Describing Secure Interfaces with Interface Automata

Matias Lee    Pedro R. D’Argenio

FaMAF - UNC
CONICET

FESCA Workshop
Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
   - Bisimulation-based (Strong) Non-deterministic Non-interference

2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm - Example

3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

4. Contribution and future works
Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
     - Composition
     - Bisimulation-based (Strong) Non-deterministic Non-interference

2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm - Example

3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

4. Contribution and future works
Interface Automata (IA):

We use Interface Automata [De Alfaro, Hezinger 2001,2005] to represent interfaces. E.g.:

IA has three different sorts of actions: *input*, *output* and *hidden*. As usual, input are suffixed by ? and output by !. We indicate hidden actions by suffixing ;.
Interface Structure for Security (ISS)

- Extends IA to cope with security.
- Visible actions are separated in two classes:
  - public or low: can be observed/manipulated by any user
  - private or high: only for users with appropriate clearance.

High actions are underlined.
Component Based Development and Design has become main approach for software development. Example: web services.

We need good interface description that allows us to analyze interaction between components. In this way, we can predict if the composed system can satisfy our requirements.

IA captures temporal aspects of the component interface. This framework requires that the communication is properly carried out by the interfaces.

ISS inherits the properties of IA and also allows us to study properties related with secure data flow.
Example:

A distributed transaction processing system (DTPS):

- a main server (*Transaction Service*) that provides a service
- a remote transaction process unit (*Trans. Processing Unit*)
- a supervisor module (*Supervisor*).
Matias Lee, Pedro R. D’Argenio

Interfaces Structure for Security

Deriving secure ISS
Preserving BSNNI after Composition
Contribution and future works

Interfaces Automata and Interface Structure For Security

Why IA and ISS?
Composition
Bisimulation-based (Strong) Non-deterministic Non-interference

Transaction Service

\[
S_1 \xrightarrow{\text{acceptT?}} S_2 \xrightarrow{\text{startT!}} S_3 \xrightarrow{\text{endT?}} S_4 \xrightarrow{\text{endM?}} S_5 \xrightarrow{\text{startM?}} S_6 \xrightarrow{\text{newT?}} S_7 \xrightarrow{\text{logM!}} \]

Trans. Processing Unit

\[
t_1 \xrightarrow{\text{startT?}} t_2 \xrightarrow{\text{nOk!}} t_4 \xrightarrow{\text{endT!}} t_3 \xrightarrow{\text{ok!}} \text{logF!} \]

Supervisor

\[
U_1 \xrightarrow{\text{logF?}} U_2 \xrightarrow{\text{mOn?}} U_3 \xrightarrow{\text{logM?}} U_4 \xrightarrow{\text{logF?}} U_5 \xrightarrow{\text{endM!}} \]

Transaction Service

\[
S_1 \xrightarrow{\text{newT?}} S_2 \xrightarrow{\text{startT!}} S_3 \xrightarrow{\text{endT?}} S_4 \xrightarrow{\text{endM?}} S_5 \xrightarrow{\text{startM?}} S_6 \xrightarrow{\text{newT?}} S_7 \xrightarrow{\text{logM!}} \]

Trans. Processing Unit

\[
t_1 \xrightarrow{\text{startT?}} t_2 \xrightarrow{\text{nOk!}} t_4 \xrightarrow{\text{endT!}} t_3 \xrightarrow{\text{ok!}} \text{logF!} \]

Supervisor

\[
U_1 \xrightarrow{\text{mOn?}} U_2 \xrightarrow{\text{startM!}} U_3 \xrightarrow{\text{logM?}} U_4 \xrightarrow{\text{logM?}} U_5 \xrightarrow{\text{endM!}} \]

Transaction Service

\[
S_1 \xrightarrow{\text{acceptT?}} S_2 \xrightarrow{\text{startT!}} S_3 \xrightarrow{\text{endT?}} S_4 \xrightarrow{\text{endM?}} S_5 \xrightarrow{\text{startM?}} S_6 \xrightarrow{\text{newT?}} S_7 \xrightarrow{\text{logM!}} \]

Trans. Processing Unit

\[
t_1 \xrightarrow{\text{startT?}} t_2 \xrightarrow{\text{nOk!}} t_4 \xrightarrow{\text{endT!}} t_3 \xrightarrow{\text{ok!}} \text{logF!} \]

Supervisor

\[
U_1 \xrightarrow{\text{logF?}} U_2 \xrightarrow{\text{mOn?}} U_3 \xrightarrow{\text{logM?}} U_4 \xrightarrow{\text{logF?}} U_5 \xrightarrow{\text{endM!}} \]
We are interested in studying how the components work together.

Therefore, we need a concept of composition.
Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
     - Bisimulation-based (Strong) Non-deterministic Non-interference

2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm - Example

3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

4. Contribution and future works

Matias Lee, Pedro R. D'Argenio

Interface Structure for Security
Composition

- CSP likes parallel composition in IA:
  - the state space is the product of the set of states of the components,
  - synchronization through shared action, i.e. both component should perform a transition with the same synchronizing label (one input, and the other output), and
  - transitions with non-shared actions are interleaved.

Besides, shared actions are hidden in the product.
Interfaces Structure for Security
Deriving secure ISS
Preserving BSNNI after Composition
Contribution and future works

Why IA and ISS?
Composition
Bisimulation-based (Strong) Non-deterministic Non-interference

Matias Lee, Pedro R. D’Argenio
Error, Incompatible and Compatible states

- In state $s_3 t_4 u_2$, the TP unit sends a message (LogF!) to the Supervisor, which is not ready to receive it. We call this *miscommunication*. The state $s_3 t_4 u_2$ is an error state.

- States $s_3 t_3 u_2$ and $s_2 t_1 u_2$ are incompatibles states because they reach an error/incompatible state autonomously (i.e. using only output and/or hidden actions).

- A state that is not incompatible is called *compatible*.

- For example, $s_1 t_1 u_2$ is compatible.

- If the initial state of the product is compatible, then the interfaces are compatible.
Interfaces Structure for Security
Deriving secure ISS
Preserving BSNNI after Composition
Contribution and future works

Interfaces Automata and Interface Structure For Security
Why IA and ISS?
Composition
Bisimulation-based (Strong) Non-deterministic Non-interference

\[
\begin{array}{c}
\text{s}_2 \text{t}_1 \text{u}_2 \xrightarrow{\text{startT};} \text{s}_3 \text{t}_3 \text{u}_2 \xrightarrow{\text{nOk}!} \text{s}_3 \text{t}_4 \text{u}_2 \xrightarrow{\text{logF}!} \\
\text{s}_1 \text{t}_1 \text{u}_1 \xrightarrow{\text{acceptT}?} \text{s}_1 \text{t}_1 \text{u}_2 \xrightarrow{\text{newT}?} \text{s}_4 \text{t}_1 \text{u}_3 \xrightarrow{\text{logM};}\xrightarrow{\text{newT}?} \\
\text{s}_3 \text{t}_3 \text{u}_1 \xrightarrow{\text{endT};} \xrightarrow{\text{newT}?} \text{s}_3 \text{t}_2 \text{u}_1 \xrightarrow{\text{startM};} \text{s}_7 \text{t}_1 \text{u}_3 \xrightarrow{\text{logM};} \\
\text{s}_4 \text{t}_1 \text{u}_5 \xrightarrow{\text{endT};} \xrightarrow{\text{startT};} \text{s}_6 \text{t}_3 \text{u}_3 \xrightarrow{\text{ok}!} \text{s}_5 \text{t}_1 \text{u}_3 \xrightarrow{\text{logF};} \\
\text{s}_6 \text{t}_3 \text{u}_4 \xrightarrow{\text{endT};} \xrightarrow{\text{logF};} \text{s}_6 \text{t}_4 \text{u}_3 \xrightarrow{\text{nOk}!} \\
\end{array}
\]
2nd Step: Avoid reaching incompatibles states

If a set of interfaces are compatible, reaching incompatibles states in the composition can be avoided by not allowing certain inputs.

In this way, we finally obtain the composition of the interface.
Interfaces Structure for Security
Deriving secure ISS
Preserving BSNNI after Composition
Contribution and future works

Matias Lee, Pedro R. D'Argenio

Interface Structure for Security
Matias Lee, Pedro R. D'Argenio  

Interfaces Structure for Security
Deriving secure ISS
Preserving BSNNI after Composition
Contribution and future works

Interfaces Automata and Interface Structure For Security
Why IA and ISS?
Composition
Bisimulation-based (Strong) Non-deterministic Non-interference
Interfaces Structure for Security

Deriving secure ISS

Preserving BSNNI after Composition

Contribution and future works

Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
   - Bisimulation-based (Strong) Non-deterministic Non-interference

2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm - Example

3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

4. Contribution and future works
Motivation

- In the previous example, low-level users should not be allowed to know whether they are under supervision.
- I.e. they should not distinguish the occurrence of high actions.
- Therefore, for a low-level user, the system should behave in the same way regardless whether high actions are performed or not.

⇒ *non-interference*.

In our setting, the concept of non-interference is formalized by *bisimulation-based strong non-deterministic non-interference* (BSNNI) and *bisimulation-based non-deterministic non-interference* (BNNI).
BSNNI and BNNI (Focardi, Gorrieri 2001)

- $S \approx S'$ denotes weak bisimulation between $S$ and $S'$.
- $S/X$ represents the hiding of actions $X$ in $S$.
- $S\setminus X$ represents the restriction of actions $X$ in $S$.

**Definition**

(i) $S$ is bisimulation-based strong non-deterministic non-interference (BSNNI) if $S\setminus A^h \approx S/A^h$.

(ii) $S$ is bisimulation-based non-deterministic non-interference (BNNI) if $S\setminus A^{I,h}/A^{O,h} \approx S/A^h$. 

Matias Lee, Pedro R. D'Argenio
Example: $\mathcal{S}$ is BSNNI

$\mathcal{S}$

$\mathcal{S}/A^h \approx \mathcal{S}\setminus A^h$

Matias Lee, Pedro R. D'Argenio
Every single ISS component of our example is BSNNI but...

... the composed system is not! :'(
Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
   - Bisimulation-based (Strong) Non-deterministic Non-interference

2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm - Example

3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

4. Contribution and future works
Algorithm for Checking Bisimulation

A variation of Fernandez and Mounier’s algorithm to check bisimulation *on the fly*. Roughly, it works as follows:

- the IA are saturated adding all weak transitions
- a full synchronous product is constructed where transitions synchronize whenever they have the same label;
- whenever there is a mismatching transition, a new transition is added on the product leading to a special *fail* state;
- if reaching a fail state is inevitable (we later define this properly) the IA are not bisimilar; if there is always a way to avoid reaching a fail state, the IA are bisimilar.
Consider a simplified version of the composed DTPS

\[
\begin{align*}
S & \xrightarrow{acceptT?} S_1 \\
S_1 & \xrightarrow{mOn?} S_2 \\
S_2 & \xrightarrow{startM;} S_3 \\
S_3 & \xrightarrow{ok!} S_4 \\
S_4 & \xrightarrow{newT?} S_5 \\
S_5 & \xrightarrow{newT?} S_3
\end{align*}
\]
We first construct the restriction and hiding of the system.
Saturation marking set \( B \) (with \( B = \{mOn?\} \))

Actions in \( B \) will be replaced by \( \varepsilon' \), to record that these actions are high inputs actions that can be pruned. Other hidden actions are replaced by \( \varepsilon \).
Saturation adds to all state a self loop with \( \varepsilon \) and \( \varepsilon' \) (not depicted)
Actions added by the saturation are overlined.
Saturation marking set $B$ with $B = \{ \text{mOn}\}$

After the saturation we obtain this interface:

Note: In the next slides we will omit some actions added by the saturation process that are redundant.
Synchronous Product: $S \backslash A^h_\emptyset \times S / A^h_{A^h, I}$

The saturated interfaces (with some transitions omitted):

After saturating both interfaces, we can construct the synchronous product:

$S_1, S_1$
Synchronous Product: $S\setminus A^h_\emptyset \times S/A^h_{A^h,l}$

The product synchronizes using common actions:

- $S_1, S_1 \xrightarrow{\varepsilon'} S_1, S_2$
- $S_4, S_4 \xrightarrow{\text{newT?}}$
Synchronous Product: $\overline{S \setminus A^h_\emptyset} \times \overline{S / A^h_{A^h,l}}$

The process continue adding transitions and new states:

- $S_1 \rightarrow S_1$, $S_1 \rightarrow S_2$
- $S_1 \rightarrow S_2$
- $S_2 \rightarrow S_3$
Synchronous Product: $S\backslash A^h\emptyset \times S/A^h_{A^h,1}$

Notice that $s_1 \xrightarrow{accept?} s_1$ but $s_3 \xrightarrow{accept?} s_3$. 
Therefore, we add a transition to a special state $fail$.
Synchronous Product: $S \backslash A^h_\emptyset \times S / A^h_{A^h, I}$

- $s_1 s_3$ contains a pair of state that are not bisimilar.
- In this case, we say the state $s_1 s_3$ does not pass the bisimulation test.
- We let NoPass be the set of pair of states not passing the bisimulation test.
- The definition of NoPass is inductive. Under some restrictions, it propagates the condition “does not pass the bisimulation test” to predecessor states in the synchronous product.
Synchronous Product: $\overline{S\setminus A^h_\emptyset} \times \overline{S/A^h_{A^h,l}}$

- If the initial state does not pass the bisimulation test, the interfaces are not bisimilar.
- Otherwise, the interfaces are bisimilar, and then, the system is secure.
- In the example, the interfaces are not bisimilar and hence the system is not secure.
Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
   - Bisimulation-based (Strong) Non-deterministic Non-interference

2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm - Example

3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

4. Contribution and future works
If the system does not pass a the bisimulation test, we divide all the pairs of states of the synchronous product that does not pass the bisimulation test in 3 disjoint sets:

- **May State**: contains all pairs that may become bisimilar if some particular low input transition is not executed.
- **Fail State**: contains all pairs that cannot be turned into bisimilar by avoiding input transitions.
- **Undetermined state**: Contains all undetermined pairs. This is consequence that they may become bisimilar if a high input transitions is not executed.
May State Example:

The interface is not secure as a consequence of transition \( s_3 \xrightarrow{b?} s_4 \).
If this transition is forbidden/pruned (i.e., the interface provides fewer services), the resulting ISS is secure.
This is the same approach used to avoid miscommunication.
Fail state Example:

A similar example to the previous one, but now transition $b$ is an output action.

Then, the transition cannot be pruned (since it is not controllable), and hence the interface is not "recoverable".
Undetermined states (example 1):

\[ S \]

\[ S \backslash A^h_\emptyset \]

\[ S / A^h \{ h' \} \]

\[ S \backslash A^h_\emptyset \times S / A^h \{ h' \} \]

\[ S \text{ is not secure.} \]

The only option to recover is the elimination of transition \( s_2 \xrightarrow{h'} s_3 \).

Then, we obtain the next interface, which is not secure:

\[ S_0 \xrightarrow{h!} S_2 \]

\[ a! \]

\[ S_1 \]
Undetermined states (example 2):

If transition $s_2 \xrightarrow{h'?} s_3$ is eliminated, the resulting interface is secure.

Notice: this example is the previous one with the new transition: $s_2 \xrightarrow{a!} s_1$. 

Matias Lee, Pedro R. D'Argenio
The definitions of May/Fail/Undetermined state are inductive. Under some restrictions the property propagates to predecessor states in the synchronous product.

If the initial state of the sync. product is a may (fail, undetermined) state, we say that interface may pass (fail, is undetermined w.r.t.) the bisimulation test.

\[ S_0, S_0 \xrightarrow{\xi} S_0, S_2 \]
\[ S_1, S_1 \xrightarrow{a!} S_0, S_3 \]
\[ b? \]
\[ \text{fail} \]

\[ S_0, S_0 \xrightarrow{\xi} S_0, S_2 \]
\[ S_1, S_1 \xrightarrow{a!} S_0, S_3 \]
\[ b! \]
\[ \text{fail} \]

\[ S_0, S_0 \xrightarrow{\xi} S_0, S_2 \]
\[ S_1, S_1 \xleftarrow{a!} S_0, S_3 \]
\[ \text{fail} \]
Main result:

**Theorem**

If \( S \setminus A_h^\emptyset \times S \setminus A_h^{A_h,t} \) may pass the bisimulation test, then there is a set \( \rightarrow \chi \) of low input transitions s.t. the ISS obtained from \( S \) by removing all transitions in \( \rightarrow \chi \) is BSNNI.

- The set \( \rightarrow \chi \) is obtained by calculating a succession sets \( EC(S) \) of *eliminable candidates*
- \( EC(S) \) contains all (low input) transitions that go from *May* states to *Fail* or *Undetermined* states.
- The proof is constructive and it defines the algorithm.
Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
   - Bisimulation-based (Strong) Non-deterministic Non-interference

2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm - Example

3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

4. Contribution and future works
Iteration 1

\[ EC(S) = \{ s_1 \xrightarrow{\text{acceptT?}} s_1 \} \]
Iteration 2

\[ EC(S) = \{s_2 \xrightarrow{\text{acceptT?}} s_2\} \]
Iteration 3

We obtain a secure interface!

and $\rightarrow_{\chi} = \{ s_1 \xrightarrow{\text{acceptT?}} s_1, s_2 \xrightarrow{\text{acceptT?}} s_2 \}$
Outline

1 Interfaces Structure for Security
   • Interfaces Automata and Interface Structure For Security
   • Composition
   • Bisimulation-based (Strong) Non-deterministic Non-interference

2 Deriving secure ISS
   • Checking BSNNI
   • Synthesizing Secure ISS
   • The algorithm - Example

3 Preserving BSNNI after Composition
   • Preserving BSNNI after Composition

4 Contribution and future works
The following lemma give sufficient conditions to ensure that composition leads to secure systems.

**Lemma**

Let $S = \langle S, A^h_S, A^l_S \rangle$ and $T = \langle T, A^h_T, A^l_T \rangle$ be two composable ISS. Define

- $S' = \langle S, A^h_S - \text{shared}(S, T), A^l_S \cup \text{shared}(S, T) \rangle$
- $T' = \langle T, A^h_T - \text{shared}(S, T), A^l_T \cup \text{shared}(S, T) \rangle$

If $S'$ and $T'$ are BSNNI/BNNI and $S \otimes T$ has not error states, then $S \parallel T$ is BSNNI/BNNI.
Contributions

- We extended Interface Automata to cope with security and adapted the definition of no interference to this context.
- We design an algorithm to synthesis a secure interface from a non-secure one whenever possible.
- The algorithm proceeds by controlling the permitted low input transitions.
- We give sufficient conditions to ensure that the composition of ISS results in a non-interferent ISS.
Future Work

- Relax the necessary conditions to preserve no interference under composition.
- Adapt the concept of refinement of IA to ISS and studying its relation to BSNNI and BNNI.
- Define new concepts of “security” to ISS and adapt the results of this work to the new definitions.