \times \{m\} \times \{n\}) \cup (NET[\{k\}] \times \{m\} \times \{k\}))) \cup (NET[\{k\}] \times \{n\} \times \{k\}) \cup (\{n\} \times NET[\{k\}] \times \{k\}) \cup (NET[\{n\}] \times \{k\} \times \{n\}) \cup (\{k\} \times NET[\{n\}] \times \{n\}) \cup (\{k\} \times \{n\} \times (NODE \setminus \{m\})) \cup (\{n\} \times \{k\} \times (NODE \setminus \{m\}))$

end

Event *FaultDecRec2* $\widehat{=}$

any

n

m

k

where

grd1 : $n \mapsto 0 \in N \land m \mapsto 1 \in N \land k \mapsto 0 \in N$

grd2 : $n \neq m \land m \neq k \land n \neq k$

grd3 : $n \mapsto k \mapsto m \in l\_net \land n \mapsto m \mapsto m \notin l\_net \land k \mapsto n \mapsto m \in l\_net \land k \mapsto m \mapsto m \notin l\_net$

grd4 : $n \mapsto k \in closure(NET)$

grd5 : $n \mapsto m \notin NET \land k \mapsto m \notin NET$

then

act1 : $l\_net := l\_net \setminus (\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\} \cup (NET[\{n\}] \times \{m\} \times \{n\}) \cup (NET[\{k\}] \times \{m\} \times \{k\}))$

end

END

An Event-B Specification of Model_ctx2
Creation Date: 28 Jan 2010 @ 07:59:34 PM

CONTEXT Model_ctx2

EXTENDS Model_ctx

CONSTANTS

`K` kind of a node: actor or sensor

AXIOMS

`axm1` : $K \in NODE \rightarrow \{0, 1\}$
0=Actor 1=Sensor

END

An Event-B Specification of Model_r2
Creation Date: 28 Jan 2010 @ 07:59:34 PM

MACHINE Model_r2
adding sensor nodes to the network

REFINES Model_r

SEES Model_ctx2

VARIABLES

`N`

`NET` actor network

`l_net`

`SNET` sensor network

`SANET` sensor actor network

INVARIANTS

`inv1` : $SNET \in dom(N) \leftrightarrow dom(N)$

`inv2` : $SANET \in dom(N) \leftrightarrow dom(N)$

`inv3` : $SNET \cap NET = \varnothing$

`inv4` : $NET \cap SANET = \varnothing$

`inv5` : $SNET \cap SANET = \varnothing$

`inv6` : $SNET = SNET^{-1}$

`inv7` : $SANET = SANET^{-1}$

`inv8` : $dom(N) \lhd id \cap SNET = \varnothing$

`inv9` : $dom(N) \lhd id \cap SANET = \varnothing$

`inv10` : $\forall n, m \cdot n \mapsto m \in SANET \Rightarrow (K(n) = 0 \land K(m) = 1) \lor (K(m) = 0 \land K(n) = 1)$

`inv11` : $\forall n, m \cdot n \mapsto m \in SNET \Rightarrow n \mapsto 0 \in N \land m \mapsto 0 \in N$

`inv12` : $\forall n, m \cdot n \mapsto m \in SANET \Rightarrow n \mapsto 0 \in N \land m \mapsto 0 \in N$

`inv13` : $NET \in (K^{-1})[\{0\}] \leftrightarrow (K^{-1})[\{0\}]$

`inv14` : $SNET \in (K^{-1})[\{1\}] \leftrightarrow (K^{-1})[\{1\}]$

`inv15` : $dom(l\_net) \in (K^{-1})[\{0\}] \leftrightarrow (K^{-1})[\{0\}]$

inv16 : $\forall n, k, x, y \cdot n \mapsto k \mapsto x \in l\_net \wedge k \mapsto n \mapsto y \in l\_net \wedge x \mapsto 1 \in K \wedge y \mapsto 1 \in K \Rightarrow x \in SANET[\{n\}] \wedge y \in SANET[\{k\}] \wedge x \mapsto y \in closure(SNET)$

inv17 : $\forall n, k, x1, x2 \cdot x1 \mapsto 1 \in K \wedge x2 \mapsto 1 \in K \wedge n \mapsto k \mapsto x1 \in l\_net \wedge n \mapsto k \mapsto x2 \in l\_net \Rightarrow x1 = x2$

inv18 : $l\_net \rhd (K^{-1})[\{0\}] \in NODE \times NODE \twoheadrightarrow (K^{-1})[\{1\}]$

inv19 : $dom(l\_net \rhd (K^{-1})[\{0\}]) = (dom(l\_net \rhd (K^{-1})[\{0\}]))^{-1}$

inv20 : $\forall n, k, x \cdot n \mapsto k \mapsto x \in l\_net \wedge x \mapsto 1 \in K \Rightarrow x \in SANET[\{n\}]$

EVENTS

Initialisation

extended

begin

act3 : $\mathtt{N} := \mathtt{NODE} \times \{1\}$
act2 : $\mathtt{NET} := \varnothing$
act1 : $\mathtt{l\_net} := \varnothing$
act4 : $SNET := \varnothing$
act5 : $SANET := \varnothing$

end

Event *AddNode* $\widehat{=}$

extends *AddNode*

any

n

where

grd1 : $\mathtt{n} \mapsto 1 \in \mathtt{N}$

then

act1 : $\mathtt{N} := \mathtt{N} \Leftarrow \{\mathtt{n} \mapsto 0\}$
a node becomes non-failed

end

Event *RemoveNode* $\widehat{=}$

refines *RemoveNode*

any

n

where

grd1 : $n \mapsto 0 \in N$
grd2 : $n \mapsto 0 \in K$

then

act1 : $N := N \Leftarrow \{n \mapsto 1\}$
a node fails
act2 : $NET := \{n\} \lhd NET \rhd \{n\}$

act3 : $l\_net := l\_net \setminus ((\{n\} \times NODE \times NODE) \cup (dom(NET) \times \{n\} \times \{n\}) \cup (NODE \times \{n\} \times K^{-1}[\{1\}]))$
immediate neighbors of a failed node delete their links with it
act4 : $SANET := \{n\} \lhd SANET \rhd \{n\}$

end

Event *AddLink* $\widehat{=}$

extends *AddLink*

any

n
m

where

grd1 : $n \mapsto 0 \in N \wedge m \mapsto 0 \in N$
nodes n,m are non-failed
grd2 : $n \mapsto m \notin NET \wedge m \mapsto n \notin NET$
grd3 : $n \neq m$
grd4 : $n \mapsto 0 \in K$
grd5 : $m \mapsto 0 \in K$

then

act1 : $NET := NET \cup \{n \mapsto m\} \cup \{m \mapsto n\}$
act2 : $l\_net := l\_net \cup \{n \mapsto m \mapsto m, m \mapsto n \mapsto n\}$

end

Event *Addl_net2hoplink* $\widehat{=}$

extends *Addl_net2hoplink*

any

n
m
k

where

grd1 : $n \mapsto 0 \in N \wedge m \mapsto 0 \in N \wedge k \mapsto 0 \in N$
grd2 : $m \mapsto k \mapsto k \in l\_net \wedge n \mapsto m \mapsto m \in l\_net \wedge n \mapsto k \mapsto m \notin l\_net \wedge k \mapsto n \mapsto m \notin l\_net$
grd3 : $m \neq n \wedge n \neq k \wedge m \neq k$
grd4 : $n \mapsto 0 \in K \wedge m \mapsto 0 \in K \wedge k \mapsto 0 \in K$

then

act1 : $l\_net := l\_net \cup \{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}$
adding 2hop neighbors of each node

end

Event *FaultDetRec* $\widehat{=}$

refines *FaultDetRec*

any

n non-failed node

m failed node
k non-failed node
x
y

where

grd1 : $n \mapsto 0 \in N \wedge m \mapsto 1 \in N \wedge k \mapsto 0 \in N$
grd2 : $n \neq m \wedge m \neq k \wedge n \neq k$
grd3 : $n \mapsto k \mapsto m \in l\_net \wedge n \mapsto m \mapsto m \notin l\_net \wedge k \mapsto n \mapsto m \in l\_net \wedge k \mapsto m \mapsto m \notin l\_net$
grd4 : $n \mapsto k \notin closure(NET)$
grd5 : $n \mapsto m \notin NET \wedge k \mapsto m \notin NET$
grd6 : $x \in SANET[\{n\}] \wedge y \in SANET[\{k\}]$
grd7 : $x \mapsto y \in closure(SNET)$
grd8 : $m \mapsto 0 \in K$
grd9 : $n \mapsto k \notin dom(l\_net \setminus \{n \mapsto k \mapsto m\})$

then

act1 : $NET := NET \cup \{n \mapsto k, k \mapsto n\}$
act2 : $l\_net := (l\_net \setminus ((\{n \mapsto k \mapsto m, k \mapsto n \mapsto m\}) \cup (NET[\{n\}] \times \{m\} \times \{n\}) \cup (NET[\{k\}] \times \{m\} \times \{k\}))) \cup (NET[\{k\}] \times \{n\} \times \{k\}) \cup (\{n\} \times NET[\{k\}] \times \{k\}) \cup (NET[\{n\}] \times \{k\} \times \{n\}) \cup (\{k\} \times NET[\{n\}] \times \{n\}) \cup \{n \mapsto k \mapsto x, k \mapsto n \mapsto y\}$

end

Event *FaultDecRec2* $\widehat{=}$

extends *FaultDecRec2*

any

n
m
k

where

grd1 : $\mathtt{n} \mapsto 0 \in \mathtt{N} \wedge \mathtt{m} \mapsto 1 \in \mathtt{N} \wedge \mathtt{k} \mapsto 0 \in \mathtt{N}$
grd2 : $\mathtt{n} \neq \mathtt{m} \wedge \mathtt{m} \neq \mathtt{k} \wedge \mathtt{n} \neq \mathtt{k}$
grd3 : $\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m} \in \mathtt{l\_net} \wedge \mathtt{n} \mapsto \mathtt{m} \mapsto \mathtt{m} \notin \mathtt{l\_net} \wedge \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m} \in \mathtt{l\_net} \wedge \mathtt{k} \mapsto \mathtt{m} \mapsto \mathtt{m} \notin \mathtt{l\_net}$
grd4 : $\mathtt{n} \mapsto \mathtt{k} \in \mathtt{closure}(\mathtt{NET})$
grd5 : $\mathtt{n} \mapsto \mathtt{m} \notin \mathtt{NET} \wedge \mathtt{k} \mapsto \mathtt{m} \notin \mathtt{NET}$
grd6 : $n \mapsto k \in dom(l\_net \setminus \{n \mapsto k \mapsto m\})$
grd7 : $m \mapsto 0 \in K$

then

act1 : $\mathtt{l\_net} := \mathtt{l\_net} \setminus (\{\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m}, \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m}\} \cup (\mathtt{NET}[\{\mathtt{n}\}] \times \{\mathtt{m}\} \times \{\mathtt{n}\}) \cup (\mathtt{NET}[\{\mathtt{k}\}] \times \{\mathtt{m}\} \times \{\mathtt{k}\}))$

end

Event *AddSLink* $\widehat{=}$

any

n

m

where

grd1 : $n \mapsto 0 \in N \wedge m \mapsto 0 \in N$

grd2 : $n \notin dom(NET) \wedge m \notin dom(NET)$

grd3 : $n \mapsto m \notin SNET$

grd4 : $n \neq m$

grd5 : $n \mapsto 1 \in K \wedge m \mapsto 1 \in K$

then

act1 : $SNET := SNET \cup \{n \mapsto m, m \mapsto n\}$

end

Event *AddSAlink* $\widehat{=}$

any

n

m

where

grd1 : $n \mapsto 0 \in N \wedge m \mapsto 0 \in N$

grd2 : $(K(n) = 0 \wedge K(m) = 1) \vee (K(n) = 1 \wedge K(m) = 0)$

grd3 : $n \mapsto m \notin SANET$

grd4 : $n \neq m$

then

act1 : $SANET := SANET \cup \{n \mapsto m, m \mapsto n\}$

end

END

An Event-B Specification of Model_r3
Creation Date: 28 Jan 2010 @ 08:06:48 PM

MACHINE Model_r3

REFINES Model_r2

SEES Model_ctx2

VARIABLES

`N`

`NET` actor network

`l_net`

`SNET` sensor network

`SANET` sensor actor network

`flag` to create degree set

failedNodeNeigh

degree

locX

locY

INVARIANTS

inv17 : $flag \in BOOL$

inv19 : $failedNodeNeigh \subseteq dom(N)$

inv20 : $degree \in dom(N) \nrightarrow 0\,..\,card(dom(N))$

inv21 : $locX \in dom(N) \rightarrow 0\,..\,1000$

inv22 : $locY \in dom(N) \rightarrow 0\,..\,1000$

inv23 : $failedNodeNeigh \cap dom(degree) = \varnothing$

inv24 : $flag = TRUE \Rightarrow degree = \varnothing$

EVENTS

Initialisation

extended

begin

act3 : $\mathtt{N} := \mathtt{NODE} \times \{\mathtt{1}\}$

act2 : $\mathtt{NET} := \varnothing$

act1 : $\mathtt{l\_net} := \varnothing$

act4 : $\mathtt{SNET} := \varnothing$

act5 : $\mathtt{SANET} := \varnothing$

act6 : $flag := TRUE$

act8 : $failedNodeNeigh := \varnothing$

act9 : $degree := \varnothing$

act21 : $locX := NODE \times \{0\}$

act22 : $locY := NODE \times \{0\}$

end

Event *AddNode* $\widehat{=}$

extends *AddNode*

any

n

i

j

where

grd1 : $\mathtt{n} \mapsto \mathtt{1} \in \mathtt{N}$

grd2 : $flag = TRUE$

grd3 : $i \in 1\,..\,1000$

grd4 : $j \in 1\,..\,1000$

then

act1 : $N := N \Leftarrow \{n \mapsto 0\}$
a node becomes non-failed
act2 : $locX := locX \Leftarrow \{n \mapsto i\}$
act3 : $locY := locY \Leftarrow \{n \mapsto j\}$
end

Event *RemoveNode* $\widehat{=}$

extends *RemoveNode*

any
n
where
grd1 : $n \mapsto 0 \in N$
grd2 : $n \mapsto 0 \in K$
grd3 : $flag = TRUE$
then
act1 : $N := N \Leftarrow \{n \mapsto 1\}$
a node fails
act2 : $NET := \{n\} \lhd NET \rhd \{n\}$
act3 : $l\_net := l\_net \setminus ((\{n\} \times NODE \times NODE) \cup (dom(NET) \times \{n\} \times \{n\}) \cup (NODE \times \{n\} \times K^{-1}[\{1\}]))$
immediate neighbors of a failed node delete their links with it
act4 : $SANET := \{n\} \lhd SANET \rhd \{n\}$
act5 : $flag := FALSE$
act6 : $failedNodeNeigh := dom(l\_net^{-1}[\{n\}])$
end

Event *AddLink* $\widehat{=}$

extends *AddLink*

any
n
m
where
grd1 : $n \mapsto 0 \in N \wedge m \mapsto 0 \in N$
nodes n,m are non-failed
grd2 : $n \mapsto m \notin NET \wedge m \mapsto n \notin NET$
grd3 : $n \neq m$
grd4 : $n \mapsto 0 \in K$
grd5 : $m \mapsto 0 \in K$
grd6 : $flag = TRUE$
then
act1 : $NET := NET \cup \{n \mapsto m\} \cup \{m \mapsto n\}$
act2 : $l\_net := l\_net \cup \{n \mapsto m \mapsto m, m \mapsto n \mapsto n\}$
end

Event *Addl_net2hoplink* $\widehat{=}$

extends *Addl_net2hoplink*

any

n

m

k

where

grd1 : $\mathtt{n} \mapsto 0 \in \mathtt{N} \land \mathtt{m} \mapsto 0 \in \mathtt{N} \land \mathtt{k} \mapsto 0 \in \mathtt{N}$

grd2 : $\mathtt{m} \mapsto \mathtt{k} \mapsto \mathtt{k} \in \mathtt{l\_net} \land \mathtt{n} \mapsto \mathtt{m} \mapsto \mathtt{m} \in \mathtt{l\_net} \land \mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m} \notin \mathtt{l\_net} \land \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m} \notin \mathtt{l\_net}$

grd3 : $\mathtt{m} \neq \mathtt{n} \land \mathtt{n} \neq \mathtt{k} \land \mathtt{m} \neq \mathtt{k}$

grd4 : $\mathtt{n} \mapsto 0 \in \mathtt{K} \land \mathtt{m} \mapsto 0 \in \mathtt{K} \land \mathtt{k} \mapsto 0 \in \mathtt{K}$

grd5 : $\mathit{flag} = \mathit{TRUE}$

then

act1 : $\mathtt{l\_net} := \mathtt{l\_net} \cup \{\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m}, \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m}\}$

adding 2hop neighbors of each node

end

Event *FaultDetRec* $\widehat{=}$

extends *FaultDetRec*

any

n on node

m failed node

k on node

x

y

where

grd1 : $\mathtt{n} \mapsto 0 \in \mathtt{N} \land \mathtt{m} \mapsto 1 \in \mathtt{N} \land \mathtt{k} \mapsto 0 \in \mathtt{N}$

grd2 : $\mathtt{n} \neq \mathtt{m} \land \mathtt{m} \neq \mathtt{k} \land \mathtt{n} \neq \mathtt{k}$

grd3 : $\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m} \in \mathtt{l\_net} \land \mathtt{n} \mapsto \mathtt{m} \mapsto \mathtt{m} \notin \mathtt{l\_net} \land \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m} \in \mathtt{l\_net} \land \mathtt{k} \mapsto \mathtt{m} \mapsto \mathtt{m} \notin \mathtt{l\_net}$

grd4 : $\mathtt{n} \mapsto \mathtt{k} \notin \mathtt{closure}(\mathtt{NET})$

grd5 : $\mathtt{n} \mapsto \mathtt{m} \notin \mathtt{NET} \land \mathtt{k} \mapsto \mathtt{m} \notin \mathtt{NET}$

grd6 : $\mathtt{x} \in \mathtt{SANET}[\{\mathtt{n}\}] \land \mathtt{y} \in \mathtt{SANET}[\{\mathtt{k}\}]$

grd7 : $\mathtt{x} \mapsto \mathtt{y} \in \mathtt{closure}(\mathtt{SNET})$

grd8 : $\mathtt{m} \mapsto 0 \in \mathtt{K}$

grd9 : $\mathtt{n} \mapsto \mathtt{k} \notin \mathtt{dom}(\mathtt{l\_net} \setminus \{\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m}\})$

grd10 : $\mathit{flag} = \mathit{FALSE}$

grd11 : $\mathit{failedNodeNeigh} = \varnothing$

grd12 : $n \in \mathit{dom}(\mathit{degree}) \land k \in \mathit{dom}(\mathit{degree})$

grd13 : $\mathit{degree}(n) > \mathit{min}(\mathit{dom}(\mathit{degree}^{-1}))$

grd14 : $n \mapsto k \notin \mathit{dom}(l\_net \setminus \{n \mapsto k \mapsto m\})$

grd15 : $\forall i \cdot i \in dom(\{n,k\} \lhd degree) \Rightarrow (locX(n) - locX(k)) * (locX(n)-locX(k))+(locY(n)-locY(k))*(locY(n)-locY(k)) < (locX(n)-locX(i))*(locX(n)-locX(i))+(locY(n)-locY(i))* (locY(n) - locY(i))$
shortest distance
grd16 : $degree(k) > degree(n) \Rightarrow (\exists i \cdot i \in dom(\{n,k\} \lhd degree) \wedge (locX(k)-locX(i))*(locX(k)-locX(i))+(locY(k)-locY(i))* (locY(k)-locY(i)) < (locX(k)-locX(n))*(locX(k)-locX(n))+ (locY(k) - locY(n)) * (locY(k) - locY(n)))$

then

act1 : $\mathtt{NET} := \mathtt{NET} \cup \{\mathtt{n} \mapsto \mathtt{k}, \mathtt{k} \mapsto \mathtt{n}\}$
act2 : $\mathtt{l\_net} := (\mathtt{l\_net} \setminus ((\{\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m}, \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m}\}) \cup (\mathtt{NET}[\{\mathtt{n}\}] \times \{\mathtt{m}\} \times \{\mathtt{n}\}) \cup (\mathtt{NET}[\{\mathtt{k}\}] \times \{\mathtt{m}\} \times \{\mathtt{k}\}))) \cup (\mathtt{NET}[\{\mathtt{k}\}] \times \{\mathtt{n}\} \times \{\mathtt{k}\}) \cup (\{\mathtt{n}\} \times \mathtt{NET}[\{\mathtt{k}\}] \times \{\mathtt{k}\}) \cup (\mathtt{NET}[\{\mathtt{n}\}] \times \{\mathtt{k}\} \times \{\mathtt{n}\}) \cup (\{\mathtt{k}\} \times \mathtt{NET}[\{\mathtt{n}\}] \times \{\mathtt{n}\}) \cup \{\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{x}, \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{y}\}$
act3 : $degree := \{n\} \lhd degree$
@act4 flag:= TRUE

end

Event *FaultDecRec2* $\widehat{=}$

extends *FaultDecRec2*

any

n
m
k

where

grd1 : $\mathtt{n} \mapsto 0 \in \mathtt{N} \wedge \mathtt{m} \mapsto 1 \in \mathtt{N} \wedge \mathtt{k} \mapsto 0 \in \mathtt{N}$
grd2 : $\mathtt{n} \neq \mathtt{m} \wedge \mathtt{m} \neq \mathtt{k} \wedge \mathtt{n} \neq \mathtt{k}$
grd3 : $\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m} \in \mathtt{l\_net} \wedge \mathtt{n} \mapsto \mathtt{m} \mapsto \mathtt{m} \notin \mathtt{l\_net} \wedge \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m} \in \mathtt{l\_net} \wedge \mathtt{k} \mapsto \mathtt{m} \mapsto \mathtt{m} \notin \mathtt{l\_net}$
grd4 : $\mathtt{n} \mapsto \mathtt{k} \in \mathtt{closure}(\mathtt{NET})$
grd5 : $\mathtt{n} \mapsto \mathtt{m} \notin \mathtt{NET} \wedge \mathtt{k} \mapsto \mathtt{m} \notin \mathtt{NET}$
grd6 : $\mathtt{n} \mapsto \mathtt{k} \in \mathtt{dom}(\mathtt{l\_net} \setminus \{\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m}\})$
grd7 : $\mathtt{m} \mapsto 0 \in \mathtt{K}$
grd8 : $flag = FALSE$
grd9 : $failedNodeNeigh = \varnothing$
grd10 : $n \in dom(degree) \wedge k \in dom(degree)$
grd11 : $n \mapsto k \in dom(l\_net \setminus \{n \mapsto k \mapsto m\})$

then

act1 : $\mathtt{l\_net} := \mathtt{l\_net} \setminus (\{\mathtt{n} \mapsto \mathtt{k} \mapsto \mathtt{m}, \mathtt{k} \mapsto \mathtt{n} \mapsto \mathtt{m}\} \cup (\mathtt{NET}[\{\mathtt{n}\}] \times \{\mathtt{m}\} \times \{\mathtt{n}\}) \cup (\mathtt{NET}[\{\mathtt{k}\}] \times \{\mathtt{m}\} \times \{\mathtt{k}\}))$
act2 : $degree := \{n\} \lhd degree$

end

Event *flag* $\widehat{=}$

when

grd1 : $flag = FALSE$

grd2 : $card(dom(degree)) = 1$

grd3 : $failedNodeNeigh = \varnothing$

then

act1 : $flag := TRUE$

act2 : $degree := \varnothing$

end

Event *AddSLink* $\widehat{=}$

extends *AddSLink*

any

n

m

where

grd1 : $\mathtt{n} \mapsto \mathtt{0} \in \mathtt{N} \wedge \mathtt{m} \mapsto \mathtt{0} \in \mathtt{N}$

grd2 : $\mathtt{n} \notin \mathtt{dom}(\mathtt{NET}) \wedge \mathtt{m} \notin \mathtt{dom}(\mathtt{NET})$

grd3 : $\mathtt{n} \mapsto \mathtt{m} \notin \mathtt{SNET}$

grd4 : $\mathtt{n} \neq \mathtt{m}$

grd5 : $\mathtt{n} \mapsto \mathtt{1} \in \mathtt{K} \wedge \mathtt{m} \mapsto \mathtt{1} \in \mathtt{K}$

grd6 : $flag = TRUE$

then

act1 : $\mathtt{SNET} := \mathtt{SNET} \cup \{\mathtt{n} \mapsto \mathtt{m}, \mathtt{m} \mapsto \mathtt{n}\}$

end

Event *AddSALink* $\widehat{=}$

extends *AddSALink*

any

n

m

where

grd1 : $\mathtt{n} \mapsto \mathtt{0} \in \mathtt{N} \wedge \mathtt{m} \mapsto \mathtt{0} \in \mathtt{N}$

grd2 : $(K(n) = 0 \wedge K(m) = 1) \vee (K(n) = 1 \wedge K(m) = 0)$

grd3 : $\mathtt{n} \mapsto \mathtt{m} \notin \mathtt{SANET}$

grd4 : $\mathtt{n} \neq \mathtt{m}$

then

act1 : $\mathtt{SANET} := \mathtt{SANET} \cup \{\mathtt{n} \mapsto \mathtt{m}, \mathtt{m} \mapsto \mathtt{n}\}$

end

Event *Degree* $\widehat{=}$

any

n

where

grd1 : $flag = FALSE$
grd3 : $failedNodeNeigh \neq \varnothing$
grd4 : $n \in failedNodeNeigh$

then

act1 : $degree := degree \cup \{n \mapsto card(NET[\{n\}])\}$
act2 : $failedNodeNeigh := failedNodeNeigh \setminus \{n\}$

end

END

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku

- Department of Information Technology
- Department of Mathematics

Åbo Akademi University

- Department of Information Technologies

Turku School of Economics

- Institute of Information Systems Sciences

ISBN 978-952-12-2411-9
ISSN 1239-1891