The Security of Mobile Agent Systems

Wiesław Pawłowski
Modeling and Reasoning about Mobile Agent Systems
- with some applications to security -
Joint work with

- Marek A. Bednarczyk – Institute of Computer Science PAS, PL
- Luca Bernardinello – University of Milano Bicocca, IT
- Tomasz Borzyszkowski – Gdańsk University, PL
- Wojciech Jamroga – University of Clausthal, DE
- Lucia Pomello – University of Milano Bicocca, IT
**Systems, views, models**

- **System**
  - dynamic (in a non-technical sense)
  - composed of (mobile) agents possessing “identity”

- **View**
  - description of a certain aspect of system behavior

- **System model**
  - “synchronous composition” of several views
Systems, views, specifications

- **System**
  - *dynamic (in a non-technical sense)*
  - *composed of (mobile) agents possessing “identity”*

- **View**
  - description of a certain aspect of system behavior

- **System specification**
  - “synchronous composition” of several views
Simple example

“Flip-flop” message exchange

- **agents:**
  - communicators – senders/receivers
  - messages

- **assumptions**
  - there is a message flow infrastructure between communicators
  - messages are addressed – contain info about sender and addressee
  - only addressee can receive a message
  - addressee can send the message back to the sender
Different aspects of system behavior

- **physical view** – *message flow*
  - **agents**: communicators, messages, communication infrastructure
  - **describes**: *topology* of the communication infrastructure

- **logical view** – *addressing of the messages*
  - **agents**: communicators and messages
  - **describes**:
    - sender and addressee for each message
    - message *delivery control*
Modeling the physical view

- We shall use Petri hypernets as a modeling framework

- agents:
  - communicators, messages and communication infrastructure

- actions (transitions):
  - send – a communicator sends a message
  - receive – a message is delivered to a communicator
  - pass_on – a message changes its location within the communication infrastructure
Petri hypernets in a nutshell

- **Agents** \( \approx \) Petri nets with "communication ports"
  - based on the "nets-within-nets" idea (nets as tokens)
  - **locations** \( \equiv \) places in nets
- **Hypernet** \( \approx \) set of "typed" nets
- **Dynamics**
  - hypernet state (hypermarking) \( \approx \) type-preserving placement of agents in locations (agent has **at most one** location at a given time)
  - agents form **consortia** to perform common actions (transitions)
  - hierarchy of agents evolves as a result of firing consortia
  - number and "identity" of agents **does not change**
**Communication infrastructure agent**

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) `pass_on` is a part of the message transport module
Infrastructure & messages

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)

- Action (transition) `pass_on` is a part of the message transport module
Infrastructure & messages

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
Infrastructure & messages

- Infrastructure composed of two "modules":
  - message transport (gray)
  - communicator (white)

- Action (transition) pass_on is a part of the message transport module
Infrastructure & messages

- Infrastructure composed of two "modules":
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
Infrastructure & messages

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
Infrastructure composed of two "modules":
- message transport (gray)
- communicator (white)

Action (transition) pass_on is a part of the message transport module

pass_on, pass_on
Infrastructure & messages

- Infrastructure composed of two "modules":
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module

pass_on, pass_on, pass_on, ...
Infrastructure, messages & communicators

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
- Delivery/sending of messages is modeled by “inter-level” transitions (yellow)
Infrastructure, messages & communicators

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
- Delivery/sending of messages is modeled by “inter-level” transitions (yellow)
Infrastructure, messages & communicators

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
- Delivery/sending of messages is modeled by “inter-level” transitions (yellow)
Infrastructure, messages & communicators

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
- Delivery/sending of messages is modeled by “inter-level” transitions (yellow)
Infrastructure, messages & communicators

- Infrastructure composed of two “modules”:
  - message transport (gray)
  - communicator (white)
- Action (transition) pass_on is a part of the message transport module
- Delivery/sending of messages is modeled by “inter-level” transitions (yellow)
Physical message flow

- firings of receive and send
  - manipulate pairs of agents
  - message – receiver/sender
- receive – moves a message from a communication infrastructure location “inside” the receiver agent
- send – moves a message from “inside” the sender to a communication infrastructure location
Physical message flow – receive action
Physical message flow – an abstraction

- \( A \equiv \text{Mess} \cup \text{Comm} \cup \{ \text{infra} \} \) \(\Leftrightarrow\) agents
- \( S \equiv (\text{Mess} \rightarrow \text{Loc}^m) \times (\text{Comm} \rightarrow \text{Loc}^c) \) \(\Leftrightarrow\) states
  - where
    - \( \text{Loc}^m \) – set of message locations
    - \( \text{Loc}^c \) – set of communicator locations
- \( T \equiv \{ \text{pass\_on, send, receive} \} \) \(\Leftrightarrow\) transitions
- \( \delta : T \rightarrow \mathcal{P}(A) \rightarrow (S \rightharpoonup S) \) \(\Leftrightarrow\) transition function

\( \delta(t)(\alpha)(s) \equiv \text{if } \alpha \text{ is a } t\text{-consortium in } s \text{ then } \text{“result of firing the consortium } s\text{”} \text{ else } \text{“undefined”} \)
Logical view – message addressing

- \( A' \overset{\text{def}}{=} \text{Mess} \cup \text{Comm} \)
- \( S' \overset{\text{def}}{=} \text{Mess} \rightarrow \text{Comm} \times \text{Comm} \)
- \( T' \overset{\text{def}}{=} \{ \text{send}, \text{receive} \} \)

- \( \delta'(\text{send})(\alpha)(s) \overset{\text{def}}{=} \text{if } (\exists \ m,c,c') \ \alpha=\{m,c\} \ \land \ s(m)=(c,c') \ \text{then } s \ \text{else } \text{“undefined”} \)

- \( \delta'(\text{receive})(\alpha)(s) \overset{\text{def}}{=} \text{if } (\exists \ m,c,c') \ \alpha=\{m,c'\} \ \land \ s(m)=(c,c') \ \text{then } s [m \mapsto (c',c)] \ \Leftarrow \text{address transposition} \ \text{else } \text{“undefined”} \)
Agent-aware transition systems

agent-aware transition system \((A, T, S, \delta)\):

- \(A\) set of agents
- \(T\) set of transitions
- \(S\) set of states
- \(\delta : T \rightarrow \mathcal{P}(A) \rightarrow (S \rightharpoonup S)\) transition function

(simple modification of the deterministic transition system notion)

View = agent-aware transition system
Synchronization of views

Let \( \{ S_i \mid i \in I \} \) - a family of views \( S_i = (A_i, T_i, S_i, \delta_i) \)

\[
\prod_{i \in I} S_i \equiv \langle \bigcup_{i \in I} A_i , \bigcup_{i \in I} T_i , \prod_{i \in I} S_i , \Delta \rangle
\]

where

\[
\Delta(t)(\alpha)(s) \approx \begin{cases} 
\text{if} & (\exists k \in I) \ t \in T_k \land \delta_k(t)(\alpha \cap A_k)(s_k) \text{ is undefined} \\
\text{then} & \text{“undefined”} \\
\text{else} & \langle \begin{cases} \text{if} & t \in T_i \ \text{then} \ \delta_i(t)(\alpha \cap A_i)(s_i) \ \text{else} \ s_i \mid i \in I \end{cases} \rangle
\end{cases}
\]
Some “unwanted” features of views considered separately:

- **physical**: abstracts from addressing \((\text{send}, \text{receive})\)
- **logical**: disregards physical placement of agents \((\text{send}, \text{receive})\)

After synchronization:

- action **pass_on** (autonomous for the physical view) modeled exactly as before
- actions **send** and **receive** take into account both addressing and physical placement of message and sender/receiver
Synchronization as a product

morphism $\sigma : S_1 \rightarrow S_2$ of agent-aware t.s. $S_i = (A_i, T_i, S_i, \delta_i)$ $i=1,2$

s.t. $A_1 \supseteq A_2$ and $T_1 \supseteq T_2$ is a function $\sigma : S_1 \rightarrow S_2$ such that:

- $\forall t \in T_1$, $\alpha \subseteq A_1$, $s \in S_1$ if $t \in T_2$ and $\delta_1(t)(\alpha)(s)$ is defined then
  $\delta_2(t)(\alpha \cap A_2)(\sigma(s)) = \sigma(\delta_1(t)(\alpha)(s))$

- $\forall t \in T_1$, $\alpha \subseteq A_1$, $s \in S_1$ if $t \notin T_2$ and $\delta_1(t)(\alpha)(s)$ is defined then
  $\sigma(s) = \sigma(\delta_1(t)(\alpha)(s))$

ATS – the category of agent-aware transition systems and their morphisms

Proposition. Synchronization = categorical product in ATS.
**Synchronization as a product**

morphism $\sigma : S_1 \rightarrow S_2$ of agent-aware t.s. $S_i = (A_i, T_i, S_i, \delta_i)$ $i=1,2$

s.t. $A_1 \supseteq A_2$ and $T_1 \supseteq T_2$ is a function $\sigma : S_1 \rightarrow S_2$ such that:

\[
\begin{cases}
    \sigma(s) = \sigma(s') & t \in T_2 \\
    \sigma(s) \xrightarrow{t, \alpha \cap A_2} \sigma(s') & \text{otherwise}
\end{cases}
\]

ATS – the category of agent-aware transition systems and their morphisms

**Proposition.** Synchronization = categorical product in ATS.
Slightly more interesting example

Exchange of (possibly) encrypted messages

- **agents:**
  - communicators – senders/receivers
  - messages
  - keys – public and secret

- **assumptions**
  - there is a message flow infrastructure between communicators
  - message *might* be encrypted with receiver's public key
  - appropriate secret key is needed to decrypt an encrypted message
Five views

• physical

• logical 1 – encrypting and decrypting of messages
  – agents: messages and public keys

• logical 2 – public and secret keys relationship
  – agents: public and secret keys

• logical 3 and 4 – communicators' knowledge about public/secret keys
  – agents: communicators and public/secret keys
1. Encrypting and decrypting messages

- $A^1 \equiv \text{Mess} \cup \text{PubKey}$
- $S^1 \equiv \text{Mess} \rightarrow \text{PubKey}^*$
- $T^1 \equiv \{ \text{send, receive} \}$

- $\delta^1(\text{send})(\alpha)(s) \equiv \text{case } \alpha \text{ of}$
  - $\{m,pk\} : s \ [ m \mapsto pk \cdot s(m) ] \iff \text{encrypt with “pk”}$
  - $\{m\} : s \iff \text{“plain-text”}$
  - else “undefined”
1. Encrypting and decrypting of messages

- $A^1 \triangleq \text{Mess} \cup \text{PubKey}$
- $S^1 \triangleq \text{Mess} \rightarrow \text{PubKey}^*$
- $T^1 \triangleq \{ \text{send, receive} \}$

- $\delta^1(\text{receive})(\alpha)(s) \triangleq \begin{cases} \text{if } \alpha = \{m, pk\} \land s(m) = pk \cdot pks \\
\text{then } s[m \mapsto pks] \Leftrightarrow \text{message decrypting} \\
\text{elseif } \alpha = \{m\} \land s(m) = \text{null} \\
\text{then } s \Leftrightarrow \text{"plain-text" message} \\
\text{else } \text{"undefined"} \end{cases}$
2. *Public–Secret key relationship*

- \( A^2 \overset{\text{def}}{=} \text{PubKey} \cup \text{SecKey} \)
- \( S^2 \overset{\text{def}}{=} \text{PubKey} \rightarrow \text{SecKey} \)
- \( T^2 \overset{\text{def}}{=} \{ \text{receive} \} \)

\[
\delta^2(\text{receive})(\alpha)(s) \overset{\text{def}}{=} \begin{cases} 
\text{if } \alpha = \emptyset \lor \alpha = \{pk,sk\} \land s(pk) = sk \\
\text{then} & s \\
\text{else} & \text{“undefined”}
\end{cases}
\]
3. *Communicators' knowledge – public keys*

- \( A^3 \overset{\text{def}}{=} \text{Comm} \cup \text{PubKey} \)
- \( S^3 \overset{\text{def}}{=} \text{Comm} \rightarrow \mathcal{P}(\text{PubKey}) \)
- \( T^3 \overset{\text{def}}{=} \{ \text{send} \} \)

- \( \delta^3(\text{send})(\alpha)(s) \overset{\text{def}}{=} \text{if } \alpha = \{c\} \lor \alpha = \{c,pk\} \land pk \in s(c) \text{ then } s \text{ else } \text{“undefined”} \)
4. *Communicators' knowledge – secret keys*

- \( A^4 \overset{\text{def}}{=} \text{Comm} \cup \text{SecKey} \)
- \( S^4 \overset{\text{def}}{=} \text{Comm} \rightarrow \mathcal{P}(\text{SecKey}) \)
- \( T^4 \overset{\text{def}}{=} \{ \text{receive} \} \)

\[ \delta^4(\text{receive})(\alpha)(s) \overset{\text{def}}{=} \begin{cases} s & \text{if } \alpha = \{c\} \lor \alpha = \{c,sk\} \land sk \in s(c) \\ \text{"undefined"} & \text{else} \end{cases} \]
Synchronous transition function - \textit{send}

$\Delta(\text{send})(\alpha)(s)$

- physical: $\{ \text{inf, m, c} \}$
- logical 1: $\{ \text{m} \}$
- logical 2: not involved
- logical 3: $\{ \text{c} \}$
- logical 4: not involved

\text{message encryption}
\text{public/secret keys rel.}
\text{public keys knowledge}
\text{secret keys knowledge}
Synchronous transition function – receive

\[ \Delta(\text{receive})(\alpha)(s) \]

- physical: \{ inf, m, c \}
- logical 1: \{ m \} \rightarrow \{ m, pk \}
- logical 2: \emptyset \rightarrow \{ pk, sk \}
- logical 3: not involved
- logical 4: \{ c \} \rightarrow \{ c, sk \}

message encryption
public/secret keys rel.
public keys knowledge
secret keys knowledge
Synchronous transition function – \textit{pass\_on}

\[ \Delta(\text{pass\_on})(\alpha)(s) \]

- physical: \( \{ \inf, m \} \)
- logical 1: \textit{not involved}
- logical 2: \textit{not involved}
- logical 3: \textit{not involved}
- logical 4: \textit{not involved}

message encryption

public/secret keys rel.

public keys knowledge

secret keys knowledge
• We know:
  – how to model system views
  – how to obtain the description of the dynamics of the whole system from the composition of views

• Problem:
  – how to specify and verify (the desired) system properties
Interpreted agent-aware transition systems

interpreted agent-aware transition system \((A, T, S, \delta, \Sigma, \xi)\):

- \((A, T, S, \delta)\) agent-aware transition system
- \(\Sigma\) ranked alphabet (predicate names)
- \(\xi : S \rightarrow \text{RelStr}(\Sigma, A)\) interpretation function

where \(\text{RelStr}(\Sigma, A)\) is the set of \(\Sigma\)-relational structures with carrier \(A\)

Remarks:
- Interpreted agent-aware t.s. constitute a category \(\text{IATS}\)
- Synchronization of \(iaas\) is a categorical product in \(\text{IATS}\)
Agent-terms and valuations

Agent-term: either agent name or agent variable

\[ \text{Term}(A) ::= \{ [a] \mid a \in A \} \cup X \]

Valuations and agent-term interpretations

\[ v : X \rightarrow A \]
\[ v^\# : \text{Term}(A) \rightarrow A \text{ where } v^\#([a]) \overset{\text{def}}{=} a \]
Formulas and satisfaction relation

• Formulas
  \[ \varphi ::= \text{false} \mid r(t_1,\ldots,t_n) \mid \neg \varphi \mid \varphi \lor \psi \mid E[\varphi U \psi] \mid EG \varphi \]

• Satisfaction relation for \( M = (A, T, S, \delta, \Sigma, \xi) \)
  \[ s, v \models \text{false} \quad \text{never} \]
  \[ s, v \models \neg \varphi \quad \text{iff} \quad \text{not} \ s, v \models \varphi \]
  \[ s, v \models \varphi \lor \psi \quad \text{iff} \quad s, v \models \varphi \text{ or } s, v \models \psi \]
  \[ s, v \models r(t_1,\ldots,t_n) \quad \text{iff} \quad (v^#(t_1), \ldots, v^#(t_n)) \in r_{\xi(s)} \]
  \[ s, a \models E[\varphi U \psi] \quad \text{iff} \quad \text{there is a path } \pi \text{ from } s \text{ such that } \pi_k, v \]
  \[ \models \psi \quad \text{for some } k, \text{ and } \pi_j, v \models \varphi, \text{ for } j=1,\ldots,k-1 \]
  \[ s, a \models EG \varphi \quad \text{iff} \quad \text{there is a path } \pi \text{ from } s \text{ such that } \pi_k, v \models \varphi \]
  \[ \text{for all } k \]
Some useful derived formulas

1. There exists a path such that $\varphi$ eventually holds on it
   - $\text{EF} \varphi \equiv \text{E}[\text{true}\cup \varphi]$

2. For all paths $\varphi$ holds “globally” – for all states occurring in them
   - $\text{AG}\varphi \equiv \neg\text{EF}\neg\varphi$

3. For all paths $\varphi$ eventually holds on them
   - $\text{AF}\varphi \equiv \neg\text{EG}\neg\varphi$

4. For all paths $\varphi$ holds “until” $\psi$ holds
   - $\text{A}[\varphi\cup\psi] \equiv \neg(\text{E}[\neg\psi\cup(\varphi \vee \psi)] \vee \text{EG}\neg\psi)$

Our logic $\approx$ CTL – “next-step operator” + “n-ary predicates”
Public key cryptography example revisited

Ranked alphabets and their interpretation for the views

- **physical**: a binary relational symbol $\text{SubAgent}$ with the obvious agent-containment interpretation
- **logical 1**: binary symbol $\text{LastEnc}$ interpreted as relation between messages and public keys ("last encrypted with")
- **logical 2**: binary symbol $\text{MatchK}$ interpreted as as a relation between public and secret keys ("matching keys")
- **logical 4**: binary symbol $\text{KnowsSecK}$ interpreted as as the obvious relation between communicators and public keys
Public key cryptography example revisited

\[
\text{LastEnc}(m, pk) \land \text{MatchK}(pk, sk) \land \text{KnowsSecK}(c, sk) \\
\downarrow \\
\text{EF SubAgent}(c, m)
\]

if a communicator possesses “the right” secret key then there is a possible temporal evolution of the system leading to a state where the encrypted message has been delivered to him

Liveness property
Public key cryptography example revisited

MatchK(pk, sk) \land \neg KnowsSecK(c, sk)
\Downarrow
\neg E[LastEnc(m, pk) \cup SubAgent(c, m)]

if a communicator does not possess “the right” secret key he will not be able to receive the encrypted message

Safety property
Public key cryptography example revisited

KnowsSecK(c,sk) ⇒ AG KnowsSecK(c,sk) ∧ ¬KnowsSecK(c,sk) ⇒ AG ¬KnowsSecK(c,sk)

communicators neither forget nor acquire secret keys

Safety property
Public key cryptography example revisited

AG ( KnowsSecK(c,[k_a]) ⇒ c = [a] )

k_a is a “private key” of the communicator a

Safety property
Conclusions

What has been presented

- agent-aware transition systems – a unified framework for modeling dynamic agent systems
- system model = categorical product of views
- simple (but expressive) logic for reasoning about properties of the system

Future work

- extend and refine the existing model checking tools for the presented logic using the Verics model-checker system
- apply the framework to a wide spectrum of case-studies