Integrating Proved State-Based Models for Constructing Correct Distributed Algorithms

Manamiary Bruno Andriamiarina[1], Dominique Méry[1], and Neeraj Kumar Singh[2]

Université de Lorraine, LORIA, BP 239, 54506 Vandœuvre-lès-Nancy, France
{Manamiary.Andriamiarina,Dominique.Mery}@loria.fr
Department of Computer Science, University of York, United Kingdom
neeraj.singh@cs.york.ac.uk, Neerajkumar.Singh@loria.fr

Abstract. The verification of distributed algorithms is a challenge for formal techniques supported by tools, such as model checkers and proof assistants. The difficulties lie in the derivation of proofs of required properties, such as safety and eventuality, for distributed algorithms. In this paper, we present a methodology based on the general concept of refinement that is used for developing distributed algorithms satisfying a given list of safety and liveness properties. The methodology is a recipe for reusing the old ingredients of the classical temporal approaches, which are illustrated through standard example of routing protocols. More precisely, we show how the state-based models can be developed for specific problems and how they can be simply reused by controlling the composition of state-based models through the refinement relationship. The *service-as-event* paradigm is introduced for helping users to describe algorithms as a composition of simple services and/or to decompose them into simple steps. Consequently, we obtain a framework to derive new distributed algorithms by developing existing distributed algorithms using *correct-by-construction* approach. The *correct-by-construction* approach ensures the correctness of developed distributed algorithms.

Keywords: Distributed algorithms, state-based models, composition, correct-by-construction, Event-B, liveness, eventuality.

1 Introduction

The formal modelling of distributed algorithms constitutes a challenge for methods and tools: these algorithms can be used to evaluate strengths and weaknesses of formal techniques supported by tools, such as model-checkers [12] and proof assistants [23, 27]. Formal techniques address properties, like safety, liveness and fairness. However, formal design and study of distributed algorithms also introduce other constraints to take into account, including time aspects [25], probabilistic features [17], fault-tolerance, scalability, dependability, etc. The *correct-by-construction* paradigm [15] offers an alternative and a promising approach to prove and derive *correct* distributed algorithms using a progressive and validated methodological approach [7]. More precisely, *refinement* is a key concept for organizing or structuring the (re-)development and (re-)discovery of distributed algorithms [2, 20] by reusing or replaying the former developments.

E.B. Johnsen and L. Petre (Eds.): IFM 2013, LNCS 7940, pp. 268–284, 2013.

In this paper, we present a way to organise incremental refinement-based designs of distributed algorithms. Our methodology is based on structures coping with the modelling of distributed algorithms and providing a semantical framework for expressing both safety and liveness properties. We provide a list of recipes for reusing the old ingredients of the classical temporal approaches, by integrating refinement. Refinement-based development necessitates guidelines for helping the user to develop systems; these guidelines have to be able to incorporate refinement and make the composition of different interacting systems as simple as possible. When dealing with composition of interacting systems, there are more elements to prove, since we should demonstrate that the interacting systems are without interference [21]. We propose to minimize the complexity of proofs and formalisation, by *reusing* previous developments and proofs, and by *organizing* them. We introduce here a *component-driven* development of an algorithm: the *service-as-event* paradigm. A *component* actually represents a *phase* or a *step* of the algorithm: for instance, there are *initialisation*, *requesting critical sections* or *stabilisation* phases. It should be noted that we work on distributed algorithms exhibiting the following properties: the algorithms can be divided into components that describe phases local to nodes; and these phases are coordinated and synchronized according to the local states of the nodes. The main goal is to reuse as much as possible previous proofs of former refinement-based developments to model the phases.

The current approach extends the methodology described in [19] for developing *sequential* programs and *combining* phases, that we have experienced on algorithmic classical case studies related to the distributed protocols. We have noticed that *graphical (sequence)* diagrams can be used for expressing possible scenarios or phases of the protocols/algorithms, as defined by Tanembaum in [26]. The initial objective was to integrate such *graphical* notations, for *co-proving* a sequential program characterised by a *pre/post* specification. In a large number of cases, by using these diagrams and applying the *correct-by-construction* paradigm, we are able to derive algorithmic solutions annotated with invariants that can be checked and verified. However, these diagrams are not abstract enough to express properties on traces and fairness, and are also difficult to refine, while preserving properties such as fairness or liveness. Therefore, we limit the usage of these diagrams to the identification of algorithmic phases. The phases of an algorithm are defined by an *initial* (PRE) and a *final* (POST) states; and these phases are linked *sequentially*, by *temporal* operators, like *leads to* ($\rightsquigarrow$). The purpose of our work is to *link* and *coordinate* phases to obtain targeted distributed algorithms, by *integrating* and *composing* formal models, using *refinement diagrams* and the *service-as-event* paradigm.

Our paper is organised as follows. Section 2 introduces the modelling framework. Section 3 depicts the temporal framework for refinement-based development, more precisely state properties and refinement diagrams. Section 4 discusses structures for refinement-based development: the temporal coordination and decomposition of models, using the *service-as-event* paradigm. Section 5 illustrates our methodology with the study of the protocol ANYCAST RP. Finally, Section 6 concludes this paper along with the future work.

2 Choice of a State-Based Modelling Language

We choose EVENT B [1] as a state-based modelling language, mainly because of the *effective 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* incrementally and progressively by *refining* the abstraction. Event-B is also supported by a complete toolset RODIN [24] providing features like refinement, proof obligations generation, proof assistants and model-checking facilities.

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; **(2)** Machines express dynamic informations about the model, safety properties, and events. An EVENT B model is defined *either* as a context or as a machine. A machine organises events (or actions) modifying state variables and uses static informations defined in a context. These basic structures are extended by the *refinement of models* which *relates* an *abstract* model and a *concrete* model.

Modelling Actions Over States. 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 users to develop EVENT B models 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.1) , which consider refinements of events and machines.

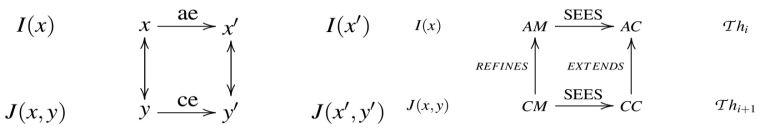


Fig. 1. Machines and Contexts relationships

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 [15]. 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.1) 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:

$$I(x) \wedge J(x,y) \wedge BA(ce)(y,y') \Rightarrow \exists x' \cdot (BA(ae)(x,x') \wedge J(x',y'))$$

We have briefly introduced the EVENT B modelling language and the structures proposed for organising the development of state-based models. In fact, the refinement-based development of EVENT B requires a very careful derivation process, integrating possible *tough* interactive proofs for discharging generated proof obligations, at each step of development.

3 State Properties and Refinement Diagrams

This section extends semantically EVENT B and introduces a way to deal with liveness properties using especially the refinement diagrams and the *leads to* ($\leadsto$) operator. Refinement diagrams have been introduced in a previous work [19], in order to help to develop sequential programs using refinement. The notation using the *leads to* operator $A \leadsto B$ is defined by the temporal assertion "$\Box(A \Rightarrow \Diamond B)$". This formula means that every A will *eventually* be followed by B.

Extending the Scope of EVENT B Properties. EVENT B allows users to express safety properties on models considered as reactive systems. An EVENT B model is valid with respect to a set of discharged proof obligations. However, since we have a list of events for each model, we can simulate reactions to events by extending the semantical scope of EVENT B properties. This extension of the properties taken into account by EVENT B to liveness ones, requires the definition of traces for an EVENT B model in an operational style. Therefore, we propose the use of the TLA [14] framework to support our proofs, as the framework provides simple temporal modalities, such as liveness and fairness.

We first define the temporal framework of an EVENT B machine M, using the following TLA notations: *Init* is the predicate specifying initial states; $\Box[Next]_y$ means that each pair of consecutive states either satisfies *Next* or leaves the values of y unchanged; $\mathrm{WF}_y(Next)$ expresses a *weak fairness* condition over *Next*.

Definition 1. *Let M be an* EVENT B *machine and C a context seen by M. Let y be the list of variables of M, let E be the set of events of M, and let $Init(y)$ be the predicate defining the initial values of y in M. The temporal framework of M is defined by the TLA specification $\mathcal{S}pec(M)$: $Init(y) \wedge \Box[Next]_y \wedge \mathrm{WF}_y(Next)$, where $Next \equiv \exists e \in E.BA(e)(y,y')$.*

Following Lamport [14], $\mathcal{S}pec(M)$ is valid for the set of infinite traces simulating M, with respect to the events of M and to fairness constraints. The set of traces for M is a subset of $Values^{\omega}$, which is the set of infinite words over the set of possible values of y in M, namely *Values*.

Liveness properties for M are, de facto, defined in TLA as follows. M satisfies $P \rightsquigarrow Q$ when $\Gamma(M) \vdash \mathcal{S}pec(M) \Rightarrow (P \rightsquigarrow Q)$. $\Gamma(M)$ is the proof context of M. Obviously, safety properties can be reformulated in the same framework. As for liveness properties, we can also use the *wp*-based approach for defining these properties under weak fairness. We can apply as well the works of Abrial et al [9, 18] on mathematical semantics in a *wp* framework, and on specific constructs [4] to state liveness properties as events.

Refinement Diagrams and *Leads To* ($\rightsquigarrow$) Operator. *Refinement diagrams* are used to develop the machine M and to add control in the EVENT B models. These diagrams are close to predicate diagrams [8] and to proof lattices introduced by Owicki and Lamport in [22] for representing (proofs of) liveness properties under fairness assumptions. We do not use these diagrams for proving but for supporting refinement. We construct the refinement lattices by applying the inference rules for the temporal operator *leads to* ($\rightsquigarrow$).

Definition 2. *Let M be an* EVENT B *machine and C a context seen by M. A is a set of assertions; $I(M)$ is the invariant of M; c are (control) variables of M, with values identifying the control points of M (e.g. start, end, etc.); G is a finite set of assertions for M called conditions of the form $g(x)$, where x are variables of M. Let E be the set of events for M.*

A refinement diagram for M, over A, is a labeled directed graph over A, with labels from G or E, satisfying the following rules:

- *If R is related to S by a unique arrow labeled $e \in E$, then*
  - *It satisfies the property $R \rightsquigarrow S$*
  - $\forall c,x,c',x'.R(c,x) \wedge I(M)(c,x) \wedge BA(e)(c,x,c',x') \Rightarrow S(c',x')$
  - $\forall c,x.R(c,x) \wedge I(M)(c,x) \Rightarrow \exists c',x'.BA(e)(c,x,c',x')$
- *If R is related to $S_1, \ldots, S_p$, then*
  - *Each arrow R to S_i is labeled by a guard $g_i \in G$.*
  - *For any i in $1..p$ the following conditions hold.*
    $\begin{cases} R \wedge I(M) \wedge g_i(x) \Rightarrow S_i \\ \forall j.j \in 1..p \wedge j \neq i \wedge R \wedge I(M) \wedge g_i(x) \Rightarrow \neg g_j(x) \end{cases}$
  - $R \wedge I(M) \Rightarrow \exists i \in 1..p.g_i$.

– *For each $e \in E$, there is only one instance of e in the diagram.*

A refinement diagram D for M, over A, is denoted by $PD(M) = (A, M, G, E)$.

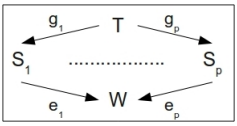


Fig. 2. A *refinement diagram*

A refinement diagram, as illustrated by the figure 2, relates a pair of assertions (T, W). We assume that T is a precondition, that can be decomposed into p assertions $S_1, \cdots, S_p$, and W is a postcondition.

Refinement diagrams can be used to infer the total correctness of an algorithm *constructed* step-by-step using refinement. The operator *leads to* ($\rightsquigarrow$) is transitive and confluent. Therefore, if a *refinement diagram* is built for a given problem, it is sound with respect to the requirements of the problem. *Refinement diagrams* possess proved properties [19], that we enumerate here.

Property 1. Let M be a machine and $D = (A, M, G, E)$ be a refinement diagram for M.

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, and 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 properties are derived from TLA definitions [14]. *Refinement diagrams* are a generalised version of diagrams proposed for developing sequential algorithms [19] and these are based on the *call-as-event* paradigm. Moreover, *refinement diagrams* are attached to EVENT B models and can be used for deriving liveness properties. The justification of such diagrams is based on the analysis of *leads to* properties and on liveness properties. The proof system of TLA contains proof rules for deriving the correctness of those properties. In the next section, we detail a paradigm for aiding the proof-based development of distributed algorithms.

4 *Service-As-Event* Paradigm

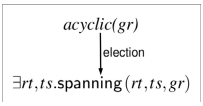
The EVENT B methodology requires skills in understanding the notion of *refinement*. Expertise is also required in the use of proof assistants and management of the modelling process, in order to ensure the discharging of proofs. In the EVENT B modelling method, the most important step is the expression of a very abstract definition of the problem to solve. The first abstract model usually gives a list of events corresponding to the *pre/post* specification with respect to the different cases. Each refinement step details progressively the abstract specifications (e.g. by decomposing them into *phases*,

in the case of algorithmic systems, etc.). Each new step is checked by discharging proof obligations. Hence, the objective is clearly to simplify the effort of proof and explore simple ways to express a problem as a combination of (possibly reusable) *components*. We are interested in distributed algorithms; therefore, a *component* is equivalent to a *phase/step* of an algorithm. *Components* can be viewed as *"sub"*-distributed algorithms composing the actual algorithm.

In this paper, we present the *service-as-event* paradigm, inspired by the *call-as-event* [19] paradigm and based on *refinement diagrams*. The *service-as-event* paradigm helps us to state problems, using liveness properties, as for instance $P \rightsquigarrow Q$. An event e models the *effective* service leading from P to Q.

Primary Usage: Service Description. The *service-as-event* paradigm can help to state a problem in an abstract manner. The abstraction of a problem in EVENT B is as follows: An abstract event e expresses a *pre/post* specification. The *pre-condition* P is stated by the guard of the event e, whereas the *post-condition* Q is defined by the action of e. Using the properties of *refinement diagrams*, we can depict this statement with the property: $(P \xrightarrow{e} Q) \Rightarrow (P \rightsquigarrow Q)$. The event e expresses, in an abstract way, the service linking P and Q: every time P holds, e will be triggered and consequently, P will (*eventually*) be followed by Q.

Example 1. For instance, the leader election problem [2] is expressed using the following property: $acyclic(gr) \rightsquigarrow \exists rt, ts.\mathsf{spanning}(rt, ts, gr)$, where $acyclic(gr)$ states that gr is an acyclic connected graph and $spanning(rt, tr, gr)$ states that tr is a directed spanning tree of gr and its root rt is the leader. The property is illustrated by the *refinement diagram* 3 and simply stated in EVENT B , as follows:



EVENT election $\widehat{=}$
WHEN
$acyclic(gr)$
THEN
$rt, ts : |\mathsf{spanning}(rt', ts', gr)$
END

Fig. 3. A *refinement diagram* for leader election

The refinement diagram Fig.3 expresses that a process *election* is characterized by an abstract event election stating *what* is computed, but *not how* it is computed. The computation process is depicted in the refinement model, which will be defined later.

Extended Usage: Phase Identification from Service Decomposition. Another way to use the *service-as-event* paradigm is to decompose liveness properties, using the inference rules of the *leads to* ($\rightsquigarrow$) operator, such as the transitivity rule. In fact, we use the rules related to $\rightsquigarrow$ to break up a global service into multiple and simpler *"sub"*-services, analogous to steps or phases. As an illustration, one can decompose an EVENT B specification of a problem, represented by the property $P \rightsquigarrow Q$, as follows:

$$\dfrac{\dfrac{\dfrac{\cdots}{P \rightsquigarrow S}\,(\text{trans}) \quad \dfrac{\cdots}{S \rightsquigarrow R}\,(\text{trans})}{P \rightsquigarrow R}\,(\text{trans}) \quad \dfrac{\dfrac{\cdots}{R \rightsquigarrow X}\,(\text{trans}) \quad \dfrac{\cdots}{X \rightsquigarrow Q}\,(\text{trans})}{R \rightsquigarrow Q}\,(\text{trans})}{P \rightsquigarrow Q}\,(\text{trans})$$

The initial property $P \rightsquigarrow Q$ is separated into several simpler properties (representing phases of the algorithm), until a satisfactory decomposition into independent phases,

linked by *services* is obtained. Therefore, allowing the phases to be developed separately.

This process is similar to refinement, as shown by the following figure (see Fig.4): The first property $P \rightsquigarrow Q$ is associated with an abstract model, describing the service offered by the algorithm with an abstract event; the use of the transitivity rule to simplify $P \rightsquigarrow Q$ is interpreted as identifying the various steps from P to Q: it corresponds to the fact that new events, modelling intermediate transitions between P and Q, are added to the model, using refinement. A level in the proof tree is associated with a level of refinement. One may continue to decompose services, until each phase of the concrete algorithm is matched with a property.

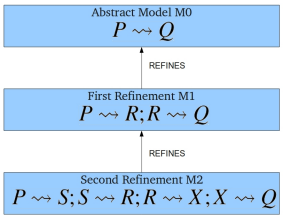


Fig. 4. Refinement and Decomposition

Example 2. For instance, a routing algorithm can be decomposed into two phases: **(1)** a route discovery step, **(2)** a route maintenance/reconstruction, if the route is broken. Another point is to decompose the routing process into steps which are simpler, safer and more stable. In the next section, we give an example in which three phases for a routing algorithm are identified, to ensure that the routing service is satisfied.

First, the development methodology consists in decomposing a complex algorithm into simple fragments (services) using the *service-as-event* paradigm and *refinement diagrams*. The following step is to detail the developed services/phases and coordinate them by adding control. Hence, we can guide our refinement-base process by using refinement diagrams related to the EVENT B models.

5 Case Study: ANYCAST RP

We present in this section an example illustrating our modelling methodology (*refinement diagrams* and *service-as-event* paradigm), with the ANYCAST RP routing protocol [10, 11]. However, due to space requirements, we do not provide the whole development[1], we only give relevant details allowing us to explain clearly the methodology and the integration of models.

Introduction. ANYCAST RP is a protocol for multicast (one-to-many) communications (see Fig.5). In this protocol, a set of routers, called Designated Routers (*DR*), are used by directly connected sources to transmit data (*msg*) to another set of distant routers, the

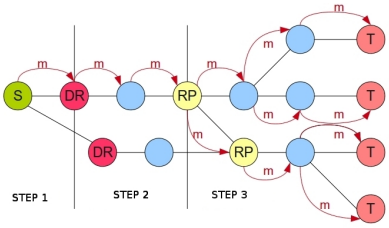
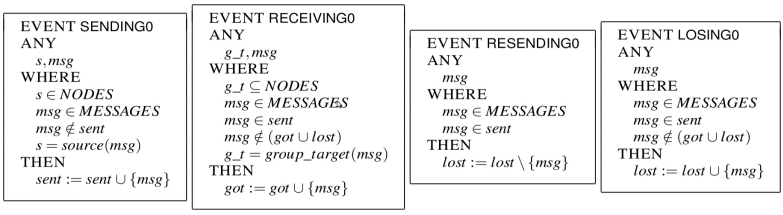


Fig. 5. ANYCAST RP

[1] Available at: `http://www.loria.fr/~andriami/ifm/index.html`

Rendezvous Points (*RP*). These Rendezvous Points (*RP*) are in charge of load sharing, redundancy and message delivery to connected destinations. ANYCAST RP is a non-toy protocol recommended by Cisco Systems, Inc as a reliable solution for multicasting [10]. Moreover, the protocol is cited as robust, scalable, having satisfactory bandwidth efficiency and good QoS [10, 13]. We use this protocol as an illustration, because it can be divided easily into *independent and sequential phases/steps*: the routing of messages (*msg*) **(1)** from sources (*s*) to Designated Routers (*DR*), **(2)** from Designated Routers (*DR*) to Rendezvous Points (*RP*) and **(3)** from Rendezvous Points (*RP*) to connected destinations. The following sections demonstrate the formal modelling of the ANYCAST RP protocol.

Abstract Model. We start with an abstract model ANYCAST_M0, describing the service offered by the protocol: that is, the routing and delivery of a message (*msg*), from a source (*s*) to a set of destinations (*g_t*). Sets of messages (*MESSAGES*), nodes (*NODES*), sources and destinations of each message (*m*) (indicated by functions *source* and *group_target*) are defined. Variables are also defined: *sent* contains messages sent by sources, *got* depicts messages received by destinations, *lost* contains lost messages. These variables are initialised with an empty set ($\varnothing$). Simple invariants constrain these variables: $got \cup lost \subseteq sent$; $got \cap lost = \varnothing$. Events define the behaviour of the system:



- We have events related to the protocol:
  - SENDING0 models the sending of a message (*msg*) by a source (*s*).
  - RECEIVING0 demonstrates the receiving of a message (*msg*) by a group of destinations (*g_t*): the message (*msg*) has been sent by a source, has not yet been lost nor received, therefore all the destinations (*g_t*) can receive the message (*msg*).
  - RESENDING0 depicts the re-sending of a message (*msg*): if the message (*msg*) has been lost, it is recovered.
- And events related to environment: LOSING0 presents the loss of a sent but not yet received message (*msg*).

This model is associated to the following *refinement diagram*:

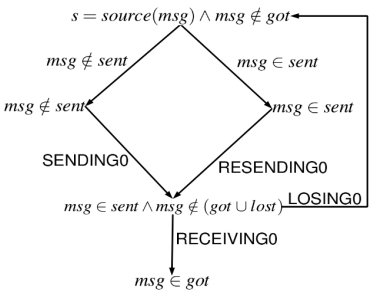


Fig. 6. Diagram $D0$ for ANYCAST_M0

The diagram $D0$ (see Fig.6) gives us the possibility to express the goal of ANYCAST RP as follows: $(s = source(msg) \wedge msg \notin got) \rightsquigarrow (msg \in got)$. The routing *service* allows a non-received message (msg), whose source is (s), to be eventually received by all of its destinations. This routing *service* can be decomposed into *sub-services*: a *sending* one (SENDING0), a *re-sending* one (RESENDING0) and a *receiving* one (RECEIVING0).

The service RECEIVING0 can be considered as the *main* service that allows users to verify the property $P \rightsquigarrow Q$ (with $P \mathrel{\widehat{=}} (s = source(msg) \wedge msg \notin got)$ and $Q \mathrel{\widehat{=}} (msg \in got)$), because it actually models the receiving of a message (msg) by the destinations ($got := got \cup \{msg\}$). The diagram $D0$ and the property $P \rightsquigarrow Q$ describe the normal behaviour of the algorithm, without errors. However, we also consider message losing in $P \rightsquigarrow F$, with $F \mathrel{\widehat{=}} (msg \in lost)$, because messages can be lost (event LOSING0). But, since lost messages are sent again to the destinations (RESENDING0), and since we assume that the messages are not stuck in the *lost* state forever (to ensure progress of the algorithm), we have $F \rightsquigarrow Q$. Therefore, $P \rightsquigarrow Q$ is verified.

First Refinement. This refinement[2] (ANYCAST_M1) adds the Designated Routers (DR) between the sources (s) and the destinations (g_t). New variables are defined: dr_rcvd contains the messages received by some selected (not yet identified at this level of *abstraction*) Designated Routers (DR) from sources and dr_sent depicts the messages sent by selected Designated Routers (DR) to destinations; simple invariants are given: **(1)** $dr\_rcvd \subseteq sent$, **(2)** $dr\_sent \subseteq dr\_rcvd$, **(3)** $got \cup lost \subseteq dr\_sent$. Previous events are refined and new ones are added: DR_RECEIVING1 models the receiving of a message (msg) by a selected Designated Router (dr), from a source (s); DR_SENDING1 demonstrates the transmission of a message (msg) by a selected Designated Router (dr), to the destinations; DR_RESENDING1 is a refinement of RESENDING0. In fact, the sources are not in charge of the re-sending procedure, but the Designated Routers; RECEIVING1 presents the receiving of a message (msg) by destinations, from a Designated Router (dr); LOSING1 models losses of message (msg) between only Designated Routers and destinations, since losses between sources and Designated Routers are highly improbable. Let us denote by X the events DR_RESENDING1, RECEIVING1, LOSING1; and by Y their corresponding abstract versions: RESENDING0, RECEIVING0, LOSING0.

EVENT DR_RECEIVING1	EVENT DR_SENDING1	EVENT X REFINES Y
WHEN	WHEN	...
$msg \in MESSAGES$	$msg \in MESSAGES$	WHERE
$msg \in sent$	$msg \in dr\_rcvd$	...
$msg \notin dr\_rcvd$	$msg \notin dr\_sent$	$\ominus$ $msg \in sent$
THEN	THEN	$\oplus$ $msg \in dr\_sent$
$dr\_rcvd := dr\_rcvd \cup \{msg\}$	$dr\_sent := dr\_sent \cup \{msg\}$	...

[2] $\oplus$: to add an element to a model, $\ominus$: to remove an element from a model, ...: unchanged parts.

This model expresses an abstraction of ANYCAST RP, as follows: $P' \rightsquigarrow Q$, with $P' \mathrel{\widehat{=}} (s = source(msg) \wedge msg \notin sent)$ and $Q \mathrel{\widehat{=}} (msg \in got)$. We can see here that the initial predicate $P \mathrel{\widehat{=}} (s = source(msg) \wedge msg \notin got)$ is transformed into P', which is more detailed and more precise, saying that the message *msg* is not received because it has not yet been sent. An additional step related to the Designated Routers is added: $P' \rightsquigarrow R \wedge R \rightsquigarrow Q$, with an intermediate step R being $msg \in dr\_rcvd$.

Decomposing ANYCAST RP into Phases. The model ANYCAST_M2 introduces another intermediate routing: messages must be redirected to their destinations by routers called Rendezvous Points (*RP*). New variables are added in this refinement: *rp_rcvd* represents the messages received by some selected (not yet identified at this level of *abstraction*) Rendezvous Points (*RP*) from Designated Routers (*DR*) and *rp_sent* depicts the messages sent by selected Rendezvous Points (*RP*) to destinations; simple invariants on these variables are defined: **(1)** $rp\_rcvd \subseteq dr\_sent$, **(2)** $rp\_sent \subseteq rp\_rcvd$, **(3)** $got \subseteq rp\_sent$, **(4)** $got \subseteq rp\_rcvd$, **(5)** $rp\_rcvd \cap lost = \varnothing$. The last invariant describes an assumption on the system: messages can only be lost between selected Designated Routers (*DR*) and Rendezvous Points (*RP*). Events are refined or added: RP_RECEIVING2 models the receiving of a message (*msg*) by a selected Rendezvous Point, from a Designated Router; RP_SENDING2 presents the sending of a message (*msg*) by selected Rendezvous Point to destinations; RECEIVING2 depicts the receiving of a message (*msg*) by all the destinations of the message; LOSING2 models the losses of messages between Designated Routers (*DR*) and Rendezvous Points (*RP*).

EVENT RP_RECEIVING2
ANY
msg
WHERE
$msg \in MESSAGES$
$msg \in dr\_sent$
$msg \notin lost$
THEN
$rp\_rcvd := rp\_rcvd \cup \{msg\}$

EVENT RP_SENDING2
ANY
msg
WHERE
$msg \in MESSAGES$
$msg \in rp\_rcvd$
THEN
$rp\_sent :=$
$rp\_sent \cup \{msg\}$

EVENT RECEIVING2
...
WHERE
$\ominus\ msg \in dr\_sent$
$\ominus\ msg \notin (got \cup lost)$
$\oplus\ msg \in rp\_sent$
$\oplus\ msg \notin got$
...

EVENT LOSING2
...
WHERE
...
$\ominus\ msg \notin (got \cup lost)$
$\oplus\ msg \notin rp\_rcvd$
$\oplus\ msg \notin lost$
...

This model defines the entire dataflow that occurs during ANYCAST RP: **(1)** from sources to Designated Routers (*DR*), **(2)** from Designated Routers (*DR*) to Rendezvous Points (*RP*) and **(3)** from Rendezvous Points (*RP*) to destinations. This description of the complete dataflow is emphasized by the *refinement diagram* of the model:

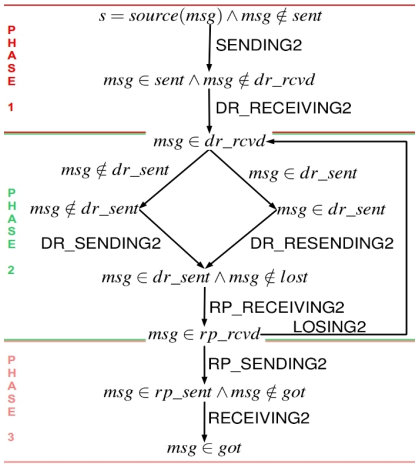


Fig. 7. Diagram $D2$ for ANYCAST_M2

The diagram $D2$ (see Fig.7) allows us to express ANYCAST RP using the following property: $P' \rightsquigarrow R \wedge R \rightsquigarrow S \wedge S \rightsquigarrow Q$, with $P' \mathrel{\widehat{=}} (s = source(msg) \wedge msg \notin sent)$, $R \mathrel{\widehat{=}} (msg \in dr\_rcvd)$, $S \mathrel{\widehat{=}} (msg \in rp\_rcvd)$, and $Q \mathrel{\widehat{=}} (msg \in got)$. We have decomposed $R \rightsquigarrow Q$, by transitivity, to add a new step related to the additional routing (Rendezvous Points). Moreover, the diagram allows us to identify phases of the protocol: **(Phase 1)** Routing from sources to Designated Routers, **(Phase 2)** Routing from Designated Routers to Rendezvous Points, **(Phase 3)** Routing from Rendezvous Points to Destinations.

The three identified phases are independent and can be developed separately: we have **(1)** $P' \rightsquigarrow R$, **(2)** $R \rightsquigarrow S$ and **(3)** $S \rightsquigarrow Q$. The development in phases is driven by the location of a message/packet in the network and by the type of nodes.

Combining and Coordinating Phases. We have divided ANYCAST RP into three *components*, described by the diagram $D2$ (see Fig.7), in the previous section. Since we develop the components independently, we relax the conditions $msg \in dr\_rcvd$, $msg \in rp\_rcvd$. We replace them by $msg \in MESSAGES$. The goal is to reintroduce these conditions (or their refined forms) in the events to add control and coordination during the combination of phases. Abstract models of each phase are composed of the events of ANYCAST_M2 related to **(1)** $P' \rightsquigarrow R$, **(2)** $R \rightsquigarrow S$ and **(3)** $S \rightsquigarrow Q$ and modified as said previously (relaxing some conditions). The next paragraphs give the development of each phase.

Phase 1: From sources to Designated Routers. We introduce the identity of a selected Designated Router (dr), which receives a message (msg) sent by a source (s). We notice that the variables $sent$, dr_rcvd have been replaced by $msg\_sent\_by\_src$ and $msg\_rcvd\_by\_dr$. These variables associate sent/received messages with the identities of senders (sources) and receivers (Designated Routers).

EVENT SENDING
ANY
$s, msg, dr, dest\_grp$
WHERE
$s \in SOURCES \wedge msg \in MESSAGES$
$dest\_grp \subseteq DR \wedge dr \in DR \wedge s = source(msg)$
$dest\_grp = \{dr\} \wedge s \mapsto msg \notin msg\_sent\_by\_src$
THEN
$msg\_sent\_by\_src := msg\_sent\_by\_src \cup \{s \mapsto msg\}$
$dr\_dest := dr\_dest \cup (dest\_grp \times \{msg\})$

EVENT DR_RECEIVING
ANY
dr, msg
WHERE
$dr \in DR \wedge msg \in MESSAGES$
$dr \mapsto msg \in dr\_dest$
$dr \mapsto msg \notin msg\_rcvd\_by\_dr$
THEN
$msg\_rcvd\_by\_dr := msg\_rcvd\_by\_dr \cup \{dr \mapsto msg\}$

SENDING models the sending of a message (*msg*) by a source (*s*) to a Designated Router (*dr*). A variable *dest_grp* indicates the Designated Router (*dr*) target of a message (*msg*). RECEIVING presents the receiving of a message (*msg*) by a Designated Router (*dr*).

Phase 2: From Designated Routers to Rendezvous Points. This model identifies the selected Designated Router (*dr*), sender of a message (*msg*) and the chosen Rendezvous Point (*rp*), target of the message.

EVENT DR_SENDING
ANY
$dr, rp, msg, dest\_grp$
WHERE
$dr \in DR \wedge rp \in RP \wedge dest\_grp \subseteq RP$
$dest\_grp = \{rp\} \wedge msg \in MESSAGES$
$msg \notin ran(msg\_sent\_by\_dr)$
THEN
$msg\_sent\_by\_dr := msg\_sent\_by\_dr \cup \{dr \mapsto msg\}$
$rp\_dest := rp\_dest \cup (dest\_grp \times \{msg\})$

EVENT LOSING
ANY
msg, dr, rp
WHERE
$dr \in DR \wedge rp \in RP \wedge msg \in MESSAGES$
$dr \mapsto msg \in msg\_sent\_by\_dr$
$rp \mapsto msg \in rp\_dest \wedge msg \notin lost$
$rp \mapsto msg \notin msg\_rcvd\_by\_rp$
THEN
$lost := lost \cup \{msg\}$

EVENT DR_RESENDING
ANY
msg, dr, rp
WHERE
$dr \in DR \wedge rp \in RP \wedge msg \in MESSAGES$
$dr \mapsto msg \in msg\_sent\_by\_dr \wedge rp \mapsto msg \in rp\_dest$
THEN
$lost := lost \setminus \{msg\}$

EVENT RP_RECEIVING
ANY
rp, msg
WHERE
$rp \in RP \wedge msg \in MESSAGES$
$rp \mapsto msg \in rp\_dest \wedge msg \notin lost$
THEN
$msg\_rcvd\_by\_rp := msg\_rcvd\_by\_rp \cup \{rp \mapsto msg\}$

We use the same techniques as in phase 1 to identify the senders and receivers of a message, namely Designated Routers and Rendezvous Points: the variables *dr_sent*, *rp_rcvd* are replaced with *msg_sent_by_dr* and *msg_rcvd_by_rp*, which associate sent/received messages with the identities of senders and receivers; *rp_dest* indicates the selected Rendezvous Points destinations of sent messages.

Phase 3: From Rendezvous Points to Destinations. The identities of selected Rendezvous Points (*rp*) sending messages (*msg*) to destinations are introduced by this model.

EVENT RP_SENDING
ANY
rp, msg
WHERE
$rp \in RP \wedge msg \in MESSAGES$
THEN
$msg\_sent\_by\_rp := msg\_sent\_by\_rp \cup \{rp \mapsto msg\}$

EVENT RECEIVING
ANY
gt, msg
WHERE
$gt \subseteq TARGETS \wedge msg \in MESSAGES \wedge msg \notin got$
$gt = group\_target(msg) \wedge msg \in ran(msg\_sent\_by\_rp)$
THEN
$got := got \cup \{msg\}$

This model is simple to understand: a Rendezvous Points (*rp*) sends (RP_SENDING) a message (*msg*) to a group of destinations (g_t), which receive the message (RECEIVING).

We show how the uses of *refinement diagrams* and the *service-as-event* paradigm help in the models (*components*) combination and coordination. First, we draw the *refinement diagrams* for each phase, and then, we add/combine predicates to link the diagrams and models. The *diagram D* (see Fig.8) shows three *sub-diagrams* for each phase, and demonstrates how the phases can be coordinated to obtain a formal model of ANYCAST RP: to link two consecutive phases, we form a conjunction with the *post-condition* of the first phase and the *pre-condition* of the other one; for example, to combine phases 1 and 2, we use a property resulting of the conjunction of the *pre/post*:

$dr \in DR \wedge dest\_grp = \{rp\} \wedge dr \mapsto msg \in msg\_rcvd\_by\_dr$, meaning that only a message (msg), received by a Designated Router (dr), can be sent to a Rendezvous Point (rp).

The same applies for the combination of phases 2 and 3, as we also use the result of the conjunction of the *pre/post*: $rp \mapsto msg \in msg\_rcvd\_by\_rp$. We notice that in fact, the combination is equivalent here to the fact of linking and ordering *sending* and *receiving services*, e.g. $sending1 \rightsquigarrow receiving1 \wedge receiving1 \rightsquigarrow sending2 \wedge ...$ These operations on *refinement diagrams* are reflected in the resulting model after integrating phases.

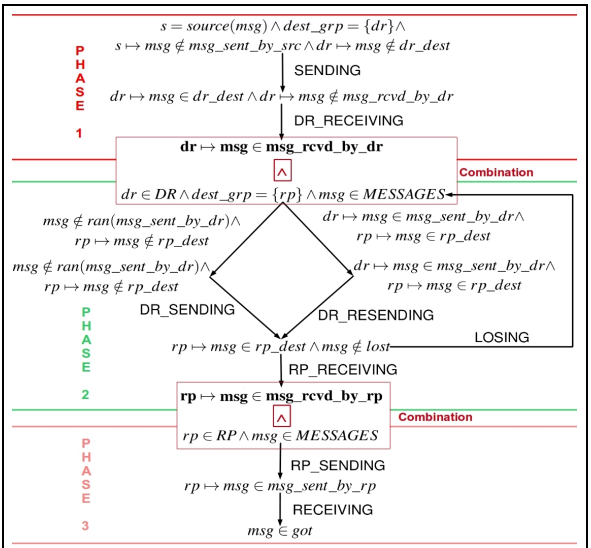
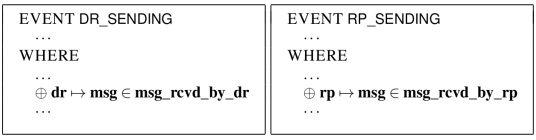


Fig. 8. Diagram D for Phases Combination

A set of invariants related to ordering and coordinating for *sending* and *receiving* services is added to the model: **msg_sent_by_dr ⊆ msg_rcvd_by_dr** and **msg_sent_by_rp ⊆ msg_rcvd_by_rp** express that only received messages are sent by respectively selected Designated Routers and Rendezvous Points. According to the diagram 8, the *pre-conditions* of the *sending* services DR_SENDING, RP_SENDING must also be modified. Therefore, we propose to modify these events as follows, by strengthening their guards: We add new guards that state that *receiving* services have to occur before following *sending* ones.



The model expresses the following property (see Fig.8), which describes ANYCAST RP: $P \rightsquigarrow R \quad \wedge \quad R \rightsquigarrow S \quad \wedge \quad S \rightsquigarrow Q$, with $P \mathrel{\widehat{=}} (s = source(msg) \wedge dest\_grp = \{dr\} \wedge s \mapsto msg \notin msg\_sent\_by\_src \wedge dr \mapsto msg \notin dr\_dest)$, $R \mathrel{\widehat{=}} (dr \in DR \wedge dest\_grp = \{rp\} \wedge dr \mapsto msg \in msg\_rcvd\_by\_dr)$, $S \mathrel{\widehat{=}} (rp \mapsto msg \in msg\_rcvd\_by\_rp)$, $Q \mathrel{\widehat{=}} (msg \in got)$. We refine this model, until we obtain a *local model* (where events are local to nodes of the network), from which an algorithmic form of the protocol can be derived. An interesting property of these kinds of combination is that one can develop the phases separately and choose at which level of refinement the combination will occur. Moreover, the splitting of ANYCAST RP into small pieces helped us to concentrate our main efforts on finding correct ways of composing and

coordinating the models of the phases, understanding and discharging the proof obligations generated by the integration of models.

6 Discussion, Conclusion and Future Work

We have introduced the *service-as-event* paradigm, as an integration of the EVENT B language with temporal notations and diagrams to cope with liveness properties, system decomposition and *components* integration, and as an extension of the *call-as-event* paradigm [19]. The diagrammatic notation describing *services*, namely *the refinement diagrams*, provides a graphical mean to support the intuition. These diagrams are particularly suited for guiding refinement-based development, because their refinement is possible [8]. The underlying semantical framework behind them is based on trace semantics and temporal structures, derived from TLA. In the literature, Manna and Pnueli [16] developed a collection of verification techniques based on *verification diagrams*, which are related to proving various temporal properties (invariance, safety, fairness, liveness, etc.) of reactive systems. They introduced different diagrams (WAIT-FOR and INVARIANCE diagrams, CHAIN diagrams, etc.), which are related to proof rules for deriving these properties. Our refinement diagrams are similar but we use them for refinement and they are integrated to the EVENT B models; the objectives are clearly to help in the refinement of complex systems and to decompose systems into subsystems in a *correct-by-construction* process. UNITY [9, 18] proposes also a combination of temporal logic and actions systems using the superposition technique, and a modelling of distributed and parallel programs under weak fairness, which is a limitation for expressing general fairness assumptions.

Our case study (ANYCAST RP) is simple to understand, because the protocol contains three identifiable, consecutive and independent routing phases, expressed as follows: $msg \notin sent \leadsto msg \in sent \wedge msg \in sent \leadsto msg \in received$. This simplicity hides technical details of the Event B models of the phases. In fact, the decomposition of the (complex) problem into smaller sub-problems allows us to discharge easy proof obligations related to parts of the algorithm and helps us to focus our efforts on the integration of models. However, decompositions of systems may present more difficulties and require a clever analysis. The application of the *service-as-event* paradigm and *refinement diagrams* is effective for modelling distributed algorithms with behaviours that can be decomposed into strict, sequential and/or non-interfering (or with little interferences) phases local to nodes: we have solved other case studies, related to Network-on-Chips [5], especially the *XY routing* and *network dynamic reconfiguration* services.

Our future works involve the connection of our approach to a platform integrating real concurrency concepts related to effective programming languages based on the *service-as-event* paradigm. Moreover, we plan to delve into the topic of feature interactions and how interferences can be taken into account. Another point is the relation between the complexity of proofs and models reuse for the description of the routing phases. The reuse and the adaptation of formal models are related to formal design patterns [3]. Finally, we intend to develop few more case studies related to distributed networks, and our goal is to develop a toolbox that can be used to implement distributed protocols using a programming language, where the toolbox will transform

the verified formal specifications of EVENT B models [5, 6] into a given programming language.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering (2010)
2. Abrial, J.-R., Cansell, D., Méry, D.: A mechanically proved and incremental development of ieee 1394 tree identify protocol. Formal Asp. Comput. 14(3), 215–227 (2003)
3. Abrial, J.-R., Hoang, T.S.: Using design patterns in formal methods: An event-B approach. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) ICTAC 2008. LNCS, vol. 5160, pp. 1–2. Springer, Heidelberg (2008)
4. Abrial, J.-R., Mussat, L.: Introducing Dynamic Constraints in B. In: B98, pp. 83–128 (1998)
5. Andriamiarina, M.B., Daoud, H., Belarbi, M., Méry, D., Tanougast, C.: Formal Verification of Fault Tolerant NoC-based Architecture. In: First International Workshop on Mathematics and Computer Science (IWMCS 2012), Tiaret, Algérie (December 2012)
6. Andriamiarina, M.B., Méry, D., Singh, N.K.: Revisiting Snapshot Algorithms by Refinement-based Techniques. In: PDCAT, IEEE Computer Society (2012)
7. Back, R.-J., Sere, K.: Stepwise refinement of action systems. Structured Programming 12(1), 17–30 (1991)
8. Cansell, D., Méry, D., Merz, S.: Diagram refinements for the design of reactive systems. J. UCS 7(2), 159–174 (2001)
9. Chandy, K.M., Misra, J.: Parallel Program Design A Foundation. Addison-Wesley Publishing Company (1988) ISBN 0-201-05866-9
10. Cisco Systems. Anycast RP, `http://www.cisco.com/en/US/docs/ios/solutions_docs/ip_multicast/White_papers`
11. Cisco Systems. Anycast RP using PIM, `http://tools.ietf.org/html/draft-ietf-pim-anycast-rp-07`
12. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
13. Kang, J., Sucec, J., Kaul, V., Samtani, S., Fecko, M.A.: Robust pim-sm multicasting using anycast rp in wireless ad hoc networks. In: Proceedings of the 2009 IEEE International Conference on Communications, ICC 2009, pp. 5139–5144. IEEE Press, Piscataway (2009)
14. Lamport, L.: A temporal logic of actions. ACM Trans. Prog. Lang. Syst. 16(3), 872–923 (1994)
15. Leavens, G.T., Abrial, J.-R., Batory, D.S., Butler, M.J., Coglio, A., Fisler, K., Hehner, E.C.R., Jones, C.B., Miller, D., Jones, S.L.P., Sitaraman, M., Smith, D.R., Stump, A.: Roadmap for enhanced languages and methods to aid verification. In: Jarzabek, S., Schmidt, D.C., Veldhuizen, T.L. (eds.) GPCE, pp. 221–236. ACM (2006)
16. Manna, Z., Pnueli, A.: Temporal verification diagrams. In: Hagiya, M., Mitchell, J.C. (eds.) TACS 1994. LNCS, vol. 789, pp. 726–765. Springer, Heidelberg (1994)
17. McIver, A., Morgan, C.: Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science). Springer (2004)
18. Méry, D.: Requirements for a temporal B: Assigning Temporal Meaning to Abstract Machines. and to Abstract Systems. In: Galloway, A., Taguchi, K. (eds.) IFM 1999 Integrated Formal Methods 1999, YORK (June 1999)
19. Méry, D.: Refinement-based guidelines for algorithmic systems. Int. J. Software and Informatics 3(2-3), 197–239 (2009)
20. Méry, D., Singh, N.K.: Analysis of DSR protocol in event-B. In: Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems, SSS 2011, pp. 401–415. Springer-Verlag, Heidelberg (2011)

21. Owicki, S., Gries, D.: An axiomatic proof technique for parallel programs I. Acta Informatica 6, 319–340 (1976)
22. Owicki, S., Lamport, L.: Proving liveness properties of concurrent programs. ACM Trans. Program. Lang. Syst. 4(3), 455–495 (1982)
23. Owre, S., Shankar, N.: A brief overview of PVS. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 22–27. Springer, Heidelberg (2008)
24. Project RODIN. Rigorous open development environment for complex systems (2004-2010), http://www.eventb.org/
25. Rehm, J., Cansell, D.: Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method. In: ISoLA, pp. 179–190 (2007)
26. Tanenbaum, A.S.: Computer networks (4. ed.). Prentice-Hall (2002)
27. Wenzel, M., Paulson, L.C., Nipkow, T.: The Isabelle Framework. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 33–38. Springer, Heidelberg (2008)