

# A Guided Tour on Total Order Specifications\*

Stefano Cimmino, Carlo Marchetti and Roberto Baldoni  
Dipartimento di Informatica e Sistemistica  
Università di Roma “La Sapienza”  
Via Salaria 113, 00198, Roma, Italy  
email: {cimmino,marchet,baldoni}@dis.uniroma1.it

## Abstract

*In the last two decades the development of Total Order (TO) broadcast and multicast communication over asynchronous distributed systems have been one of the main research issues in dependable distributed computing. As a result, a huge amount of works has been carried out, ranging from service specifications to a variety of TO implementations over different communication platforms. Differences among such specifications can make very difficult the choice of the right TO primitive, by an application designer, to enable the application to meet its correctness requirements. The aim of this paper is thus to present a clear classification of total order broadcast specifications. In particular, six specifications of total order broadcast primitives proposed in the literature are organized into a hierarchy that allows (i) to classify existing implementations of total order communication primitives, and (ii) to select the right primitive according to the application requirements in order to maximize performance.*

## 1. Introduction

Since the Lamport’s seminal paper [13] in 1978, the problem of total order (TO) communication in asynchronous distributed systems in the presence of process crashes has been extensively studied in the literature. Intuitively, a TO primitive ensures that *all* processes deliver the same sequence of messages until they possibly crash<sup>1</sup>. In the last years, researchers pointed out several application scenarios where the use of TO communications is extremely useful, e.g. applications having to maintain consistency among the internal states of a set of deterministic

replicas (active software replication [14]) or when facing the problem of providing a set of stock quotes in the same order to a set of stock operators to preserve fairness of exchanges [5]. Only recently industries start to understand the importance of such a paradigm in an asynchronous environment, especially when implementing mission critical systems. Air traffic control and space missions are examples of business areas where there is a growing interest around products centered on asynchronous TO communications. The capability of these communication primitives to work over a distributed system that modifies its degree of synchrony during its lifetime (in terms of either message transfer delay and/or the time taken by a process to execute a computational step) has indeed a positive impact on the design, test, deployment and maintenance of a product. To let this growing interest have a real impact on final products, it is mandatory to provide a clear and unique framework that can be used to classify specifications, implementations and platforms<sup>2</sup>. The set of runs that can indeed be produced by distinct TO implementations can considerably differ one each other.

The research on TO communications during the last two decades did not answer this need of clarification by producing tens of TO implementations and several TO specifications whose differences and similarities among themselves are often left unclear. Concerning TO specifications, it is actually not uncommon the case in which TO primitives are either loosely specified (e.g. [2]) or formalized using ad-hoc notations (e.g. [12]), which are difficult to compare. Even in case of similar formalisms, differences between various TO specifications are often left hidden. Similar considerations hold about TO implementations. Therefore, developers of distributed applications needing a TO primitive are presented with a plethora of distinct systems, each providing its own guarantees (i.e. enforced TO speci-

---

\* This work has been partially supported by a grant from EU IST Project “EU-PUBLICOM” (#IST-2001-35217), and by a grant from MIUR on the context of project “MAIS”

1 Given the intuitive definition of total order we do not consider best effort approaches to total ordering which trade scalability and performance for delivery guarantees on the set of correct processes [9, 4, 8].

2 Recently a nice survey has appeared in the literature [6] which covers all group communication multicast primitives. In this paper we only focus on TO broadcast primitives.

---

$correct(p)$	$\triangleq$	$crash \notin h_p$
$faulty(p)$	$\triangleq$	$crash \in h_p$
$send(p, m)$	$\triangleq$	$TOsend(m) \in h_p$
$del(p, m)$	$\triangleq$	$TOdeliver(m) \in h_p$
$del(p, m) < del(p, m')$	$\triangleq$	$\exists i, j \ e_i = TOdeliver(m) \in h_p \wedge e_j = TOdeliver(m') \in h_p \wedge i < j$

---

**Table 1. Shorthand predicates definition**

---

fication) and performance. A bad choice of a TO primitive can either be catastrophic for its application (if the selected TO implementation is able to produce runs which are not admissible by the application) or negatively affect performances (if the selected TO implementation enforces a very strong TO specification with respect to one that could be tolerated by the application).

This paper analyzes six distinct TO specifications, explaining differences among them in terms of their admitted sets of runs. Specifications are given using a well-defined formal model, and are organized into a hierarchy, which is an extension of the one introduced by Wilhelm and Schiper in [15]. The six specifications are obtained by varying two of the four properties defining the TO problem, namely *Agreement* and *Order*<sup>3</sup>. The hierarchy represents a unique framework allowing to compare distinct TO implementations in terms of the TO specification they enforce.

The remainder of this paper is organized as follows. Section 2 presents the system model. Section 3 shows the hierarchy of TO specifications, which is achieved by a detailed study of the properties defining the TO problem (Appendix A contains formal proofs). Finally, Section 4 concludes the paper.

## 2. System model

**Asynchronous distributed system.** We consider a system composed by a finite set of processes  $\Pi = \{p_1 \dots p_n\}$  communicating by message passing. Each process behaves according to its specification until it possibly crashes. A process that never crashes is *correct*, while a process that crashes is denoted as *faulty*. Processes exchange messages by means of *quasi-reliable* channels [3]. As a consequence, messages sent by correct processes are eventually delivered by correct processes (there is no bound known or unknown to the message transfer delay). In contrast, communications from and to faulty processes are unreliable. In order to broadcast a message  $m$ , a process invokes the  $TOsend(m)$  primitive. Upon receiving a message  $m$ , the underlying layer of a process invokes the  $TOdeliver(m)$  primitive, which is an upcall used to deliver  $m$  to the process.

**Histories and runs.** Each process  $p \in \Pi$  can experience the occurrence of three types of events, namely  $TOsend(m)$ ,  $TOdeliver(m)$  or *crash*. An history  $h_p$  is the sequence of events occurred at  $p$  during its lifetime. We denote as  $e_i \in h_p$  the  $i$ -th event in the history of  $p$ . Note that *crash* may only occur as the last event in the history of a faulty process. A *system run* is a set of histories  $h_{p_i}$ , one for each process  $p_i \in \Pi$ . We denote as  $\mathcal{R}$  the set of all possible runs in the system. To characterize runs in  $\mathcal{R}$ , we introduce the predicates defined in Table 1.

**Properties and specifications.** A *property*  $P$  on  $\mathcal{R}$  is a predicate on  $\mathcal{R}$  defining a set  $R_P \subseteq \mathcal{R}$  of runs. More precisely, a run  $r \in \mathcal{R}$  is *admitted* by  $P$ , i.e.  $r \in R_P$ , iff  $r$  satisfies  $P$ . Let  $P$  and  $P'$  be two properties on  $\mathcal{R}$ ,  $P \Rightarrow P'$  iff  $R_P \subseteq R_{P'}$ . If  $P \Rightarrow P'$ , we say that  $P$  is stronger than  $P'$ , and that  $P'$  is weaker than  $P$ .

A *specification*  $S(P_1 \dots P_m)$  (with  $m \geq 1$ ) on  $\mathcal{R}$  is a predicate on  $\mathcal{R}$  composed by the logical and of  $m$  properties, i.e.  $S = \bigwedge_{i=1 \dots m} P_i$ , defining a set  $R_S = \bigcap_{i=1 \dots m} R_{P_i} \subseteq \mathcal{R}$ .  $R_S$  is composed by all system runs satisfying  $S$ . More precisely, a run  $r \in \mathcal{R}$  is *admitted* by a specification  $S(P_1 \dots P_m)$ , i.e.  $r \in R_S = \bigcap_{i=1 \dots m} R_{P_i}$ , iff  $r$  satisfies  $S$ . Given two specifications  $S(P_1 \dots P_m)$  and  $S'(P'_1 \dots P'_\ell)$  (with  $m$  not necessarily equal to  $\ell$ ),  $S$  is stronger than  $S'$ , denoted  $S \rightarrow S'$ , iff  $R_S \subseteq R_{S'}$ . In this case we also say that  $S'$  is weaker than  $S$ .

## 3. Total order specifications

Total order broadcast is commonly specified through four properties, namely *Validity*, *Integrity*, *Agreement*, and *Order*. Informally speaking, a *Validity* property guarantees that messages sent by correct processes will eventually be delivered at least by correct processes; an *Integrity* property guarantees that no spurious or duplicate messages will be delivered; an *Agreement* property ensures that (at least correct) processes deliver the same set of messages; an *Order* property constrains (at least correct) processes delivering the same messages to deliver them in the same order. Each property can be formally defined in distinct ways, thus generating distinct specifications. A typical example of differing formulations of a property of a TO specification is given by its *uniform* and *non-uniform* versions. A uniform property imposes some restrictions on the histories of (at least)

---

<sup>3</sup> The hierarchy of [15] has been also extended in [7] by Dèfago in another direction by varying a third property, namely *Integrity*.

---

$NUV$	$\triangleq$	$\forall p \forall m \text{ send}(p, m) \wedge \text{correct}(p) \Rightarrow \exists q \text{ del}(q, m) \wedge \text{correct}(q)$
$UI$	$\triangleq$	$\forall m \forall p e_i = \text{TOdeliver}(m) \in h_p \Rightarrow (\exists q \text{ send}(q, m) \wedge \forall e_j \in h_p e_i = e_j \Leftrightarrow i = j)$
$UA$	$\triangleq$	$\forall p \forall m \text{ del}(p, m) \Rightarrow (\forall q \text{ correct}(q) \Rightarrow \text{del}(q, m))$
$NUA$	$\triangleq$	$\forall p \forall m \text{ correct}(p) \wedge \text{del}(p, m) \Rightarrow (\forall q \text{ correct}(q) \Rightarrow \text{del}(q, m))$
$SUTO$	$\triangleq$	$\forall p \forall m, m' \text{ del}(p, m) < \text{del}(p, m') \Rightarrow (\forall q \text{ del}(q, m') \Rightarrow \text{del}(q, m) < \text{del}(q, m'))$
$SNUTO$	$\triangleq$	$\forall p \forall m, m' \text{ correct}(p) \wedge \text{del}(p, m) < \text{del}(p, m') \Rightarrow (\forall q \text{ correct}(q) \wedge \text{del}(q, m') \Rightarrow \text{del}(q, m) < \text{del}(q, m'))$
$WUTO$	$\triangleq$	$\forall p, q \forall m, m' \text{ del}(p, m) \wedge \text{del}(p, m') \wedge \text{del}(q, m) \wedge \text{del}(q, m') \Rightarrow (\text{del}(p, m) < \text{del}(p, m') \Leftrightarrow \text{del}(q, m) < \text{del}(q, m'))$
$WNUTO$	$\triangleq$	$\forall p, q \forall m, m' \text{ correct}(p) \wedge \text{correct}(q) \wedge \text{del}(p, m) \wedge \text{del}(p, m') \wedge \text{del}(q, m) \wedge \text{del}(q, m') \Rightarrow (\text{del}(p, m) < \text{del}(p, m') \Leftrightarrow \text{del}(q, m) < \text{del}(q, m'))$

---

**Table 2. Formal definition of the properties defining TO specifications**

---

correct processes on the basis of some events (or event patterns) occurred in the history of some processes (correct or not). In contrast, a non-uniform property imposes some restrictions on the histories of correct processes on the basis of some events (or event patterns) occurred in the history of some *correct* processes. If these restrictions are not verified the properties are not satisfied. As a consequence, given a property  $P$ , its uniform formulation  $UP$  turns out to be stronger than its non-uniform formulation  $NUP$ . Therefore,  $UP \Rightarrow NUP$ , and  $R_{UP} \subset R_{NUP}$ .

In the following we discuss the four previous properties, introducing their formal definition and highlighting the differences related to their various formulations.

### 3.1. Validity and Integrity

The assumption of quasi-reliable channels makes it impossible to guarantee that a message sent by a faulty process will be eventually delivered by some correct process. Therefore, we consider only the non-uniform version of the *Validity* property, defined as follows<sup>4</sup>:

**Non-uniform Validity ( $NUV$ ).** If a *correct* process tocasts<sup>5</sup> a message  $m$ , then some correct process will eventually todeliver  $m$ .

Concerning the *Integrity* property, the crash fault model assumption allows to easily enforce its uniform version without additional overhead. Hence, we only deal with *Uniform Integrity*, defined as follows:

**Uniform Integrity ( $UI$ ).** For any message  $m$ , every process  $p$  todelivers  $m$  at most once, and only if  $m$  was previously tocast by some process.

In the remainder of the paper we will consider only specifications containing *Non-uniform Validity* and *Uniform Integrity* and differing for the *Agreement* and *Order* proper-

ties<sup>6</sup>. Therefore, for the sake of simplicity, we denote as  $TO(A, O)$  the TO specification composed by the *Agreement* property  $A$  and the *Order* property  $O$ , along with  $NUV$  and  $UI$ .

### 3.2. The Agreement property

The role of the *Agreement* property is to impose some constraints on the set of messages delivered by (at least correct) processes. This property is usually specified with one of the following formulations:

**Uniform Agreement ( $UA$ ).** If a process todelivers a message  $m$ , then all correct processes will eventually todeliver  $m$ ;

**Non-uniform Agreement ( $NUA$ ).** If a *correct* process todelivers a message  $m$ , then all correct processes will eventually todeliver  $m$ .

Generally speaking, both formulations enforce all correct processes to deliver the same set of messages. The difference between the two properties lies in the restrictions imposed on the set of messages that can be delivered by faulty processes. Let us first consider  $UA$ . This property imposes that each message delivered by some process (correct or not) is also delivered by every correct process (in any order). In contrast, faulty processes are allowed to omit the delivery of some messages delivered by some other process. This implies that when considering a run satisfying  $UA$ , a faulty process delivers a subset of the messages delivered by correct processes. Figure 1(a) depicts an example of a run satisfying  $UA$ .

On the contrary,  $NUA$  allows faulty processes to deliver messages that are not delivered by any other process (e.g. message  $m_5$  delivered by faulty process  $p_3$  in Figure 1(b)). Therefore, when considering a run satisfying  $NUA$ , the set of messages delivered by a faulty process intersects the set

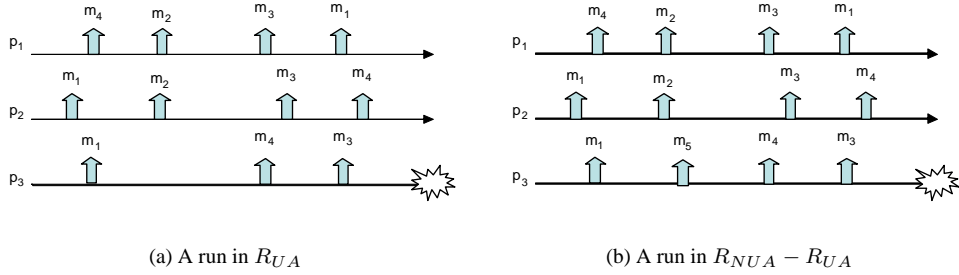
---

<sup>4</sup> The formal definition of all the properties analyzed in this section are shown in Table 2.

<sup>5</sup> We say that a process  $p \in \Pi$  tocasts a message  $m$  when it executes  $\text{TOsend}(m)$ . Analogously, we say that a process  $p \in \Pi$  todelivers a message  $m$  upon executing  $\text{TOdeliver}(m)$ .

---

<sup>6</sup> Considering only *Agreement* and *Order* allows us to restrict our attention to deliveries and crashes as the events characterizing system runs. Therefore figures in this section will only contain such events.



**Figure 1. Differences between  $UA$  and  $NUA$**

of messages delivered by correct processes (with the intersection possibly empty).

### 3.3. The Order property

The *Order* property has four formulations: Strong Uniform Total Order ( $SUTO$ ), Strong Non-uniform Total Order ( $SNUTO$ ), Weak Uniform Total Order ( $WUTO$ ) and Weak Non-uniform Total Order ( $WNUTO$ ). All these formulations constrain correct processes delivering the same messages to deliver them in the same order. However, they impose different restrictions concerning (i) the order of messages delivered by faulty processes, and (ii) the actual composition of the sets of messages delivered by processes. The remainder of this section analyzes differences and implications among these four formulations.

**$SUTO$ .** The set  $R_{SUTO}$  is composed by all system runs satisfying the following property:

**Strong Uniform Total Order ( $SUTO$ ).** If some process delivers some message  $m$  before message  $m'$ , then a process delivers  $m'$  only after it has delivered  $m$ .

$SUTO$  imposes that, until some process omits to deliver a message (which can even occur at the run starting time), all processes have delivered exactly the same ordered set of messages (as shown by the leftmost dashed line depicted in Figure 2(a)). The set of messages delivered by a process after a delivery omission has to be disjoint by the set of messages delivered by other processes. As an example, when process  $p_3$  in Figure 2(a) misses to deliver  $m_3$ , it is then constrained to deliver messages distinct by those delivered by  $p_1$  and  $p_2$ . Instead,  $p_1$  and  $p_2$  will continue to deliver the same ordered set of messages until  $p_2$  misses to deliver  $m_6$  (see the rightmost dashed line). In that case,  $p_2$  is forced to deliver messages distinct from  $p_1$ , other than from  $p_3$ .

**$WUTO$ .** The set  $R_{WUTO}$  is composed by all system runs satisfying the following property:

**Weak Uniform Total Order ( $WUTO$ ).** If processes  $p$  and  $q$  both deliver messages  $m$  and  $m'$ , then  $p$  delivers  $m$  before  $m'$  if and only if  $q$  delivers  $m$  before  $m'$ .

As  $SUTO$ ,  $WUTO$  is a uniform property. However,  $WUTO$  is weaker than  $SUTO$  (i.e.  $SUTO \Rightarrow WUTO$ , as shown by Lemma 1 in Appendix A) as it imposes an order only on pairs of messages delivered by pairs of distinct processes.  $WUTO$  does not prevent a process from omitting to deliver a message while still continuing to deliver the same messages delivered by other processes. As a consequence, the sequence of messages delivered by a process can contain holes with respect to the sequences of messages delivered by other processes. As an example, Figure 2(b) depicts a run in  $R_{WUTO} - R_{SUTO}$ :  $p_2$  delivers  $m_3$  after having missed  $m_2$ . In contrast,  $p_1$  and  $p_3$  deliver  $m_2 < m_3$ . Therefore  $p_2$  exhibits a hole in its sequence of message deliveries with respect to  $p_1$  and  $p_3$ .

**$SNUTO$ .** The set  $R_{SNUTO}$  is composed by all system runs satisfying the following property:

**Strong Non-uniform Total Order ( $SNUTO$ ).** If some correct process delivers some message  $m$  before message  $m'$ , then a correct process delivers  $m'$  only after it has delivered  $m$ .

$SNUTO$  is the non-uniform counterpart of  $SUTO$ . Hence, due to the relation between uniform and non-uniform properties,  $SUTO \Rightarrow SNUTO$ . The difference between these two properties lies in the behavior of faulty processes. In fact,  $SNUTO$  allows faulty processes to change the order of message deliveries, as well as occasionally omit the delivery of some message (see message  $m_3$  at process  $p_3$  in Figure 3(a)). In contrast, correct processes are forced (i) to agree on a prefix of the ordered set of delivered messages and (ii) to deliver messages distinct from those delivered by other correct processes after a delivery omission occur (see processes  $p_1$  and  $p_2$  in Figure 3(a)).

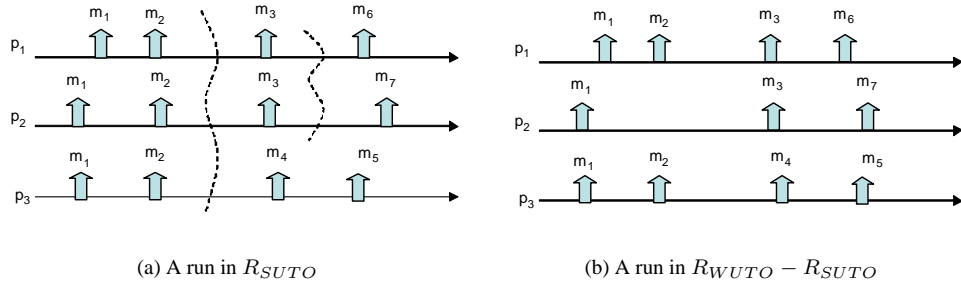


Figure 2. Differences between *SUTO* and *WUTO*

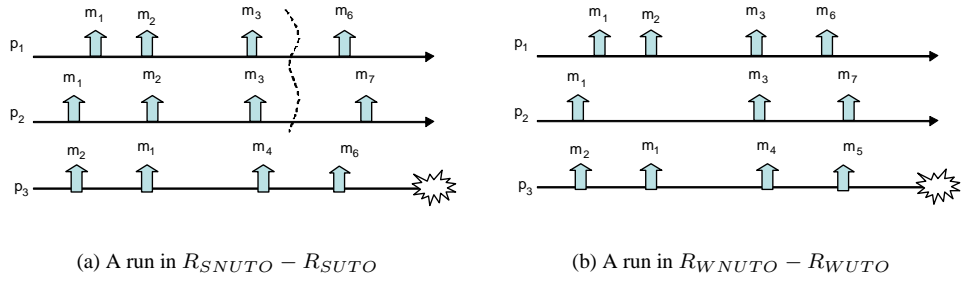


Figure 3. Differences between *SNUTO* and *SUTO* and between *WNUTO* and *WUTO*

**WNUTO.** The set  $R_{WNUTO}$  is composed by all system runs satisfying the following property:

**Weak Non-uniform Total Order (WNUTO).** If *correct* processes  $p$  and  $q$  both todeliver messages  $m$  and  $m'$ , then  $p$  todelivers  $m$  before  $m'$  if and only if  $q$  todelivers  $m$  before  $m'$ .

*WNUTO* is the non-uniform counterpart of *WUTO*. Hence  $WUTO \Rightarrow WNUTO$ . *WNUTO* allows faulty processes to change the order of message deliveries (see Figure 3(b)) while inheriting from *WUTO* the possibility to have holes in the sequence of message delivered by processes.

### 3.4. Combining Agreement and Order

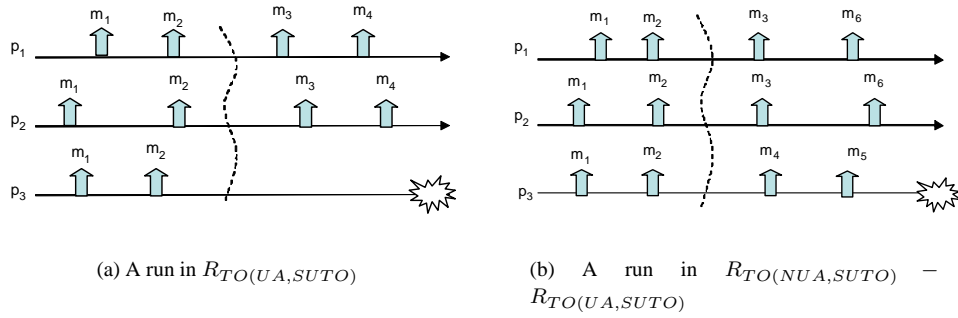
The aim of this section is to point out the TO specifications that can be obtained by combining the various formulations of the *Agreement* and *Order* properties, highlighting differences and relations among them.

**TO(UA, SUTO).** The interaction between *UA* and *SUTO* makes  $TO(UA, SUTO)$  the specification closest to the intuitive notion of total order broadcast. Figure 4(a) depicts an example of a run admitted by this specification. Let us consider a process, e.g.  $p_3$ , omitting to deliver a message, e.g.

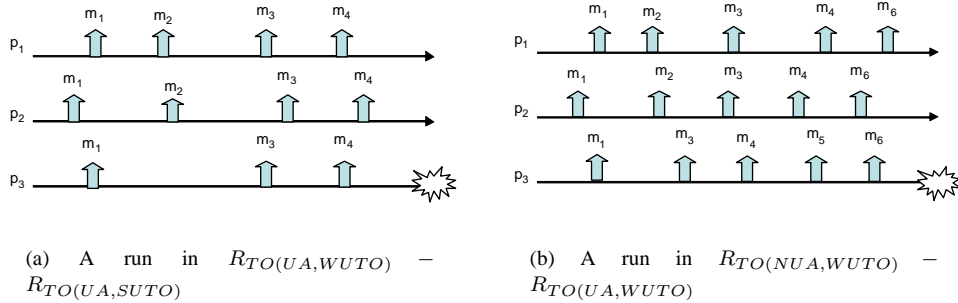
$m_3$ . There are two consequences: (i) due to *SUTO*, the successive messages delivered by  $p_3$  have to be distinct from those delivered by other processes and (ii) due to *UA*,  $p_3$  is faulty (sooner or later there will be a crash event in the history of  $p_3$ ). Now suppose that  $p_3$  delivers a new message  $m$  after missing  $m_3$  and before crashing. Then  $m$  has to be delivered by correct processes (due to *UA*), but this would violate *SUTO*. Hence a process like  $p_3$  cannot deliver any message after a delivery omission occur, and its sequence of delivered messages thus remains the same of other processes until it crashes. This explains why this specification is the closest to the intuitive notion of total order broadcast.

**TO(NUA, SUTO).** Being based upon *NUA*, this specification allows a faulty process to deliver spurious messages. Furthermore, due to *SUTO*, the set of messages delivered after the spurious one have to be disjoint from those delivered by other processes. As an example, Figure 4(b) depicts a run in  $R_{TO(NUA, SUTO)} - R_{TO(UA, SUTO)}$ , in which the faulty process  $p_3$  delivers the spurious message  $m_4$ . Since  $UA \Rightarrow NUA$ , this example suffices to show that  $TO(UA, SUTO) \rightarrow TO(NUA, SUTO)$ .

**TO(UA, WUTO).** Differently from  $TO(UA, SUTO)$ ,  $TO(UA, WUTO)$  allows a faulty process to occasionally omit the delivery of some message. This behavior is



**Figure 4. Differences between  $TO(UA, SUTO)$  and  $TO(NUA, SUTO)$**



**Figure 5. Differences between  $TO(UA, WUTO)$  and  $TO(NUA, WUTO)$**

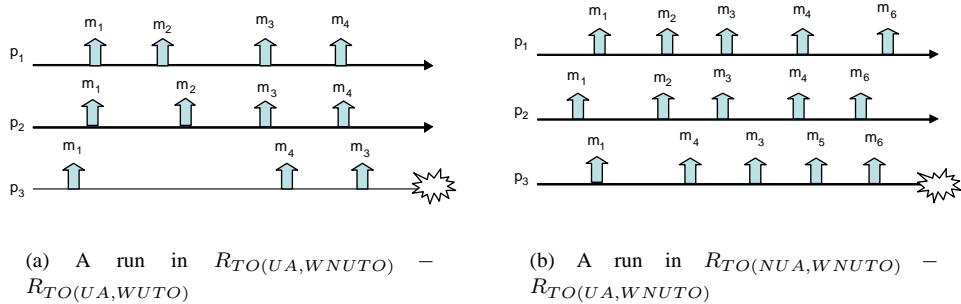
due to the presence of  $WUTO$  in the specification, but characterizes only faulty processes thanks to  $UA$ . In particular,  $UA$  enforces all correct processes to deliver exactly the same set of messages. Furthermore,  $WUTO$  ensures that the order of message deliveries is always the same among all processes. Figure 5(a) shows an example of a run in  $R_{TO(UA, WUTO)} - R_{TO(UA, SUTO)}$ , in which process  $p_3$  delivers  $m_3$  after having missed  $m_2$ , while correct processes deliver  $m_2 < m_3$ . Therefore  $p_3$  exhibits a hole in its sequence of delivered messages. Since  $SUTO \Rightarrow WUTO$ , this example suffices to show that  $TO(UA, SUTO) \rightarrow TO(UA, WUTO)$ .

**$TO(NUA, WUTO)$ .**  $TO(NUA, WUTO)$  can be obtained either from  $TO(NUA, SUTO)$  substituting  $SUTO$  with  $WUTO$  or from  $TO(UA, WUTO)$  substituting  $UA$  with  $NUA$ . Actually, it is easy to see that  $TO(NUA, SUTO) \rightarrow TO(NUA, WUTO)$  and  $TO(UA, WUTO) \rightarrow TO(NUA, WUTO)$ . In particular  $TO(NUA, WUTO)$  admits runs (e.g. the run depicted in Figure 5(b)) such that a faulty process (e.g.  $p_3$ ) (i) occasionally omits the delivery of some message (e.g.  $m_2$ ) due to  $WUTO$ , and (ii) delivers spurious mes-

sages (e.g.  $m_5$ ) due to  $NUA$ .

**$TO(UA, SNUTO)$  and  $TO(UA, WNUTO)$ .** Both  $WNUTO$  and  $SNUTO$  can be combined with  $UA$  to define a  $TO$  specification. In Appendix A (Lemma 2) we show that  $TO(UA, SNUTO) \equiv TO(UA, WNUTO)$ . Therefore in the following we will only consider  $TO(UA, WNUTO)$ . Since this specification contains  $UA$ , faulty processes are not allowed to deliver spurious messages. However, due to the presence of  $WNUTO$ , faulty processes are allowed (i) to occasionally omit to deliver some message and (ii) to deliver messages in an order different from the one chosen by correct processes. As an example, Figure 6(a) shows a run in  $R_{TO(UA, WNUTO)} - R_{TO(UA, WUTO)}$ , in which the faulty process  $p_3$  omits to deliver message  $m_2$ , and delivers  $m_4 < m_3$ , differently from correct processes. Since  $WUTO \Rightarrow WNUTO$ , this example shows that  $TO(UA, WUTO) \rightarrow TO(UA, WNUTO)$ .

**$TO(NUA, SNUTO)$  and  $TO(NUA, WNUTO)$ .** In a way similar to Lemma 2, it is possible to show that  $TO(NUA, SNUTO) \equiv TO(NUA, WNUTO)$ . Therefore, in the following we will consider only the latter specification. It is easy to see that  $TO(UA, WNU-$



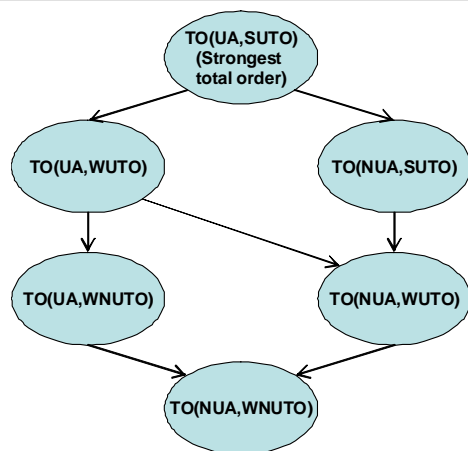
**Figure 6. Differences between  $TO(UA, WNUTO)$  and  $TO(NUA, WNUTO)$**

$TO) \rightarrow TO(NUA, WNUTO)$  and  $TO(NUA, WUTO) \rightarrow TO(NUA, WNUTO)$ . In particular,  $TO(NUA, WNUTO)$  allows faulty processes (i) to occasionally omit the delivery of some message, (ii) to deliver spurious messages and (iii) to deliver messages in an order different from that chosen by correct processes (see Figure 6(b)).

### 3.5. A hierarchy of total order specifications

Exploiting the precedence relation  $\rightarrow$  among specifications, it is possible to organize the TO specifications into the hierarchy depicted in Figure 7. This hierarchy is actually an extension of that introduced by Wilhelm and Schiper in [15].

Previous sections show that each specification weaker than  $TO(UA, SUTO)$  allows a faulty process to experience one (or more) of the following events: deliver spurious messages, exhibit an hole in the sequence of message deliveries, and deliver messages in an order not consistent with the one of correct processes. If we consider the first time such an event occurs in a faulty process  $p$ , then each message sent by  $p$  between the event and the crash of  $p$  may *contaminate*<sup>7</sup> the execution either of some other processes of  $\Pi$  (internal contamination) or of some processes/resources external to  $\Pi$  (external contamination). For example, process  $p_3$  in Figure 5(a) could contaminate some other processes by sending messages before crashing and after the delivery of message  $m_3$ . Let us remark that the occurrence of a “contamination” strictly depends on the semantics of the application. If the application tolerates the events described, this is clearly no longer a problem.



**Figure 7. A hierarchy of TO specifications**

## 4. Conclusion and future work

This paper presents a hierarchy of TO specifications which can be used by application designers to select the right TO specification enabling to meet their application requirements. Such a specification should be the weakest of the hierarchy not allowing dangerous runs for the application, i.e. runs containing events that could hinder the application to meet its requirements. Indeed, stronger specifications commonly require more resources and are thus more expensive in terms of achieved performances. Selecting the weakest specification satisfying the application requirements thus enables to maximize performances.

We are currently designing a *methodology* allowing to classify TO *implementations* according to the set of runs they can generate. We plan to apply this methodology to classify and compare several existing TO implementations<sup>8</sup>.

<sup>7</sup> The notion of contamination has been introduced for the first time by Gopal and Toueg in [10] with respect to the internal state of a process.

<sup>8</sup> We are studying and evaluating six TO implementations provided by three group toolkits, namely Ensemble [11], Spread [1], JavaGroups [2].

The comparison (both in terms of enforced specification and in terms of achieved performances) will provide designers and developers with a uniform framework to deal with TO communications.

## References

- [1] Y. Amir and J. Stanton. The Spread Wide Area Group Communication System. Technical Report CNDS-98-4, Center for Networking and Distributed Systems, Computer Science Department, Johns Hopkins University, 3400 N. Charles Street Baltimore, MD 21218-2686, April 1998.
- [2] B. Ban. Design and implementation of a reliable group communication toolkit for java. Cornell University, Sept. 1998.
- [3] A. Basu, B. Charron-Bost, and S. Toueg. Simulating Reliable Links with Unreliable Links in the Presence of Process Crashes. In *Proc. of Distributed Algorithm (WDAG '96)*, pages 105–122, October 1996.
- [4] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.
- [5] K. P. Birman. A Review of Experiences with Reliable Multicast. *Software – Practice and Experience*, 29(9):741–774, 1999.
- [6] G. Chockler, I. Keidar, and R. Vitenberg. Group Communications Specifications: a Comprehensive Study. *ACM Computing Surveys*, 33(4):427–469, December 2001.
- [7] X. Dèfago. *Agreement-Related Problems: From Semi Passive Replication to Totally Ordered Broadcast*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2000. PhD thesis no. 2229.
- [8] P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kouznetsov. Lightweight Probabilistic Broadcast. In *Proceedings of The International Conference on Dependable Systems and Networks (DSN 2001)*, pages 443–452, 2001.
- [9] P. Felber and F. Pedone. Probabilistic Atomic Broadcast. Technical Report HPL-2002-69, HP Labs, 2001.
- [10] A. Gopal and S. Toueg. Inconsistency and contamination. In *Proc. of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 257–272, Montreal, Canada, 1991.
- [11] M. Hayden. *The Ensemble System*. PhD thesis, Cornell University, Ithaca, NY, USA, 1998.
- [12] J. Hickey, N. Lynch, and R. van Renesse. Specifications and proofs for ensemble layers. In R. Cleaveland, editor, *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 1579, pages 119–133. Springer-Verlag, Berlin Germany, 1999.
- [13] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.
- [14] F. B. Schneider. Replication Management Using the State Machine Approach. In S. Mullender, editor, *Distributed Systems*. ACM Press - Addison Wesley, 1993.
- [15] U. G. Wilhelm and A. Schiper. A hierarchy of totally ordered multicasts. In *Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems (SRDS-14)*, Bad Neuenahr, Germany, Sept. 1995.

## A. Proofs

### Lemma 1. $SUTO \Rightarrow WUTO$

*Proof.* By contradiction. Let  $r$  be a run such that  $r \in R_{SUTO} \wedge r \notin R_{WUTO}$ . In order to violate WUTO, there must exist two processes  $p, q$  and two messages  $m, m'$  such that  $p$  delivers  $m$  before  $m'$  in  $r$  while  $q$  delivers  $m'$  before  $m$  in  $r$ . However, in order to satisfy SUTO, if  $p$  delivers  $m$  before  $m'$  then  $q$  has to deliver  $m'$  only after it has delivered  $m$ . Therefore  $r$  does not satisfy SUTO, which is a contradiction.  $\square$

### Lemma 2. $R_{SNUTO} \cap R_{UA} \equiv R_{WNUTO} \cap R_{UA}$

*Proof.*

1. ( $\subseteq$ ) (By contradiction) Let  $r$  be a run such that  $r$  in  $R_{SNUTO} \cap R_{UA} \wedge r \notin R_{WNUTO} \cap R_{UA}$ . Since  $r \in R_{SNUTO} \cap R_{UA}$ ,  $r$  must satisfy UA. As a consequence, it violates WNUTO, i.e. there exists two correct processes  $p, q$  and two messages  $m, m'$  such that  $p$  delivers  $m$  before  $m'$  in  $r$  while  $q$  delivers  $m'$  before  $m$  in  $r$ . However, in order to satisfy SNUTO, if  $p$  delivers  $m$  before  $m'$  then  $q$  has to deliver  $m'$  only after it has delivered  $m$ . Therefore  $r$  does not satisfy SNUTO, which is a contradiction.
2. ( $\supseteq$ ) (By contradiction) Let  $r$  be a run such that  $r$  in  $R_{WNUTO} \cap R_{UA} \wedge r \notin R_{SNUTO} \cap R_{UA}$ . Since  $r \in R_{WNUTO} \cap R_{UA}$ , it must satisfy UA. As a consequence, it violates SNUTO. Suppose that  $p$  is a correct process and  $m, m'$  are two messages and  $p$  delivers  $m$  before  $m'$  in  $r$ . Suppose also that  $q$  is a correct process in  $r$ . Since  $r$  violates SNUTO, we can have the following two cases:
  - (a) ( $q$  delivers  $m'$  without delivering  $m$ ) This contradicts the hypothesis that  $r$  satisfies UA.
  - (b) ( $q$  delivers  $m$  after delivering  $m'$ ) This contradicts the hypothesis that  $r$  satisfies WNUTO.

$\square$