Extending noninterference properties to the timed world

Jian Huang
Oxford University, Computing Laboratory
Wolfson Building, Parks Road,
Oxford, OX1 3QD, UK
jian.huang @ comlab.ox.ac.uk

A.W.Roscoe
Oxford University, Computing Laboratory
Wolfson Building, Parks Road,
Oxford, OX1 3QD, UK
bill.roscoe@ comlab.ox.ac.uk

ABSTRACT

Most previous work on information flow in process algebras has been based on untimed models of concurrency. It is obvious, however, that an observer might well use time to gain information about what a high-level user of the system is doing. We use the priority *tock* view (a discrete timed model) to extend several traditional untimed noninterference properties to the timed world. These are the determinism-based conditions of [14], [15] and [17], and Forster's local noninterference properties [6], [7].

Keywords

Information flow, noninterference, discrete time, CSP

1. INTRODUCTION

Noninterference properties are concerned with the security of systems. Given a process that has multiple users interacting with it, we are interested in preventing unspecified information flow between users. The definition of noninterference originated with Goguen and Meseguer in [8] and later in [9].

Their work stimulated much research, for example [20] and [21]. Determinism-based conditions were introduced in [14] and [18], which argued that the determinism of the low-level user's viewpoint guarantees the security of the system. Focardi and Gorrieri's primary condition, *bisimulation nondeductibility on compositions*(BNDC), was introduced in [1], [2] and [3], which requires that a system P is secure iff $\forall \Pi$ where $\Sigma\Pi \subseteq H \bullet P\backslash H \approx_B (P \quad || \quad \Pi)\backslash H$. Forster proposed the idea of local noninterference (LNI) in [6]. This introduced a family of security conditions built on the idea that low-level users should not be able to tell the difference between system states linked by a high-level action and pointed out problems with Focardi and Gorrieri's conditions.

The purpose of this paper is to extend the determinism and LNI definitions to the timed world. There has been a huge amount of work on different flavours of noninterference besides that cited here. However the only other noninterference work on *timed* process algebras that we are aware of is [5]. It is for that reason that we use that paper as our point of comparison.

We assume familiarity with the basic concepts of CSP. Readers without this can refer to [15] (available on the web).

1.1 Timed versions of CSP

The best known version of CSP [16, 15] does not have an explicit concept of time, merely the relative order of events. However we sometimes need to apply its ideas to systems where timing is important, either for the correct internal operation or, to meet external constraints. There are a number of choices of timed extensions to CSP; we need decide which to follow. The original is the continuous Timed CSP: a detailed analysis can be found in [13]. That gives a model in which all events are given an arbitrary real-number time. We believe that it will be possible to extend concepts of noninterference to this model, in particular that based on nondeterminism, though as discussed in Conclusions this is actually a little paradoxical.

That version, *Timed CSP*, is, however, significantly more complex than the untimed ones. For example, it presents a much greater challenge in trying to verify it automatically. Therefore in this paper we will concentrate mainly on discrete models.

There have been several discrete time dialects of CSP, which are all variations on the theme of using a special event *tock* to represent the regular passage of time. In [15], a very general version was presented where *tock* was essentially allowed into the language like every other event (subject to *tock* not being renamed or hidden). This proved to need a different operational model than standard CSP, because the principle of *maximal progress* required that τ events be given priority over *tock*. (This principle is used in all timed variants of CSP, as it is in other timed calculi.)

Ouaknine and Worrell, in [10, 12], showed that a significant subset of (continuous) Timed CSP can be related to a language for discrete timed CSP in which time (i.e. *tock*) is only introduced via a timeout operator. In other words properties can be verified automatically of continuous Timed CSP by model checking processes in this discrete model.

That variant, like Timed CSP, has the property that whenever a visible event has been continuously available up to a time (or *tock* in the discrete variant), but not happened, then the same event is possible at that time (or instantly after the *tock*). We will discuss this in the Conclusions.

Ouaknine and Lowe [11] have recently produced a slightly different variant, though more restrictive since it forbids *sig-*

nal events (see later), together with an abstract semantic model that is fully abstract for it. It is clear that this model can be adapted to encompass all the variants of discrete timed CSP. Both of these latter variants can be viewed as sub-languages of the discrete dialect from [15].

Since this hierarchy of discrete timed models has not fully settled down, in this paper we consider timed systems from a mainly operational viewpoint which contains them all, specifically LTS's with *tock* transitions as well as τ and ordinary visible ones.

1.2 Noninterference

We are given a timed system S, which has two users interacting with it, a high-level one and a low-level one. We want to ensure that no information can flow from high level to low level through their use and observation of S. It is assumed that the complete visible alphabet of S is $\Sigma^t = \Sigma \cup \{tock\}$ and Σ can be further partitioned into disjoint subsets H and L. The high-level user observe the events set $H^t = H \cup \{tock\}$ and can control those in H which are not *signals* (see below). The low-level one can similarly observe $L^t = L \cup \{tock\}$ and control the non-signals in L.

We should also be aware that though neither user can control the passage of time, the way some timed effects on other events are modelled sometimes make it look as if *tock* can be refused — as discussed in Section 3.1. Obviously we have to ensure that no information flow can appear from this.

2. INTRODUCTION TO DISCRETE TIME CSP

Delayable and signal events

The visible events in the timed world fall into two categories:

- Delayable: the environment can delay one of these by refusing to agree to it and letting *tock* pass.
- Signal: these can be observed but not delayed by the environment.

Frequently, for example, the process outputting on a channel will treat it as signals, whereas the process it inputs to will wait for its inputs and so treat the channel as delayable. Once these two processes are synchronised the event will be a signal to outside observers.

The notions of delayable and signal events have frequently been discussed in the untimed world, for example in Chapter 12 of [15]. In many untimed applications it is not necessary to distinguish between these two sorts of event. The presence of *tock* make it vital to do so.

In mapping LTS's to sets of failures we treat signals exactly as we treat the termination signal ✓ in untimed CSP, namely:

$$t\hat{\ }\langle s\rangle \in traces[\![P]\!] \Rightarrow (t, \Sigma \setminus \{s\}) \in failures[\![P]\!]$$

(In this paper we do not consider processes that terminate, so ✓ is not in our set of events.)

Time consistency

The *tock* event can be used to set out the timed details of a process. But not all processes that can be defined in this way make sense. Parallel timed processes always synchronise on *tock* so that their timing constraints are aligned. But sometimes, as discussed in [15], these constraints are inconsistent and generate a *time stop*.

The following check can discover whether the timed behaviour of a process is consistent: we will always assume that systems we consider satisfy this property, since otherwise they are not considered to be descriptions of real timed processes.

Definition 1. We are given a timed process P with alphabet $\Sigma^t = \Sigma \cup \{tock\}$ whose non-tock alphabet Σ can be further partitioned into delayable events set D and signal events set S. It satisfies the Time Consistency Check(TCC) iff, in the failures/divergences model:

$$TOCKS \sqsubseteq (P \underset{D\cup\{tock\}}{\|} CHAOST(D))\backslash\Sigma$$

where $CHAOST(D)$ is the most nondeterministic non-divergent timed process using events from D. We will show how to define $CHAOST(D)$ later, and $TOCKS = tock \rightarrow TOCKS$ is the process that just lets time pass. There are only two ways this refinement can fail: a deadlock corresponds to a time stop; divergence corresponds to P performing infinitely many events in a finite time, which we also consider to be an error.

Priority

It would be ideal if we could treat *tock* like any other event in the operational semantics of CSP (see Chapter 7 of [15] for full details of this). Yet it can be shown that this conflicts with the very nature of time passing and it is often necessary to treat *tock* in a special way. More detailed discussions on priority in the *tock* timed model can be found in [15], [10]. In the following, we start from the transition system produced by the standard operational semantics, and see that it is necessary to discard some transitions.

Consider a state which can both accept an h event and a *tock* transition. What will happen if one hides that h event? It is clear that h is changed to τ by hiding. The problem is: should the *tock* transition still be possible? For the reasons set out below, we are forced to conclude that the answer is **no**.

In untimed CSP, a τ transition is invisible and must happen at once unless some other action occurs instantly. The problem here is that unlike other visible events, it is likely that *tock* cannot happen soon enough to be available. If we want to have a consistent and uniform interpretation of how a node with both a τ and a *tock* available behaves then the *tock* can not occur — otherwise a succession of this would lead to τ being indefinitely delayed. Therefore the *maximal progress* assumption that no *tock* ever happens when τ is available, which is common across a variety of timed models, makes sense. To ensure that this assumption does not exclude any behaviours that are really possible, we will postulate that any event which happens at the same instant as a *tock* follows it in a trace: anything preceding it happens some positive time before it.

Recall the definition of a signal event. Just as with τ, in the timed world all signal events must happen immediately (before a *tock* can happen), and if a state has a signal event transition, it cannot permit a *tock* transition either. Similarly, if a state has both a signal event and a delayable event, unless the delayable event happens at once, it will never get opportunity to be executed because the signal one is sure to happen immediately.

Since higher priority should be assigned to all signal events and τs, some *tock* transitions that can be inferred from the standard operational semantics for CSP must be prohibited. In order better to understand the real possible transitions and study the timed details and properties of these processes, all impossible *tock* transitions should be removed from the original *LTS*s by inspecting that transition system.

In the rest of this paper, we assume that all impossible *tock* transitions have been removed and all non-*tock* events are delayable unless defined to be signals.

Looking at a timed system in this prioritised way gives a (possibly strict) refinement of the way one would interpret the same process definition over the untimed models of CSP, thinking of *tock* as a normal visible event. This is because all the choices which are forced by the priority view are ones an untimed process would be free to make in the untimed view.

To simulate a system running under these new rules we have to give internal and signal events priority over *tock*. (Other events are not affected by this, which means delayable events have no relative priority with others.) In terms of operational semantics, what one needs to do is to describe a new transition rule $\xrightarrow{x}_T$ in terms of the old one $\xrightarrow{x}$: here T is a set of time-like actions (*tock*).

Unless $x \in T$, $P \xrightarrow{x}_T Q \Leftrightarrow P \xrightarrow{x} Q$
and if $x \in T$ then $P \xrightarrow{x}_T Q \Leftrightarrow P \xrightarrow{x} Q \wedge \neg \exists\, Q' \bullet P \xrightarrow{\tau} Q'$

If there is a set of signal events S,
$P \xrightarrow{x}_T Q \Leftrightarrow P \xrightarrow{x} Q \wedge \neg \exists\, Q' \bullet (P \xrightarrow{\tau} Q' \vee P \xrightarrow{s} Q')$ where $s \in S$.

The traces, failures etc of any process can therefore be calculated by extracting them from this reduced transition system.

3. DETERMINISM-BASED CONDITIONS

Determinism-based noninterference conditions originated in [17] and [14], and are based on the idea that if the view of a low-level user is a deterministic process, nothing that a high-level user does can affect what he sees. The following is a natural extension of the untimed definition.

Definition 2. A timed process P is timed deterministic if and only if it is non-divergent and
$\forall\, s \in (\Sigma^t)^*, a \in \Sigma^t \bullet \neg((s, \{a\}) \in \mathit{failures}[\![P]\!] \wedge s^\frown\langle a\rangle \in \mathit{traces}[\![P]\!])$

T- L-Independence

The following is a natural noninterference condition: it states that nothing that a high-level user can do affects, either explicitly or implicitly, what a low-level one sees. Its untimed analogue may be found in [14].

Definition 3. L is *T-L-Independent* in process P, written *T-L-Ind*(P), if and only if it is non-divergent and

$(s_1, \{a\}) \in \mathit{failures}[\![P]\!] \Leftrightarrow (P/s_2)^0 \cap \{a\} = \emptyset$
$\forall\, a \in L^t, s_1, s_2 \in \mathit{traces}[\![P]\!]$ with $s_1 \upharpoonright L^t = s_2 \upharpoonright L^t$

3.1 Timed abstractions

An abstraction operator hides or removes some computational details of a process. In a security context, an abstraction of one process might be the low-level or high-level user's viewpoint of that process. Abstractions have been used to give a more process-oriented view, in the untimed world, of the sort of noninterference property captured in the previous section.

Several abstractions have been developed in the untimed world for this and other purposes, see [15]. The primary sorts have been the *lazy* and *mixed* abstractions.

3.1.1 Lazy Abstraction

The low-level view of the system should be consistent with all possible high-level behaviours. So the untimed *lazy abstraction* suggests that the most general low-level view (over the stable failures model $\mathcal{F}$) is to put P in parallel with the most non-deterministic and most general process that the high-level user can behave like, and then hide all high-level events. The most non-deterministic divergence-free high-level process in the untimed world of failures is

$$\mathit{Chaos}_H = \mathit{Stop} \sqcap ?x : H \rightarrow \mathit{Chaos}_H$$

which may both accept and refuse any high-level event after any trace at all.

We can then argue that, however the high-level user does behave, it must refine this abstraction.

Definition 4. Given a process P, its untimed lazy abstraction (over $\mathcal{F}$) is
$\mathcal{L}_H^{\mathcal{F}}(P) \triangleq (P \underset{H}{\parallel} \mathit{Chaos}_H)\backslash H$

As argued in [15], we take this stable failures version as definitive, and simply specify that the abstraction never diverges. The corresponding security condition requires this abstraction to be deterministic: namely there is no trace after which it can both accept and refuse one of the events (the low-level ones) that remain visible.

Definition 5. Given an untimed process P, it is lazily secure if $\mathcal{L}_H(P)$ is deterministic.

The lazy security condition works well in the untimed world, with many attractive properties. Yet, as formulated above, it turns out that it is inappropriate for the timed world.

Example 1. Let $H = \{d\}$ and $L = \{l\}$. Consider the following process P:

$$\begin{array}{lcl} P & = & \mathit{tock} \rightarrow Q \;\Box\; d \rightarrow \mathit{TOCKS} \\ Q & = & \mathit{tock} \rightarrow Q \;\Box\; d \rightarrow l \rightarrow \mathit{TOCKS} \end{array}$$

P is not secure since when the l event is present, the low-level user can deduce that a d has been performed. Yet the untimed lazy abstraction of P over $\mathcal{F}$ is TOCKS, which is timed deterministic. The problem is that if the *tock* transition from P to Q is allowed then the STOP branch of *Chaos* is blocking the d event from Q and does not change state.

Rather than use the untimed process CHAOS_H we need to use a timed one that can change its mind when time passes. A good way to define this is the following, where *tn* is a new event:

$$\begin{array}{lcl} \mathit{CHAOST}(D) & = & \mathit{CHAOST}'(D)\backslash\{\mathit{tn}\} \\ \mathit{CHAOST}'(D) & = & ?x : D \rightarrow \mathit{CHAOST}'(D) \\ & & \Box\;\; \mathit{tn} \rightarrow \mathit{tock} \rightarrow \mathit{CHAOST}'(D) \end{array}$$

This offers events in H as alternatives to τ, not *tock*, and so the members of H do not preempt *tock* since they are never possible in the same operational state. Furthermore this version changes state each time a *tock* occurs so does not cause the problem seen in the example above.

Definition 6. Given a process P, its timed lazy abstraction is

$$\mathcal{L}_{tH}(P) \triangleq (P \underset{H^t}{\parallel} CHAOST_H)\backslash H$$

Definition 7. Given a process P, it is timed lazily secure iff $\mathcal{L}_{tH}(P)$ is timed deterministic.

3.1.2 *Mixed abstraction*

The lazy security condition assumes that all high-level events can be delayed, yet sometimes this is not the case. If the event is not an action by the high-level user, but rather some feedback, it may be reasonable to assume that this event cannot be delayed or refused. For example, if the event represents some output to the high-level user's VDU or an alarm, it may not be possible to prevented or delay it.

If the set of high-level events can be divided into two parts, the delayable events D and the signal events S, the following hybrid condition has been used for untimed CSP (see [15] for more detail and examples).

Definition 8. Let $H = D \cup S$ with $D \cap S = \emptyset$ where events in D are delayable and those in S are signals. Given a process P, its mixed abstraction is
$\mathcal{M}^{\mathcal{S}}_H(P) \triangleq \mathcal{L}_H(P\backslash S)$
It satisfies the mixed security condition if this abstraction is deterministic.

In the timed world, timed lazy abstraction shows how to manipulate delayable events. So its natural timed extension is: hide all high-level signal events and then apply the timed lazy security condition.

Definition 9. Let $H = D \cup S$ with $D \cap S = \emptyset$ where events in D are delayable and those in S are signal. Given a process P, its timed mixed abstraction is $\mathcal{M}^{\mathcal{S}}_{tH}(P) \triangleq \mathcal{L}_{tH}(P\backslash S)$
Given a process P, it satisfies the timed mixed security condition if $\mathcal{M}^{\mathcal{S}}_{tH}(P)$ is timed deterministic.

3.2 Further discussion

We have defined several timed security conditions in this section. These included *T-L-Ind* and timed lazy security. Just as in the untimed world, these two are equivalent: this, of course, gives us extra confidence in our conditions. It should be remarked that the proof of this is not simply a trivial extension of the untimed version, thanks to the influence of priority.

THEOREM 1. *Given a process P, $\mathcal{L}_{tH}(P)$ is timed deterministic $\Leftrightarrow$ T-L-Ind(P)*

It is an elementary consequence of our definitions that a process refining another that satisfies these two properties also satisfies them.

3.2.1 *Composability*

Whether a security condition is composable is important in practice. If the condition is composable, a complex system can be built by composing some smaller, simpler systems, so we may be able to guarantee the security of the complete system from that of its parts. Composability results for the untimed determinism-based conditions were established in [22], [19]. They extend naturally to the timed world.

THEOREM 2. *Suppose P and Q are processes with alphabets A and B, and each of P and Q is T-L-independent. Then so is $P\ {}_A\|_B\ Q$.*

3.2.2 *Separability*

In the untimed world, a process is separable if it is equivalent to a parallel composition of sub-processes with disjoint alphabets. In the timed world, a process is separable if it is equivalent to a parallel composition of sub-processes which only synchronise on *tock*. The fact that both processes, thanks to timed consistency, let time simply progress, mean that no information can flow via this synchronisation.

Definition 10. Let P be a process whose non-*tock* alphabet can be partitioned into disjoint subsets H and L. P is said to be *timed separable* with respect to $\{H, L\}$ if there are processes P_H and P_L with $TCC(P_H) \wedge TCC(P_L)$ and $\alpha P_H = H^t$, $\alpha P_L = L^t$ such that $P = P_H \underset{\{tock\}}{\parallel} P_L$.

As argued in [14], separability is, surprisingly, not sufficient to exclude information flow: mere equivalence to a process that is structurally secure might conceal insecurity.

Definition 11. Let P be a process whose non-tock alphabet can be partitioned into disjoint subset H and L. P is said to be *strongly timed separable* with respect to $\{H, L\}$ if there exist timed deterministic processes P_H and P_L with $TCC(P_H) \wedge TCC(P_L)$ and $\alpha P_H = H^t$, $\alpha P_L = L^t$ such that $P = P_H \underset{\{tock\}}{\parallel} P_L$.

THEOREM 3. *A process P is strongly timed separable with respect to the partition $\{H, L\}$ iff T-H-Ind$(P) \wedge$ T-L-Ind(P)*

4. CONDITIONS ALLOWING NONDETERMINISM

There is no doubt that, within the behaviours observable in the respective models, the determinism-based conditions do ban any information flow. It is also true to say that they deem many actually secure processes to be insecure. As discussed extensively in the literature on the untimed properties, the root cause of this is the way CSP and other theories of concurrency identify different sorts, and causes of, nondeterminism.

We can only allow low to see nondeterminism if we know that none of that is influenced by high, either in a way that we can demonstrate explicitly in our semantics, or via some subtle influence on the system that is not picked up by our model. An example of the latter is some timing effect below the resolution of our clock.

4.1 Timed Local Noninterference

In [6], Forster presented the local noninterference conditions (LNI). They require a secure system to guarantee that a low-level user can not tell the difference between system states linked by a high-level action. All of this was on the assumption that all nondeterminism which is apparently possible in the LTS representing the operational semantics of a process is genuinely possible on every run of the process. In this regard we have to abandon the concept of refinement: since the possibility of nondeterminism is an important part of how a process now functions, we cannot throw it away.

By considering two equivalences: failures/divergences equivalence and weak bisimulation, and taking into account the non-determinism that arises from identically labelled high-level transitions which lead from the same state, Forster introduced four kinds of local noninterference conditions and they were also proven to be equivalent and agree with the untimed determinism-based conditions under certain circumstances. See [6] or [7] for details.

Definition 12. [1] A relation $R \subseteq Proc^t \times Proc^t \in \approx_t$ (is a weak bisimulation) iff for every $(p, q) \in R$, $\forall\, a \in \Sigma^{t,\tau}$,

- if $p \overset{a}{\rightarrow} p'$, $\exists\, q' \bullet q \overset{a}{\Longrightarrow} q'$ and $(p', q') \in R$
- if $q \overset{a}{\rightarrow} q'$, $\exists\, p' \bullet p \overset{a}{\Longrightarrow} p'$ and $(p', q') \in R$

Definition 13. A process P with H-labelled events restricted, written P/H, has the *LTS* of P with all H-labelled transitions removed. It is equivalent to $P \underset{H}{\parallel} Stop$. Note that this is just the CSP analogue of the CCS restriction operator $P \setminus H$.

Definition 14. A timed process P whose non-*tock* alphabet is partitioned by H and L satisfies timed strong local noninterference, written $tSLNI_L(P)$, if $\sigma_1 \xrightarrow{h} \sigma_2$ where $\sigma_1, \sigma_2 \in states[\![P]\!]$ and $h \in H$, implies $\sigma_1/H \approx_t \sigma_2/H$.

Definition 15. A timed process P whose non-*tock* alphabet is partitioned by H and L satisfies timed local noninterference, written $tLNI_L(P)$, if $\sigma \xrightarrow{h} \sigma_1, \ldots, \sigma \xrightarrow{h} \sigma_n$ where $h \in H$, $\sigma, \sigma_1, \ldots, \sigma_n \in states[\![P]\!]$, $n > 0$ and the states σ_i are a complete enumeration of the h-reachable states from σ, implies $\sigma/H \approx_t \sqcap\, \sigma_i/H$.

Definition 16. A timed process P whose non-*tock* alphabet is partitioned by H and L satisfies timed strong FD local noninterference, written $tSLNI_L^{FD}(P)$, if $\sigma_1 \xrightarrow{h} \sigma_2$ where $\sigma_1, \sigma_2 \in states[\![P]\!]$ and $h \in H$, implies $\sigma_1/H =_{FD} \sigma_2/H$.

Definition 17. A timed process P whose non-*tock* alphabet is partitioned by H and L satisfies timed FD local noninterference, written $tLNI_L^{FD}(P)$, if $\sigma \xrightarrow{h} \sigma_1, \ldots, \sigma \xrightarrow{h} \sigma_n$ where $h \in H$, $\sigma, \sigma_1, \ldots, \sigma_n \in states[\![P]\!]$, $n > 0$ and the states σ_i are a complete enumeration of the h-reachable states from σ, implies $\sigma/H =_{FD} \sqcap\, \sigma_i/H$.

It can be demonstrated that these conditions collapse under certain conditions.

THEOREM 4. *Let P be a non-divergent process. If P/H is timed deterministic,*

$$tSLNI_L(P) \Leftrightarrow tLNI_L(P) \Leftrightarrow tSLNI_L^{FD}(P) \Leftrightarrow tLNI_L^{FD}(P)$$

and furthermore each of these conditions is then equivalent to T-L-$Ind(P)$.

4.2 Timed Delayable Local Noninterference

The timed local noninterference conditions are too strict for processes with signal events since a signal event, which is neither delayable nor refusable by the high-level user, is like a τ in the low-level user's viewpoint. It is possible to loosen this requirement.

[1]This definition is the same as Focardi's *timed weak bisimulation* which is introduced in [5].

Example 2. Given $S = \{s_1, s_2\}$, $L = \{l_1, l_2\}$. Consider the following processes

$$\begin{array}{lcl} P & = & s_1 \rightarrow tock \rightarrow l_1 \rightarrow P \\ Q & = & s_1 \rightarrow tock \rightarrow l_1 \rightarrow Q \;\square\; s_2 \rightarrow tock \rightarrow l_1 \rightarrow Q \\ R & = & s_1 \rightarrow tock \rightarrow l_1 \rightarrow R \;\square\; s_2 \rightarrow tock \rightarrow l_2 \rightarrow R \end{array}$$

For process P, although a low-level user who knows the system design can detect the occurrence of s_1, he can not use it to get any information leak from high-level since a high-level user has no other choice other than s_1 thus the occurrence of s_1 does not really show the action/inaction of the high-level user.

For process Q, although a low-level user knows that a high-level event is communicated he knows that the high-level process could not have influenced this. Thus this process should also be labelled as secure.

In process R it is apparently the case that the high-level user gets to choose between the signals, and that which signal happens then affects what low sees. However, we have already stated that signal events are not controllable by the user, merely observable. Recall that whenever a signal event is possible, the process can refuse anything else. Therefore R is actually equivalent to

$$R' = s_1 \rightarrow tock \rightarrow l_1 \rightarrow R' \sqcap s_2 \rightarrow tock \rightarrow l_2 \rightarrow R'$$

so the high-level user cannot actually influence what low sees. There is an interesting issue which arises from this example, since plainly low can see which way nondeterminism has been resolved towards high. What we are trying to capture is, however, just ways in which high's actions can affect what low sees, and plainly this is impossible in R'.

P, Q and R do not satisfy the timed local noninterference conditions defined in last section: we therefore need a mixed version of this condition.

Definition 18. Given a timed process P whose non-*tock* alphabet is partitioned by H and L. H can further be partitioned by signal events set S and delayable events set D.

- it satisfies timed delayable strong local noninterference,

  written $tDSLNI_L(P)$, if $tSLNI_L(P \backslash S)$ holds.
- it satisfies timed delayable local noninterference,

  written $tDLNI_L(P)$, if $tLNI_L(P \backslash S)$ holds.
- it satisfies timed delayable strong FD local noninterference,

  written $tDSLNI_L^{FD}(P)$, if $tSLNI_L^{FD}(P \backslash S)$ holds.
- it satisfies timed delayable FD local noninterference,

  written $tDLNI_L^{FD}(P)$, if $tLNI_L^{FD}(P \backslash S)$ holds.

The following is a corollary of Theorem 4.

Proposition 1. Let P be a non-divergent timed process and $(P \backslash S)/H$ is timed deterministic,
$tDSLNI_L(P) \Leftrightarrow tDLNI_L(P) \Leftrightarrow tDSLNI_L^{FD}(P) \Leftrightarrow tDLNI_L^{FD}(P)$
and furthermore each of these conditions is then equivalent to mixed abstraction security.

5. CASE STUDY

In this section, we will consider a small case study to show:

- A timed "implementation" of a secure untimed process may be insecure. This demonstrates the necessity of abandoning the concept of refinement when using LNI-based security.
- We can use the conditions developed in this paper to help design a secure timed implementation.

Example 3. Let there be two users and one file in a system. The high-level user can read the file and the low-level one can write it. So information can flow from low level to high level. To avoid access conflict, it is required that before both users can access the file, they must make requests and since the high-level user should read the latest data, the low-level user is permitted to make requests and write file between a high-level request and a high-level read action. So the system process is:

$$\begin{aligned} System &= req_H \rightarrow System1 \\ &\quad \Box\ req_L \rightarrow write_L \rightarrow System \\ System1 &= read_H \rightarrow System \\ &\quad \Box\ req_L \rightarrow write_L \rightarrow System1 \end{aligned}$$

Here req_H and $read_H$ are high-level events while req_L and $write_L$ are low-level ones. This system is both *L-Ind* and *SLNI* secure.

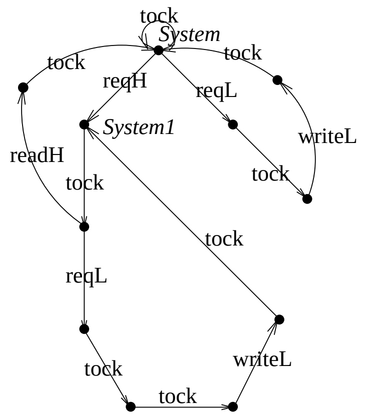

Figure 1: A timed implementation of Example 3

In the above specification, time is not modelled and all time details of the system are hidden. But in a real implementation, any communication is sure to spend some time, which adds time details in the system and brings possibilities for illegal information flow.

Example 4. Assume that when implementing that system described in Example 3, we need to fulfil the following requirement: all actions need one time unit; it requires an additional time unit to manage the low-level user's request if it follows a high-level one; the system can idle until a user makes a request.

$$\begin{aligned} System &= tock \rightarrow System\ \Box\ req_H \rightarrow System1 \\ &\quad \Box\ req_L \rightarrow tock \rightarrow write_L \rightarrow tock \rightarrow System \\ System1 &= tock \rightarrow \\ &\quad (read_H \rightarrow tock \rightarrow System \\ &\quad \Box\ req_L \rightarrow tock \rightarrow tock \rightarrow write_L \\ &\qquad \rightarrow tock \rightarrow System1) \end{aligned}$$

The *LTS* of this process is shown in Figure 1

This process is not secure for the following reasons

- The low-level user can notice the existence of two *tock* events between the req_L and $write_L$ after req_H.
- At the *System* state, the low-level user can communicate a req_L and at the *System*1 state, he can not.

A new "implementation" of Example 3 which is $tDSLNI_L$ secure is:

Example 5. The $read_H$ action is a system response to the req_H action, so we assume that $S = \{read_H\}$ and $D = \{req_H\}$. Figure 2 shows the *LTS* of the new timed "implementation" of *System* which also fulfils the requirement described in Example 4.

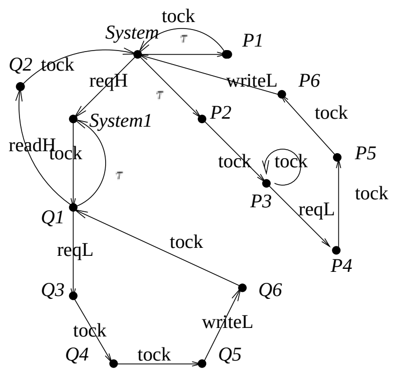

Figure 2: A secure timed implementation of Example 3

There is only one delayable action between *System* and *System*1. The following relationship R can be used to guarantee that $System\backslash S/H \approx_t System1\backslash S/H$.

$R = \{(X\backslash S/H, Y\backslash S/H) \mid (X, Y) \in R'\}$

where

$R' = \{(System, System1), (P_1, System1), (System, Q_1), (P_2, System1), (P_3, Q_1), (P_4, Q_3), (P_5, Q_4), (P_6, Q_5), (System, Q_6), (P_1, Q_6), (P_2, Q_6), (P_1, Q_2), (P_3, Q_2), (P_3, System), (P_3, P_1), (P_3, P_2), (P_3, P_3)\}$

6. RELATIONSHIP WITH *TBNDC, TBSNNI, TSBSNNI*

In [5], *tBNDC*, *tBSNNI* and *tSBSNNI* were introduced, which are the timed extensions of the conditions proposed in [1], [2] and [3]. In this section, we briefly discuss the relationship between those conditions and ours.

A timed language called *tSPA* was used to describe processes in [5]. Unlike our model it assumes *time determinacy*:

each state has at most one *tock* action, and the model does not have signals.

Rewriting some of the definitions and result from [5] in CSP gives:

Definition 19. A process E is directly weakly time alive iff $E \overset{tock}{\Longrightarrow}$. A process E is weakly time alive iff for all $E' \in states[\![E]\!]$, we have E' is directly weakly time alive.

In our language, this is similar to TCC in a world where all events are delayable.

Definition 20. Let $E \in Proc^t$. Then, $E \in tBNDC$ if and only if $\forall \Pi \in \mathcal{E}^t_H$ we have $(E \underset{H^t}{\|} \Pi)\backslash H \approx_t E\backslash H$, where $\mathcal{E}^t_H$ is the set of weakly timed alive processes that can perform only actions in $H^{\tau,t}$.

Definition 21. $E \in tBSNNI$ iff $E/H \approx_t E\backslash H$.

Definition 22. $E \in tSBSNNI$ iff $\forall E' \in state[\![E]\!] : E' \in tBSNNI$.

The relationship between these three conditions is:

Proposition 2. $tSBSNNI \subseteq tBNDC \subseteq tBSNNI$.

Just as in the untimed world, one can question whether these three conditions are really strong enough: the examples from [7] translate easily to the timed world. However there are natural relationships between these conditions and our own.

THEOREM 5. *Given a timed process* P, $tSLNI_L(P) \Rightarrow tSBSNNI(P)$.

We conjecture that $tBNDC$ is equivalent to timed lazy security when P is non-divergent and P/H is timed deterministic just as in untimed world. But further work is required to prove this.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have shown that noninterference conditions already investigated for untimed CSP transfer naturally to the *tock*-timed world and have the same interrelationships both with each other and which alternative properties, despite the complexities introduced by priority and maximal progress.

Following [7], we believe that the two styles of noninterference condition examined in this paper are safer than the style in which a process is declared to be secure if its composition with any high-level process produces equivalent results. These arguments apply unaltered in the case of timed systems. In any case our version extends the earlier work by incorporating signal events, a class that are important in timed systems. Our work was operationally based, thanks partly to the absence for the time being of an agreed denotational model for *tock*-timed CSP. Our decision was also pragmatic because the priority-based operational semantics we use is available in FDR, making most of our properties mechanically checkable with and existing tool.

We are confident that this work will transfer naturally to denotational models once these are settled, and also to continuous Timed CSP. An interesting question that arises here is that both Timed CSP and the discrete version of [12] there are very few deterministic processes. This is because of the feature described earlier that an available action cannot be withdrawn deterministically on a *tock* or the passage of time. Hence any process that can withdraw an event is bound to be formally nondeterministic: there is a moment when it can both accept and refuse this event. There are, however, refinement-maximal processes, namely ones that only have nondeterminism that cannot be removed because of this principle. It is certainly possible to re-constitute our security definitions by demanding that the appropriate abstractions are refinement maximal. We believe that this will make sense, but further research is needed on it and its possible relationships to operationally-based conditions like LNI – which will themselves presumably need to be rethought. This will also pose interesting philosophical questions about the meaning of nondeterminism and noninterference.

8. REFERENCES

[1] R.Focardi and R.Gorrieri. A classification of security properties(extended abstract). Technical Report UBLCS-93-21, October 1993.

[2] R.Focardi and R.Gorrieri. An information flow security property for CCS. *Proceedings of Second North American Process Algebra Workshop*, (NAPAW'93), August 1993.

[3] R.Focardi, R.Gorrieri. "A Classification of Security Properties". *Journal of Computer Security* 3:1 (1994/1995) 5-33.

[4] R. Focardi, R. Gorrieri and F. Martinelli. Real Time information Flow Analysis. IEEE JSAC 21(1):20-35, 2003.

[5] R. Focardi, R. Gorrieri, and F. Martinelli. "Information flow analysis in a discrete-time process algebra". *Proc CSFW XIII*, pages 170-184, 2000.

[6] R.Forster. "Noninterference Properties for Nondeterministic Systems". *D.Phil thesis, Oxford University*, 1999.

[7] R.Forster, G.M.Reed and A.W. Roscoe. " The Successes and Failures of Behavioural Models". *In Millennial Perspectives in Computer Science*, Palgrave 2000.

[8] J.A.Goguen, J.Meseguer. "Security Policies and Security Models". *IEEE Symposium on Security and Privacy*(1982), 11-20.

[9] J.A.Goguen, J.Meseguer. "Unwinding and Inference Control". *IEEE Symposium on Security and Privacy*(1984), 75-86.

[10] Joel Ouaknine. "Discrete Analysis of Continuous Behaviour in Real-Time Concurrent Systems". *D.Phil thesis, Oxford University*, 2000.

[11] Joel Ouaknine and G. Lowe. "On timed models and full abstraction", Proc MFPS 2005 (ENTCS).

[12] Joel Ouaknine and J.B. Worrell. "Timed CSP = closed time epsilon automata", Nordic Journal of Computing **10**, 2003.

[13] G.M.Reed, A.W.Roscoe. The timed failures-stability model for CSP. *Theoretical Computer Science*, 211:85-127, 1999

[14] A.W.Roscoe. "CSP and Determinism in Security Modelling". *1995 IEEE Symposium on Security and Privacy* (1995), 114-127.

[15] A.W. Roscoe. “The theory and practice of concurrency”, Prentice-Hall, 1998.

[16] C. A. R. Hoare. “Communicating Sequential Processes”. Prentice Hall, 1985.

[17] A.W.Roscoe, J.C.P.Woodcock, L.Wulf. “Noninterference through Determinism”. ESORICS 94, LNCS 875 Springer(1994), 33-55.

[18] A.W.Roscoe, J.P.C.Woodcock and L.Wulf. “Noninterference through determinism”. *Journal of Computer Security* 4, 1 27-54.

[19] A.W.Roscoe and L.Wulf. “Composing and decomposing systems under security properties”. *Proceedings of CSFW 1995* (IEEE Computer Society Press).

[20] D.Sutherland. “A Model of Information”. *Proceedings of the 9th National Computer Security Conference* (September 1986), 175-183.

[21] J.T.Wittbold, D.M.Johnson. “Information Flow in Non-deterministic Systems”. *Proceedings of the 1990 Symposium on Research in Security and Privacy*(1990), 144-161.

[22] L.Wulf, *Interaction and security in distributed computing.* Oxford University D.Phil thesis, 1997.

Acknowledgements

We would like to thank Gavin Lowe, Joel Ouaknine and Michael Goldsmith for useful discussions on this and related work. The second author’s work was partially supported by a grant from the US ONR.