Specification of Dynamic Networks

Radu Grosu* TU München

Ketil Stølen† OECD Halden Reactor Project

Abstract: A dynamic network is a network whose components may interact on channels established dynamically by the communication of their associated ports. This paper generalises a specification technique based on input/output-relations on streams to capture the special kind of privacy preservation found in such networks. A privacy preserving component never accesses, depends on or sends a port whose name it does not know. Composite specifications, describing networks of such components, are built from elementary specifications with three specially designed operators: one operator for static hiding, one for dynamic hiding, and one for parallel composition modulo many-to-many communication. The need for the three operators is motivated by a small example.

1 Introduction

The use of *input/output relations* (I/O-relations) to specify computerised components is well-known. For example, VDM [Jon90] and Z [Spi88] are both based on this approach: a specification of a *sequential* component C characterises the relationship between its *initial* and *final states*. The initial state can be understood as the input of C produced by C's environment before the execution of C is initiated. The final state can be understood as the output produced by C itself.

Reactive components can be specified in an analogous way. For example, Focus [BS97] is based on I/O-relations: a specification of a reactive component C characterises the relationship between its tuples of *input* and *output streams*. A tuple of input streams represents histories of input messages sent by C's environment along C's input channels. A tuple of output streams represents histories of output messages sent by C itself along C's output channels.

Recent advances in telecommunication and software technology have motivated the study of reactive components that may change their channel configurations dynamically. In this paper, such components and networks thereof are referred to as *dynamic*; traditional networks and their components are called *static*. Inspired by the π-calculus [MPW92], most theoretical work in this field has concentrated on process calculi; the emphasis has often been on semantic issues, and until recently, in a rather operational setting.

In this paper, dynamic channel configuration is studied in a denotational setting with an emphasis on specification issues. More explicitly, it is shown how I/O-relations on streams can be used to specify dynamic networks whose components communicate asynchronously via directed channels.

The rest of the paper is split into 7 sections: Section 2 explains how static components are modelled by I/O-relations represented by functions mapping stream tuples to sets of stream tuples; Section 3 generalises this approach to dynamic components and motivates the need for building privacy preservation into the semantics; Section 4 defines privacy preservation; Section 5 introduces typed channels; Section 6 presents a small example and motivates the need for the three operators defined in Section 7; Section 8 sums up the results and draws some conclusions.

*Institute für Informatik, TU München, D-80290, München, Germany; grosu@informatik.tu-muenchen.de

†OECD Halden Reactor Project, Institute for Energy Technology, P.O.Box 173, N-1751, Halden, Norway; Ketil.Stoelen@hrp.no

2 Static Components

A *static network* consists of a finite number of components communicating asynchronously via directed channels. As indicated by Figure 1, a component C of such a network can be thought of as a black-box that receives messages on input channels identified by a set of channel names I and sends messages along output channels identified by a set of channel names O.

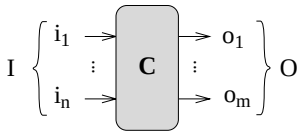

Figure 1: Static Component

The behaviour of C can be modelled by a function κ mapping input histories to sets of output histories:

$$\kappa \in (I \to H) \to \mathbb{P}(O \to H)$$

H is the set of channel histories, and $\mathbb{P}$ is the power-set operator. Each element $\alpha \in (I \to H)$ represents histories of messages sent by C's environment along the channels I; each element $\beta \in \kappa(\alpha)$ represents histories of messages sent by C itself along the channels O in response to α. For example, if $n \in I \cap O$ then $\alpha(n)$ is the history of all messages sent by C's environment along the channel n; $\beta(n)$ is the history of all messages sent by C itself along the channel n. Thus, in this model, the messages sent by the environment along n are not interleaved with the messages sent by C itself; as shown in Section 7.3, this interleaving first takes place when components are composed in parallel. This allows C to be described in a purely functional manner. Note that since κ yields a set of possible output histories, instead of just a single output history, C is not constrained to be deterministic.

In this paper, the channel histories are modelled by *timed infinite streams*. A timed infinite stream is an infinite sequence consisting of messages and *ticks*. Each tick denoted by $\surd$ represents the end of a time unit. Since time never halts, each timed infinite stream is required to have infinitely many ticks. Hence, H is the set of all timed infinite streams over the set of all messages; this means that κ is a function mapping stream tuples to sets of stream tuples. In the sequel such functions are called *interaction functions*.

3 Dynamic Components

A *dynamic network* differs from a static one in that its components may dynamically gain access to new channels by the communication of *channel ports*. The input port of a channel n is denoted by $?n$ and the output port by $!n$. Note that this is just a trick to simplify the presentation: a component in a dynamic network cannot exploit this convention to deduce the name of a port from the name of its complement port. The convention is overloaded to a set of channel names R in the obvious way: $?R \stackrel{\text{def}}{=} \{ ?r \mid r \in R\}$ and $!R \stackrel{\text{def}}{=} \{ !r \mid r \in R\}$. Moreover, $?!R \stackrel{\text{def}}{=} ?R \cup !R$. N is the set of all channel names. This means that the set of all ports P is characterised by:

$$P = ?!N = \{ ?n, !n \mid n \in N\}$$

It is assumed that any finite string of alphanumeric characters is contained in N; in standard UNIX manner: $s*$ denotes the set of all alphanumeric strings prefixed by s.

M is the set of all *pure messages*. D is the set of all pure messages and ports, from now on referred to as the set of all *messages*. It is assumed that $\surd \notin D$. For any set of messages U, $U^{\infty\!\!\!-}$ denotes the set of all timed infinite streams over U. The set of all channel histories H is therefore characterised by $D^{\infty\!\!\!-}$.

Also a dynamic component C has a set of static ports; these are the ports C knows initially; in the following $INIT$ denotes this set. Additionally, C may recursively gain access to new ports: C gains access to the ports received via C's initial input ports, via input ports received via C's initial input ports, via input ports received via input ports received via C's initial input ports, and so on. Since C in principle may receive any port in P via its initial input ports, and thereby gain send and receive access to any channel, the behaviour of C is modelled by an interaction function of the following signature:

$$\kappa \in (N \to H) \to \mathbb{P}(N \to H)$$

Note that both the input and the output histories are represented by functions defined for all channel names. This model is therefore too expressive: components that depend on or exploit ports that they have not gained access to in the recursive manner explained above can also be described. These components break the hiding invariant of dynamic networks and are therefore undesirable. In the terminology of this paper: they are not *privacy preserving*. The next section formalises the concept of privacy preservation.

The model is too expressive also in another respect (and so is the model of Section 2): interaction functions are not constrained to be *guarded*. An interaction function κ is *weakly (strongly) guarded* if its output at any point in time j depends only on input received until time j $(j-1)$. Formally, κ is weakly guarded if:

$$\alpha{\downarrow_j} = \beta{\downarrow_j} \Rightarrow \{\sigma{\downarrow_j} \mid \sigma \in \kappa(\alpha)\} = \{\sigma{\downarrow_j} \mid \sigma \in \kappa(\beta)\}$$

κ is strongly guarded if:

$$\alpha{\downarrow_j} = \beta{\downarrow_j} \Rightarrow \{\sigma{\downarrow_{j+1}} \mid \sigma \in \kappa(\alpha)\} = \{\sigma{\downarrow_{j+1}} \mid \sigma \in \kappa(\beta)\}$$

For any $\sigma \in (N \to H)$ and $j \in Nat$, $\sigma{\downarrow_j}$ denotes the result of truncating each stream in σ immediately after the jth tick[1].

Any computerised component is weakly guarded and, if the least time unit is chosen small enough, also strongly guarded. Hence, if the objective of this paper had been to come up with a model for computerised components only, then guardedness should have been imposed. However, the objective of this paper is to present a specification technique and its semantics. Since the purpose of imposing requirements in a specification is to exclude undesirable computerised components, and since no such effect is obtained by imposing guardedness, it seems that guardedness is not essential for specifications. Thus, given the stated objective, privacy preservation is important because it allows the kind of hiding required in large specifications to be captured; guardedness is not important because any real-life system is guarded.

4 Privacy Preservation

In order to give the formal characterisation of privacy preservation, a projection and a filtration operator on stream tuples are needed.

For any stream tuple $\alpha \in (N \to H)$ and set of channels $R \subseteq N$, the *projection* of α on R, written $\alpha|_R$, is defined for any $n \in N$ as follows:

$$(\alpha|_R)(n) \stackrel{\text{def}}{=} \begin{cases} \alpha(n) & \text{if } n \in R \\ \sqrt{}^{\infty} & \text{otherwise} \end{cases}$$

$\sqrt{}^{\infty}$ denotes the timed infinite stream consisting of $\sqrt{}$'s only. For any stream tuple $\alpha \in (N \to H)$ and set of messages $A \subseteq D$, the *filtration* of α with respect to A, written $A\circledS\alpha$, yields the stream tuple obtained from α by removing all occurrences of messages in $D \setminus A$.

[1] Weak guardedness is often called causality; if κ is deterministic then weak and strong guardedness correspond to non-expansiveness and contractivity with respect to the Baire metric [Eng77].

It is now easy to give a formula that holds for exactly those interaction functions that maintain the privacy invariant of dynamic networks. The function

$$\kappa \in (N \to H) \to \mathbb{P}(N \to H)$$

is *privacy preserving* with respect to the set of ports $INIT \subseteq P$ if

$$\begin{array}{lllll} (1) & & \kappa(\alpha) & = & \kappa(\alpha|_I) \\ (2) & \beta \in \kappa(\alpha) \;\Rightarrow & \beta & = & \beta|_O \\ (3) & \beta \in \kappa(\alpha) \;\Rightarrow & \beta & = & (M \cup ?I \cup !O) \circledS \beta \end{array}$$

where I and O are defined recursively, as follows:

$$\begin{array}{rcl@{\qquad}rcl} I_1 & \stackrel{\text{def}}{=} & \{i \mid ?i \in INIT\}, & O_1 & \stackrel{\text{def}}{=} & \{o \mid !o \in INIT\} \\ I_{j+1} & \stackrel{\text{def}}{=} & \{i \mid ?i \;\underline{in}\; (\alpha|_{I_j})\}, & O_{j+1} & \stackrel{\text{def}}{=} & \{o \mid !o \;\underline{in}\; (\alpha|_{I_j})\} \\ I & \stackrel{\text{def}}{=} & \cup_{j \in Nat}\, I_j, & O & \stackrel{\text{def}}{=} & \cup_{j \in Nat}\, O_j \end{array}$$

For any $p \in P$ and $\sigma \in (N \to H)$, $p \,\underline{in}\, \sigma$ holds if p occurs in σ. Informally speaking: (1) makes sure that κ does not depend on input ports it does not know; (2) makes sure that κ does not send messages via output ports it does not know; (3) makes sure that κ does not send ports it does not know. In the sequel, the decorated arrow $\stackrel{INIT}{\to}$ distinguishes interaction functions that are privacy preserving with respect to $INIT$ from those that are not.

5 Typed Channels

Channels can also be *typed*. T is the set of all channel types; N_T and P_T are the sets of all typed channels and typed ports, respectively. D_T and H_T are defined accordingly. Formally:

$$N_T \stackrel{\text{def}}{=} \{n :: t \mid n \in N \wedge t \in T\},\; P_T \stackrel{\text{def}}{=} \{p :: t \mid p \in P \wedge t \in T\},\; D_T \stackrel{\text{def}}{=} M \cup P_T,\; H_T \stackrel{\text{def}}{=} D_T^{\underline{\infty}}$$

A dynamic component with typed channels communicates typed ports; its behaviour is modelled by an interaction function of the following signature:

$$\kappa \in (N_T \to_{N_T} H_T) \to \mathbb{P}(N_T \to_{N_T} H_T)$$

The arrow $\to_{N_T}$ is used to state that only stream tuples, whose messages are type correct according to N_T, are considered. Formally:

$$N_T \to_{N_T} H_T \stackrel{\text{def}}{=} \{\alpha \in (N_T \to H_T) \mid \forall (n :: t) \in N_T : \alpha(n :: t) \in t^{\underline{\infty}}\}$$

The definition of privacy preservation carries over straightforwardly by interpreting the type as a part of the channel name: *typed* channels and *typed* ports are treated in exactly the same way as the definition in Section 4 treats channels and ports. Thus, in that case $INIT$ is a set of typed ports; I and O are sets of typed channels. From now on: when we refer to channels and ports we mean typed channels and typed ports.

6 Example

The objective of this section is to strengthen the reader's intuition and motivate the rest of the paper by specifying a simple dynamic network. The initial configuration of the network is characterised by Figure 2 if the dashed arrows are ignored. Note there is no direct connection

from A to B, but there is an indirect one via CENTRAL; from B to A there is no connection at all.

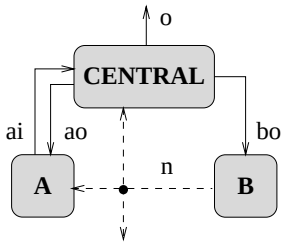

Figure 2: Dynamic Network

The network is supposed to dynamically generate direct communication links from B to both A and the environment. Each time A sends a connection request, represented by an arbitrary message, along the channel $ai : M$, the CENTRAL "creates" a new channel $n :: M$, whose name is taken from an infinite set of names $c*$; the corresponding input port $?n :: M$ is sent along $ao : P$ and $o : P$, and the corresponding output port $!n :: M$ is sent along $bo :: P$. B may now communicate directly with A via $n :: M$, as indicated by the dashed arrow from B to A. Moreover, the environment may receive what is sent by B along $n :: M$, since the input port $?n :: M$ has been sent along $o :: P$; this is why there is also a dashed arrow pointing outwards. On the other hand, the three components are not allowed to transmit $!n :: M$ to the environment; the environment may therefore not itself send messages along the channel $n :: M$ connecting B to A. CENTRAL may also receive what B sends along $n :: M$, but not itself send along $n :: M$.

The behaviour of CENTRAL is described by an elementary specification, as follows:

[CENTRAL]

$?ai, ?c* :: M$

$!ao, !o, !bo :: P$

$!c* :: M$

$\exists v \in (c*)^\infty :$

$\forall j, k \in Nat : j \neq k \Rightarrow v.j \neq v.k$

$\mathsf{let}\ l = \#\overline{\mathsf{inp}(ai :: M)}\ \mathsf{in}$

$\forall j \in [1\,..\,l] :$

$\overline{\mathsf{out}(ao :: P)}.j = \overline{\mathsf{out}(o :: P)}.j = ?(v.j) :: M$

$\overline{\mathsf{out}(bo :: P)}.j = !(v.j) :: M$

$\mathsf{out}(v.j :: M) = \surd^\infty$

$\#\overline{\mathsf{out}(o :: P)} = \#\overline{\mathsf{out}(bo :: P)} = \#\overline{\mathsf{out}(ao :: P)} = l$

The uppermost frame declares the ports that are known to CENTRAL initially with their respective types; whenever a set occurs on the left hand side, then each element of the set is assigned the type on the right-hand side.

In the lowermost frame, inp and out represent the input and output histories, respectively. $\overline{s}$ denotes the *untimed stream* obtained by removing all $\surd$'s in the stream s; informally speaking: the *time abstraction* of s. U^∞ denotes the set of all untimed infinite streams over U. The *length* operator # yields the length of a stream: $\#s$ is equal to the number of elements in s if s is finite and equal to ∞, otherwise. $s.j$ denotes the jth *element* of the stream s if $1 \leq j \leq \#s$. Line break without indentation represents conjunction; indentation captures scoping.

The existentially quantified variable v plays the role of an *oracle*. The first conjunct in the oracle's scope requires the oracle to be without repetitions. The let construct defines l to be

equal to the number of requests in $ai :: M$. The body of the let construct formalises the protocol described informally above. Note that its third conjunct makes sure that the CENTRAL does not itself send messages along the channels it generates dynamically for the two other components. Note also that there are no constraints on the elements of $c* :: M$ that are not "created" in response to a request from A.

The *semantic meaning* of this specification is the weakest interaction function that is privacy preserving with respect to the set of initial ports

$$\{?ai :: M, !ao :: P, !o :: P, !bo :: P\} \ \cup \ (?!c* :: M)$$

and whose input/output behaviours all satisfy the formula in the lower-most frame. That there is such a weakest function follows trivially from the definition of privacy preservation. For any specification S, $[\![S]\!]$ denotes its semantic meaning.

There is no need to give detailed specifications of A and B. A has $!ai :: M$ and $?ao :: P$ as initial ports and sends some (possibly infinite) number of connection requests along $ai :: M$. What A does with the information it receives from B is of no importance for this paper. B has only one port initially, namely $?bo :: P$; what B sends along the dynamically generated channels is left unspecified here.

Interesting is, however, the composite specification describing the network. It looks, as follows:

$$\langle c* :: M \rangle : [ao :: P, ai :: M, bo :: P] : \mathrm{A} \otimes (\langle c* :: M \rangle : \mathrm{CENTRAL}) \otimes \mathrm{B}$$

$[\,]$: is the operator for *static hiding*; in the composite specification above it ensures that the overall environment neither influences nor observes the communication along the channels $ao :: P$, $ai :: M$ and $bo :: P$ inside the network. The operator for static hiding is defined in Section 7.1.

$\langle\,\rangle$: is the operator for *dynamic hiding*. In the composite specification above it is used twice:

- to ensure that other components cannot observe what CENTRAL sends on the channels in $c* :: M$ unless CENTRAL itself delegates access to them by transmitting their associated ports;
- to ensure that what the overall environment sends along some channel $n \in (c* :: M)$ does not influence the communication between B and A (which actually means that the channel n on which the environment sends is different from the channel n used by B and A).

Strictly speaking, the overall effect of the network is the same if the inner-most occurrence of the dynamic hiding operator is removed. However, there is an internal effect: the channels $c* :: M$ are visible inside the network initially. This has no effect on the overall behaviour since A and B are unable to exploit this visibility: they do not have initial access to ports in $?!c* :: M$. The dynamic hiding operator is defined in Section 7.2.

$\otimes$ is the operator for parallel composition modulo many-to-many communication; it is formally defined in Section 7.3.

7 Operators

The three sub-sections below define the two hiding operators and the operator for parallel composition at the semantic level. These definitions are lifted to the syntactic level, as follows:

$$[\![[v] : S]\!] \stackrel{\mathrm{def}}{=} [v] : [\![S]\!], \quad [\![\langle v \rangle : S]\!] \stackrel{\mathrm{def}}{=} \langle v \rangle : [\![S]\!], \quad [\![S_1 \otimes S_2]\!] \stackrel{\mathrm{def}}{=} [\![S_1]\!] \otimes [\![S_2]\!]$$

7.1 Static Hiding

The operator for static hiding is *static* in the sense that the channels it hides remain hidden forever. Let

$$\kappa \in (N_T \rightarrow_{N_T} H_T) \stackrel{INIT}{\rightarrow} \mathbb{P}(N_T \rightarrow_{N_T} H_T)$$

and assume that V is a set of channels such that $?!V \subseteq INIT$. The *static hiding* of κ by V, written

$$[V] : \kappa$$

yields the interaction function such that for any $\alpha \in (N_T \rightarrow_{N_T} H_T)$

$$([V] : \kappa)(\alpha) \stackrel{\text{def}}{=} \{\widetilde{V}\circledS(\beta|_{\overline{V}}) \mid \beta \in \kappa(\widetilde{V}\circledS(\alpha|_{\overline{V}}))\}$$

where

$$\overline{V} \stackrel{\text{def}}{=} N_T \setminus V, \qquad \widetilde{V} \stackrel{\text{def}}{=} M \cup ?!\overline{V}$$

Since both α and β are projected on $\overline{V}$, it follows that $[V] : \kappa$ cannot exploit what the environment sends along V, and the other way around with respect to κ. That both α and β are filtered by $\widetilde{V}$ implies that $[V] : \kappa$ can neither gain nor give dynamic access to V by sending or receiving ports.

It can be shown that $[V] : \kappa$ is privacy preserving with respect to $INIT \backslash ?!V$.

7.2 Dynamic Hiding

The operator for dynamic hiding is *dynamic* in the sense that the channels it hides become visible if their associated ports are transmitted via the channels that are not hidden. Let

$$\kappa \in (N_T \rightarrow_{N_T} H_T) \stackrel{INIT}{\rightarrow} \mathbb{P}(N_T \rightarrow_{N_T} H_T)$$

and assume that V is a set of channels such that $?!V \subseteq INIT$. The *dynamic hiding* of κ by V, written

$$\langle V \rangle : \kappa$$

yields the interaction function such that for any $\alpha \in (N_T \rightarrow_{N_T} H_T)$

$$(\langle V \rangle : \kappa)(\alpha) \stackrel{\text{def}}{=} \{\beta|_R \mid \beta \in \kappa(\alpha|_D)\}$$

D and R are defined recursively, as follows:

$$\begin{array}{rcl@{\qquad}rcl}
D_1 & \stackrel{\text{def}}{=} & \{d \notin V \mid ?d \in INIT\} & R_1 & \stackrel{\text{def}}{=} & \{r \notin V \mid !r \in INIT\} \\
D_{j+1} & \stackrel{\text{def}}{=} & \{d \notin V \mid ?d \mathrel{\underline{in}} (\alpha|_{D_j})\} \cup & R_{j+1} & \stackrel{\text{def}}{=} & \{r \notin V \mid !r \mathrel{\underline{in}} (\alpha|_{D_j})\} \cup \\
 & & \{d \in V \mid !d \mathrel{\underline{in}} (\beta|_{R_j})\} & & & \{r \in V \mid ?r \mathrel{\underline{in}} (\beta|_{R_j})\} \\
D & \stackrel{\text{def}}{=} & \cup_{j \in Nat}\, D_j & R & \stackrel{\text{def}}{=} & \cup_{j \in Nat}\, R_j
\end{array}$$

Note the close relationship between this definition and the definition of privacy preservation in Section 4. The differences can be summed up as below:

- the ports in $?!V$ are hidden initially and therefore removed from the sets of initial ports (compare I_1, O_1 to D_1, R_1);
- a hidden input port $?n$ becomes visible, in the sense that $\langle V \rangle : \kappa$ will receive what the environment sends along this channel, if the complementary output port $!n$ is sent via a visible output port (compare I_{j+1} to D_{j+1}); this because the environment thereby gains send access to the corresponding channel;
- a hidden output port $!n$ becomes visible, in the sense that the environment may receive what $\langle V \rangle : \kappa$ sends along this channel, if the complementary input port $?n$ is output via a visible output port (compare O_{j+1} to R_{j+1}); this because the environment thereby gains receive access to the corresponding channel.

It can be shown that $\langle V \rangle : \kappa$ is privacy preserving with respect to $INIT$.

7.3 Parallel Composition

The parallel operator is defined for many-to-many communication. Given two interaction functions:

$$\kappa_1 \in (N_T \to_{N_T} H_T) \overset{INIT_1}{\to} \mathbb{P}(N_T \to_{N_T} H_T), \qquad \kappa_2 \in (N_T \to_{N_T} H_T) \overset{INIT_2}{\to} \mathbb{P}(N_T \to_{N_T} H_T)$$

Their *parallel composition* $\kappa_1 \otimes \kappa_2$ is illustrated graphically by Figure 3.

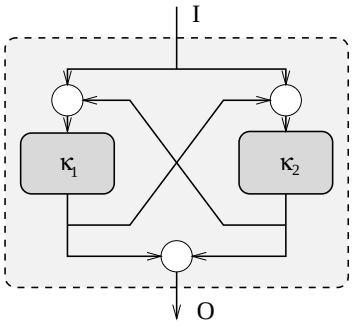

Figure 3: Parallel Composition

Formally, $\kappa_1 \otimes \kappa_2$ yields the interaction function such that for any $\alpha \in (N_T \to_{N_T} H_T)$

$$\begin{array}{lcll} (\kappa_1 \otimes \kappa_2)(\alpha) \overset{\text{def}}{=} \{ & (M \cup ?I \cup !O) \circledS (\beta|_O) & | & \beta \in \mathsf{MRG}(\beta_1, \beta_2) \\ & & & \beta_1 \in \kappa_1(\alpha_1) \\ & & & \beta_2 \in \kappa_2(\alpha_2) \\ & & & \alpha_1 \in \mathsf{MRG}(\alpha|_I, \beta_2) \\ & & & \alpha_2 \in \mathsf{MRG}(\alpha|_I, \beta_1) \quad \} \end{array}$$

I and O are defined in exactly the same way (with respect to typed channels and typed ports) as in the definition of privacy preservation in Section 4 with the exception that $INIT$ is replaced by $INIT_1 \cup INIT_2$.

$\mathsf{MRG}(\sigma_1, \sigma_2)$ denotes a component-wise, instantaneous and nondeterministic merge node; it is *component-wise* in the sense that $\sigma_1(n)$ is merged with $\sigma_2(n)$ for any channel n; *instantaneous* in the sense that MRG outputs any message or port in the same time-unit as it is received; *nondeterministic* in the sense that MRG returns the set of all stream tuples satisfying these two requirements. The three merge nodes in the definition of $\otimes$ merges the output of κ_1 with the output of κ_2, the output of the environment with the output of κ_2, and the output of the environment with the output of κ_1. MRG is formally defined, as follows:

$$\mathsf{MRG}(\sigma_1, \sigma_2) \overset{\text{def}}{=} \{\sigma \mid \forall n \in N_T : \forall j \in Nat : \sigma(n)\lceil_j \in \mathsf{merge}(\sigma_1(n)\lceil_j, \sigma_2(n)\lceil_j)\}$$

$s\lceil_j$ denotes the finite stream between the jth and the $(j+1)$st tick in the infinite timed stream s, and $\mathsf{merge}(r_1, r_2)$ denotes the set of finite streams obtained by merging the two finite streams r_1 and r_2; thus, $\mathsf{merge}(r_1, r_2)$ is equal to the set of all possible interleavings of r_1 and r_2.

From the filtering and projection operations built into the definition of $\otimes$, it follows trivially that $\kappa_1 \otimes \kappa_2$ is privacy preserving with respect to $INIT_1 \cup INIT_2$. This would not be the case if these operations were removed. This may surprise some readers since κ_1 and κ_2 are privacy preserving with respect to $INIT_1$ and $INIT_2$, respectively. In fact, this is the price to pay for not requiring the interaction functions to be strongly guarded.

To see this, consider the alternative definition without filtering and projection operations; assume there is a channel n such that $?n, !n \in INIT_1 \cap INIT_2$, and assume that $!r \notin INIT_1 \cup INIT_2$. Suppose that both κ_1 and κ_2 are such that if they receive $!r$ on the channel n then they will output $!r$ along the channel n in the same time unit as it was received. Because both functions may predict what the other function does in the same time-unit they may both decide to output $!r$ without having received this port from the overall environment. Thus, $\kappa_1 \otimes \kappa_2$ may output $!r$ without receiving it via the ports it recursively gains access to from $INIT_1 \cup INIT_2$; consequently, $\kappa_1 \otimes \kappa_2$ is not privacy preserving with respect to $INIT_1 \cup INIT_2$.

$\otimes$ is well-defined independent of whether the sets of dynamically hidden channels in the two component specifications are disjoint or not. However, unless the two sets of dynamically hidden channels are disjoint (which is syntactically decidable) there can be some less intuitive effects. To see that, let $\kappa_1 \stackrel{\text{def}}{=} \langle V_1 \rangle : \kappa_1'$, $\kappa_2 \stackrel{\text{def}}{=} \langle V_2 \rangle : \kappa_2'$, and assume that κ_1 sends the port $?n$, where $n \in V_1 \cap V_2$, to κ_2 along a common channel. Then, κ_1 does not receive what κ_2 sends on n unless κ_1 itself makes this channel visible by sending $!n$ along some visible channel. Thus, when specifications are composed in parallel, it may make sense to require that they have disjoint sets of dynamically hidden channels. This constraint can be avoided by building a renaming facility into the composition operator. However, the flexibility gained by implicit renaming is hardly worth the price of a more complex semantics.

8 Conclusions

This paper is concerned with the specification of dynamic channel configuration. In particular:

- it generalises specification techniques based on I/O-relations to deal with the kind of hiding (or encapsulation) found in dynamic networks;
- it supports the creation of direct communication links between components that are not directly connected initially; as shown in Section 6, these communication links can be in the opposite direction of the initial data-flow (there is no direct or indirect connection from B to A initially; nevertheless, direct connections from B to A are created dynamically).
- it supports hiding in the sense that privacy preservation is built into the semantics; this also simplifies elementary specifications since the hiding invariant does not have to be stated explicitly;
- it offers three operators for the construction of composite specifications from elementary ones.

This paper considers only many-to-many communication. However, the presented approach can be redefined for point-to-point.

This paper has evolved from [GS96a]. [GS96a] gives a denotational semantics for computerised components with respect to many-to-many communication based on sets of strongly guarded functions; [GS96b] does the same for point-to-point. These approaches are related to the work of Kok [Kok87, Kok89]. The major difference is that Kok does not deal with mobility. Moreover, Kok's handling of nondeterminism is different. [Kok89] employs a metric on relations and can basically handle only bounded nondeterminism. [Kok87] employs an automaton to generate the behaviours of basic agents. This guarantees the existence of fix-points also in the unbounded case. [GS96a] employs sets of strongly guarded ("deterministic") functions for the same purpose.

[Gro94, Bro95] give equational characterisations of dynamic channel configuration with respect to stream processing functions.

Several researchers have recently proposed denotational semantics for dynamic channel configuration in the context of the π-calculus (see for example [JJ95, Sta96]).

9 Acknowledgements

The authors would like to thank Manfred Broy for many inspiring discussions on this and related topics.

References

[Bro95] M. Broy. Equations for describing dynamic nets of communicating systems. In *Proc. 5th COMPASS Workshop, Lecture Notes in Computer Science 906*, pages 170–187, 1995.

[BS97] M. Broy and K. Stølen. Focus on system development. Book manuscript, January 1997.

[Eng77] R. Engelking. *General Topology.* PWN — Polish Scientific Publishers, 1977.

[Gro94] R. Grosu. *A Formal Foundation for Concurrent Object Oriented Programming.* PhD thesis, Technische Universität München, 1994. Also available as report TUM-I9444, Technische Universität München.

[GS96a] R. Grosu and K. Stølen. A denotational model for mobile many-to-many dataflow networks. Technical Report TUM-I9622, Technische Universität München, 1996.

[GS96b] R. Grosu and K. Stølen. A model for mobile point-to-point data-flow networks without channel sharing. In *Proc. AMAST'96, Lecture Notes in Computer Science 1101*, pages 504–519, 1996.

[JJ95] L. J. Jagadeesan and R. Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus. In *Proc. AMAST'95, Lecture Notes in Computer Science 936*, pages 277–291, 1995.

[Jon90] C. B. Jones. *Systematic Software Development Using VDM, Second Edition.* Prentice-Hall, 1990.

[Kok87] J. N. Kok. A fully abstract semantics for data flow nets. In *Proc. PARLE'87, Lecture Notes in Computer Science 259*, pages 351–368, 1987.

[Kok89] J. N. Kok. An iterative metric fully abstract semantics for nondeterministic dataflow. In *Proc. MFCS'89, Lecture Notes in Computer Science 379*, pages 321–331, 1989.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I. *Information and Computation*, 100:1–40, 1992.

[Spi88] J. M. Spivey. *Understanding Z, A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1988.

[Sta96] I. Stark. A fully abstract domain model for the π-calculus. In *Proc. LICS'96*, pages 36–42. IEEE Computer Society Press, 1996.