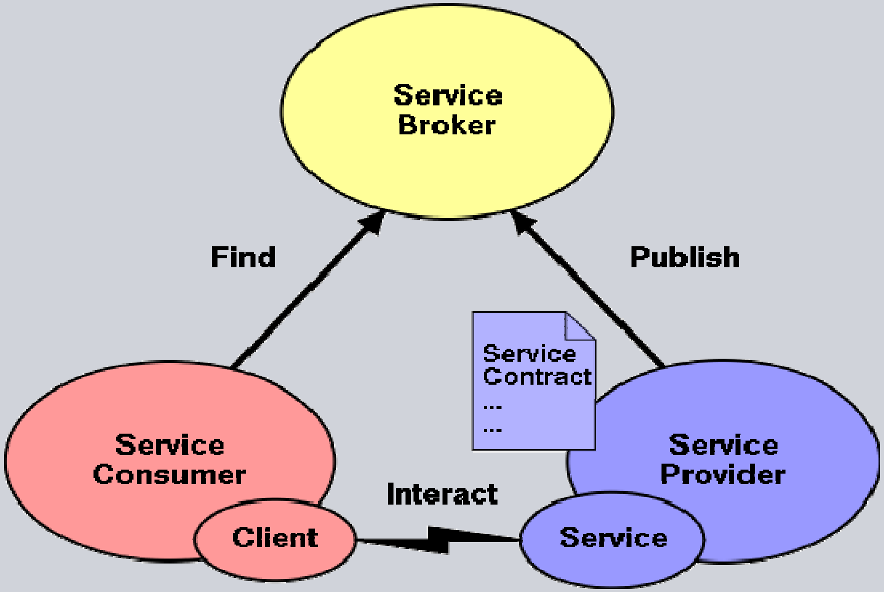
EU FP7-2007-ICT-1, ICT-1.1.4, STREP project no. 216471
Jan 2008 - Dec 2010, 590 PMs, 6M€ budget, 3.8M€ EC contribution

SIEMENS

avantssar.eu

ASLan++ — the AVANTSSAR[1] Specification Language



Presented at the FMCO 2010, Graz, Austria, 2010-Nov-30

1 **A**utomated **V**alidatio**N** of **T**rust and **S**ecurity of **S**ervice-oriented **Ar**chitectures

AVANTSSAR

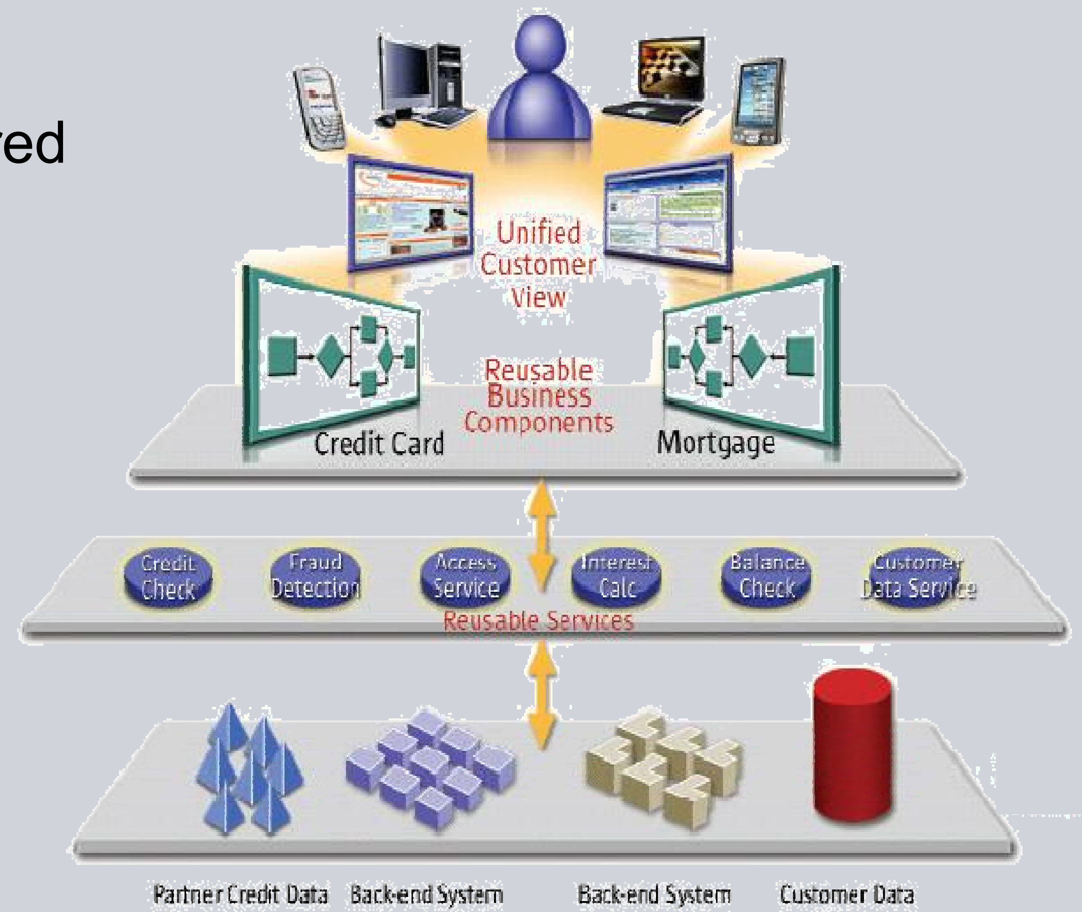
AVANTSSAR project motivation

SIEMENS

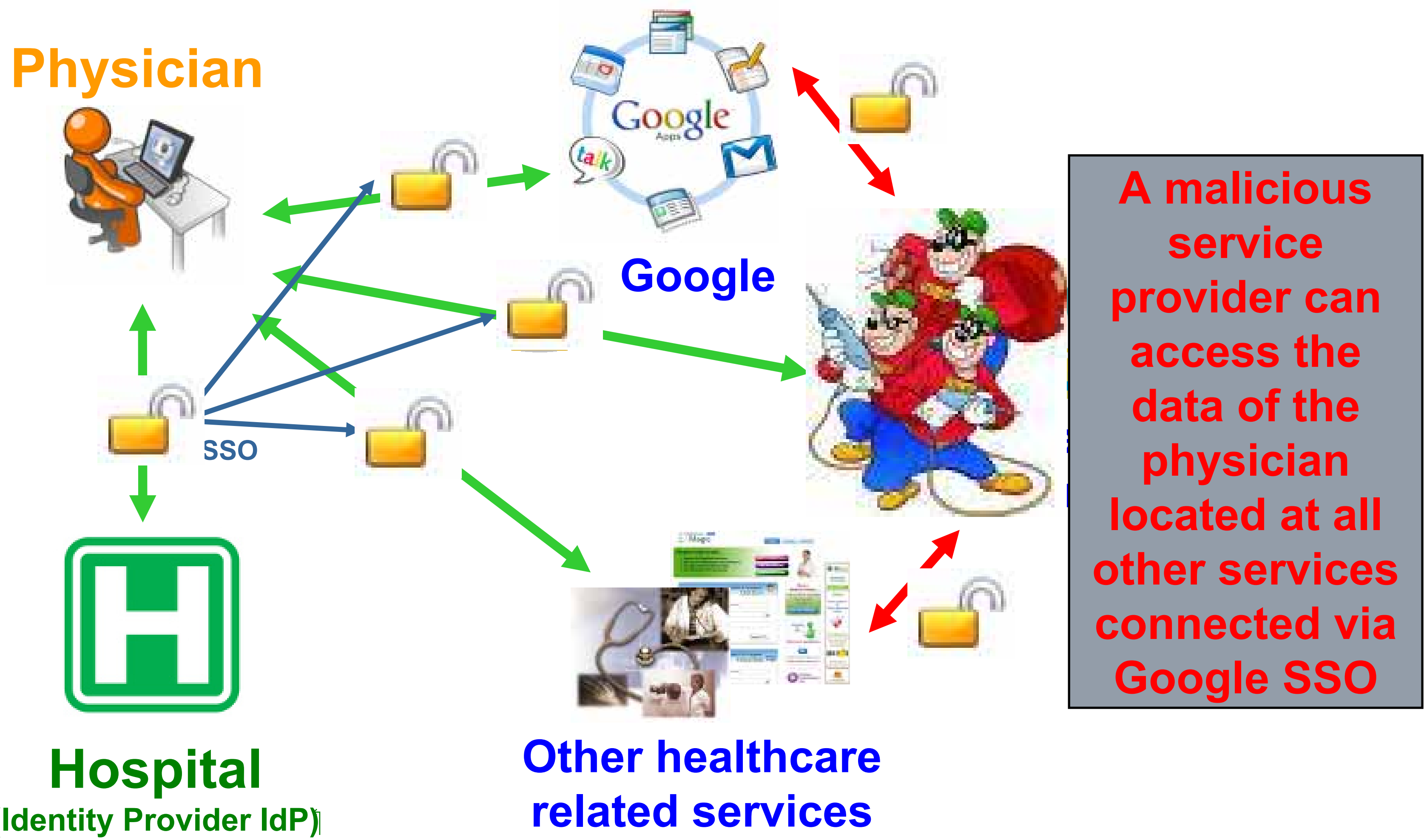
ICT paradigm shift: from components to services, composed and reconfigured dynamically in a demand-driven way.

Trustworthy service may interact with others causing novel trust and security problems.

For the composition of individual services into service-oriented architectures, validation is dramatically needed.



Example 0: Google SAML-based Single Sign-On (SSO)



Example 0: Google SAML SSO protocol flaw

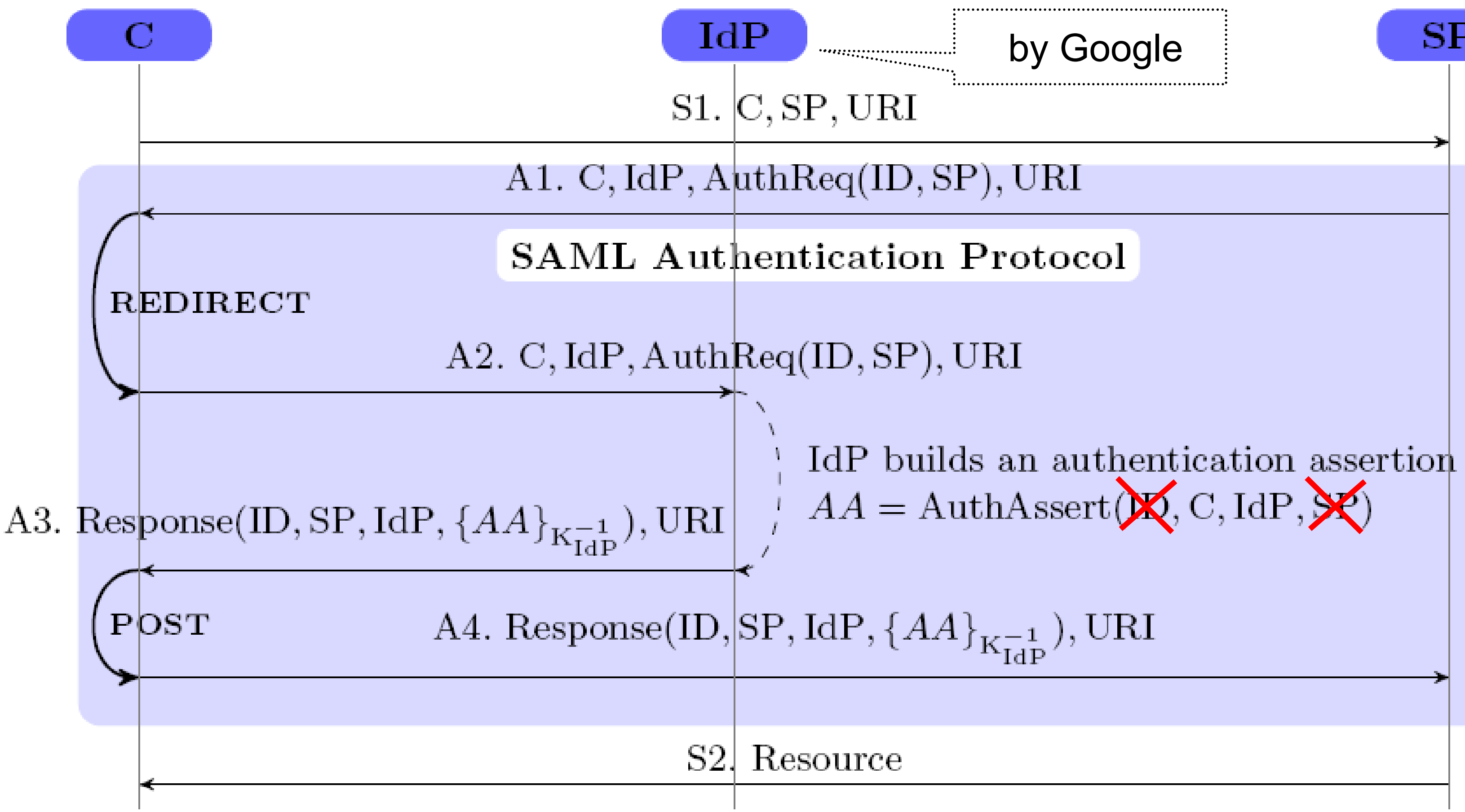


Fig. 1. SP-Initiated SSO with Redirect/POST Bindings

AVANTSSAR consortium

Industry

SAP Research France, Sophia Antipolis

Siemens Corporate Technology, München

IBM Zürich Research Labs (initial two years)

OpenTrust, Paris

Academia

Università di Verona

Università di Genova

ETH Zürich

INRIA Lorraine

UPS-IRIT, Toulouse

IEAT, Timişoara

Expertise

Service-oriented enterprise architectures

Security solutions

Standardization and industry migration

Security engineering

Formal methods

Automated security validation

SIEMENS

AVANTSSAR main objectives and principles

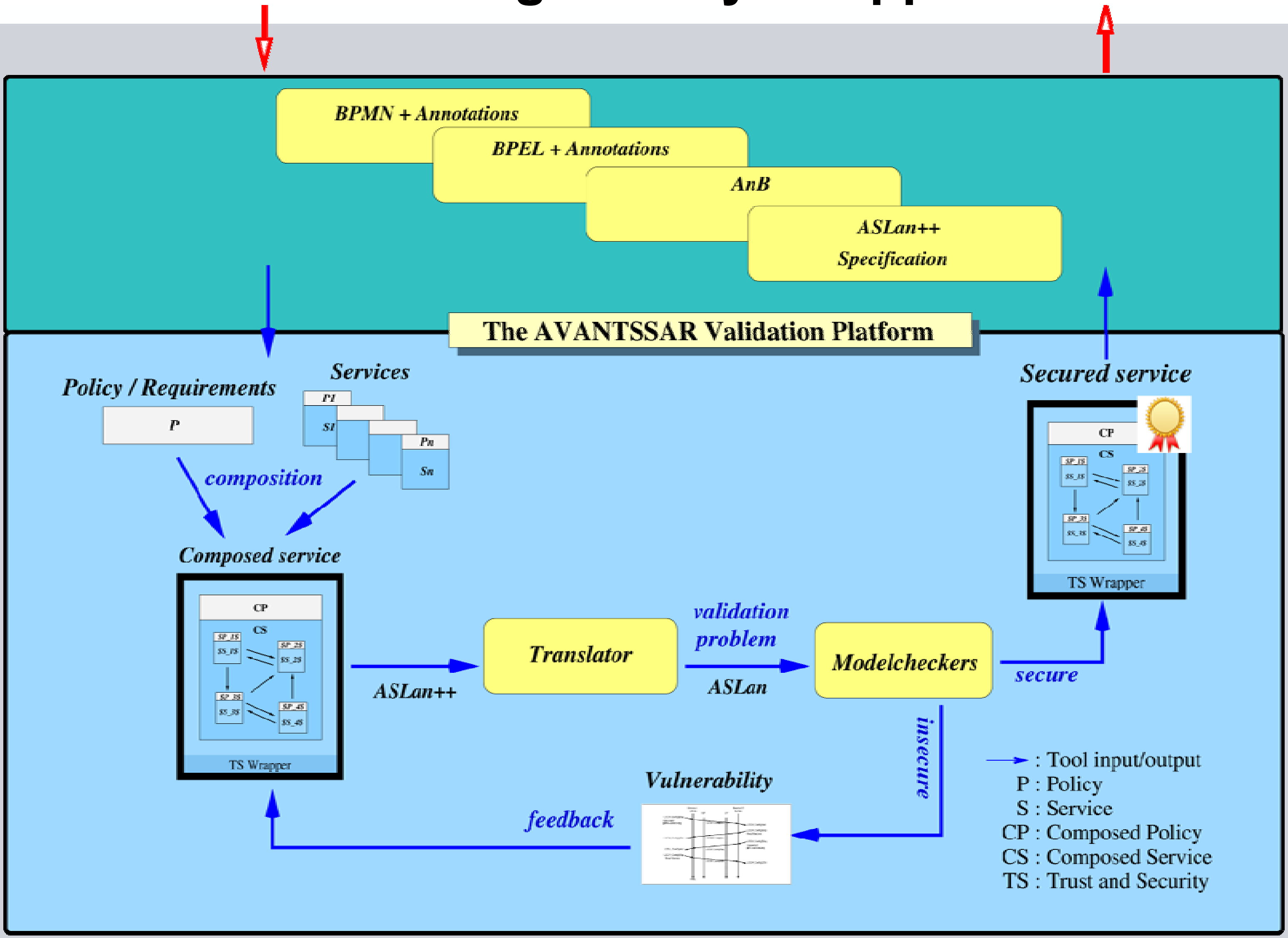
AVANTSSAR product: Platform for formal specification and automated validation of trust and security of SOAs

- Formal language for specifying trust and security properties of services, their policies, and their composition into service-oriented architectures
- Automated toolset supporting the above
- Library of validated industry-relevant case studies

Migration of platform to industry and standardization organizations

- Speed up development of new service infrastructures
- Enhance their security and robustness
- Increase public acceptance of SOA-based systems

AVANTSSAR modeling & analysis approach with ASLan++



AVANTSSAR: current status

SIEMENS

WP2: ASLan++ supports the formal specification
of trust and security related aspects of SOAs, and
of static and dynamic service and policy composition

WP3: Techniques for: satisfiability check of policies,
model checking of SOAs w.r.t. dynamic policies,
attacker models, compositional reasoning, abstraction

WP4: Second prototype of the **AVANTSSAR Platform**

WP5: Formalization of **industry-relevant problem cases**
as ASLan++ specifications and their validation

WP6: Ongoing dissemination and migration
into scientific community and industry

SIEMENS

AVANTSSAR conclusion and industry migration

Contemporary SOA has complex structure and security requirements including dynamic trust relations and application-specific policies.

On integration of the AVANTSSAR Platform in industrial development, a rigorous demonstration that the security requirements are fulfilled will:

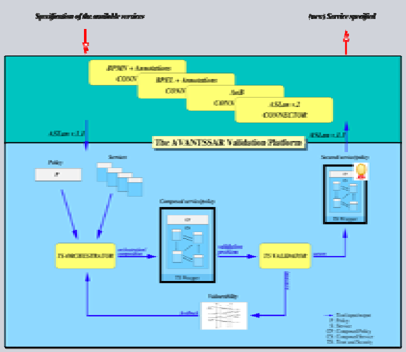
- assist developers with security architecture, analysis and certification
- increase customers' confidence in modern service-oriented architectures

The AVANTSSAR Platform advances the security of industrial vendors' service offerings: validated, provable, traceable.

AVANTSSAR will thus strengthen the competitive advantage of the products of the industrial partners.

E-Business

Infrastructure



Citizen Portals

Health care

ASLan++

SIEMENS

Example 1: ASLan++ model of NSPK_Cert (1): Alice & Bob

```
specification NSPK_Cert
  …
  entity Alice (Actor, B: agent) {
    symbols
     Na, Nb: message;
    body {
      if(trusted_pk(B)) {
        Na := fresh();
        Actor -> B: {secret_Na:Na.Actor}_pk(B);
        B -> Actor: {Alice_strong_auth_Bob_on_Na:Na.secret_Nb:?Nb}_pk(Actor);
        Actor -> B: {Bob_strong_auth_Alice_on_Nb:Nb}_pk(B); } }
  }
  entity Bob (Actor: agent) {
    symbols
      A: agent;
      Na, Nb: message;
    body {
      ?A -> Actor: {secret_Na:?Na.?A}_pk(Actor); % Bob learns A here!
      if (trusted_pk(A)) {
        Nb := fresh();
        Actor -> A: {Alice_strong_auth_Bob_on_Na:Na.secret_Nb:Nb}_pk(A);
        A -> Actor: {Bob_strong_auth_Alice_on_Nb:Nb}_pk(Actor); } }
  } …
}
```

Example 1: ASLan++ model of NSPK_Cert (2): certificates

```
specification NSPK_Cert channel_model CCM
entity Environment {

  symbols
    trusted_pk(agent): fact;
    trusted_agent(agent): fact;
    root_ca, ca: agent;
    issued(message): fact;

  macros
    A->signed(M)   = {M}_inv(pk(A)).M;
    C->cert(A,PK)  = C->signed(C.A.PK); % no validity period etc.

  clauses
    trusted_pk_direct(C):
      trusted_pk(C) :-
      trusted_agent(C);

    trusted_pk_cert_chain(A,B):
      trusted_pk(A) :-
      trusted_pk(B) & issued(B->cert(A,pk(A)));
```


Example 1: ASLan++ model of NSPK_Cert (3): sessions

```
entity Session (A, B: agent) {
  entity Alice (Actor, B: agent) {…}
  entity Bob (Actor: agent) {…}
  body {
    issued(ca->cert(A,pk(A)));
    issued(ca->cert(B,pk(B)));
    new Alice(A,B);
    new Bob(B);
  }
  goals
    secret_Na: {A,B};
    secret_Nb: {A,B};
    Alice_strong_auth_Bob_on_Na: B *->> A;
    Bob_strong_auth_Alice_on_Nb: A *->> B;
}
body { % need two sessions for Lowe's attack
  trusted_agent(root_ca);
  issued(root_ca->cert(ca,pk(ca))); % root-signed CA certificate
  issued(     ca->cert(i ,pk(i ))); % CA-signed intruder cert
  any A B. Session(A,B) where A!=B;
  any A B. Session(A,B) where A!=B; } }
```

SIEMENS

ASLan++ language design

- **Design goals**
  - **Expressive** enough for *modeling a wide range of SOAs*
  - Enable **succinct** specifications, for *minimal handling effort*
  - High **abstraction** level, to *reduce model complexity*
  - Close to **specification languages** for security protocols and web services
  - Close to procedural and object-oriented **programming languages**
  - *Minimal learning effort* for **non-expert** modelers
- **Relation with ASLan**
  - ASLan++ more **high-level** than ASLan (formerly called IF)
  - ASLan++ semantics defined by **translation** to ASLan
  - Main differences:

hierarchy of classes	vs. flat transition system
procedural statements	vs. term rewriting rules
high-level security goals	vs. attack states & auxiliary events

ASLan++ features for system modelling

- **Overall structure**
  - Hierarchy and modularity via entities (similar to classes)
  - Dynamic entity instantiation (with underspecified agents)
  - Parallel composition of sequential execution of instances
- **Local declarations**
  - Types with subtyping, tuples and generic sets
  - Constants, functions, statically scoped instance variables
  - Horn clauses describing policies and (limited) deductions
- **Local execution**
  - Classical control flow constructs (e.g. if and while)
  - Cryptographic primitives and fresh value generation
  - Pattern matching (unification modulo some equalities)
  - Send and receive instructions with guards
  - Channels with security assumptions

SIEMENS

ASLan++ features for security property modelling

- **Security goals**
  - Invariants (LTL formulas)
  - Assertions (LTL formulas)
  - Secrecy of values shared among a group of agents
  - Channel goals: authenticity, confidential transmission, freshness, …
- **Attacker model**
  - Built-in Dolev-Yao intruder model
  - Extendible intruder knowledge
  - Dishonest agents and dynamic compromise of agents
- **Limitations (mostly due to model-checking)**
  - No term evaluation (e.g., arithmetic) except very limited equations
  - No notion of time
  - No abstract data types and visibility modifiers
  - No object references except for sets

Backup slides: further examples

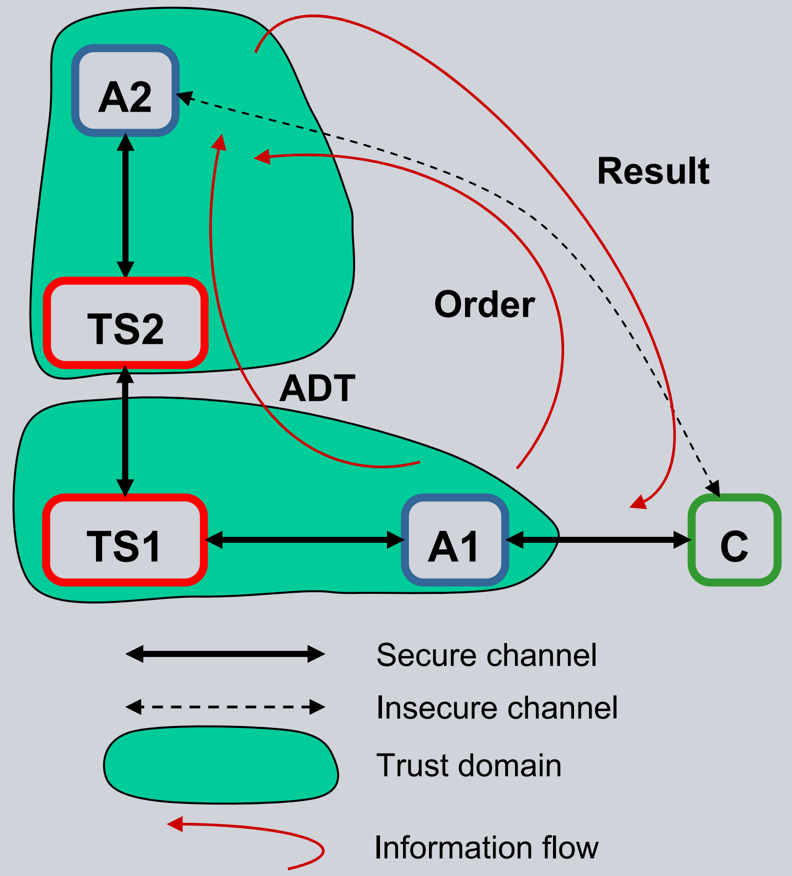
SIEMENS

Example 2: Process Task Delegation (PTD)

Authorization and trust management via token passing

- There are three roles in the protocol (**C, A, TS)** and potentially several instances for each role
- The ***client*** **C** (or *user*) uses the system for authorization and trust management, e.g. SSO
- Each ***application*** **A** is in one domain, each domain has exactly one active ***trust server*** **TS**
- **A1** uses the system to pass to **A2** some **Order** and an **ADT (Authorization Decision Token)**
  - **Order** contains:
    - workflow task information
    - application data
    - information about the client **C** and his current activity

    to be delivered securely (integrity and confidentiality)
  - **ADT** is mainly authorization *attributes* and *decisions*
    - sent via **TS1** and **TS2**, who may weaken it
    - must remain unaltered, apart from weakening by **TS**
    - must remain confidential among intended parties
- **C**, **A1**, and **A2** must be authenticated among each other



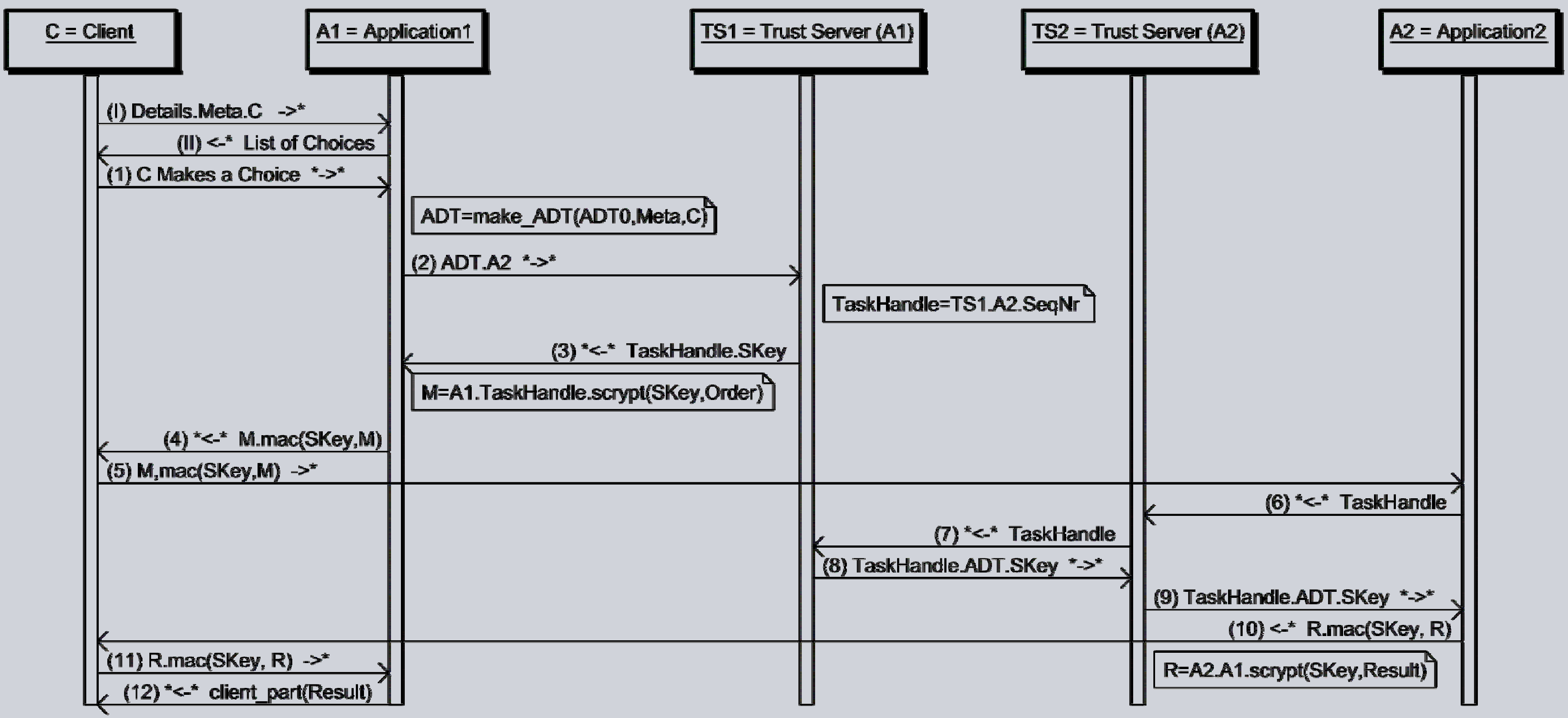
Security prerequisites:

- PKI is used for **A** and **TS**, username & pwd for **C**
- The **TS** enforce a strict time-out

SIEMENS

Example 2: Message Sequence Chart of PTD

..\..\deliv\5.2\figures\PTD_generic.pdf



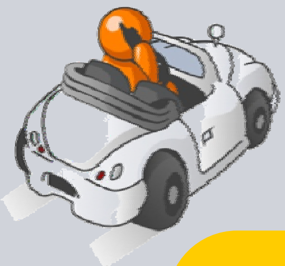
SIEMENS

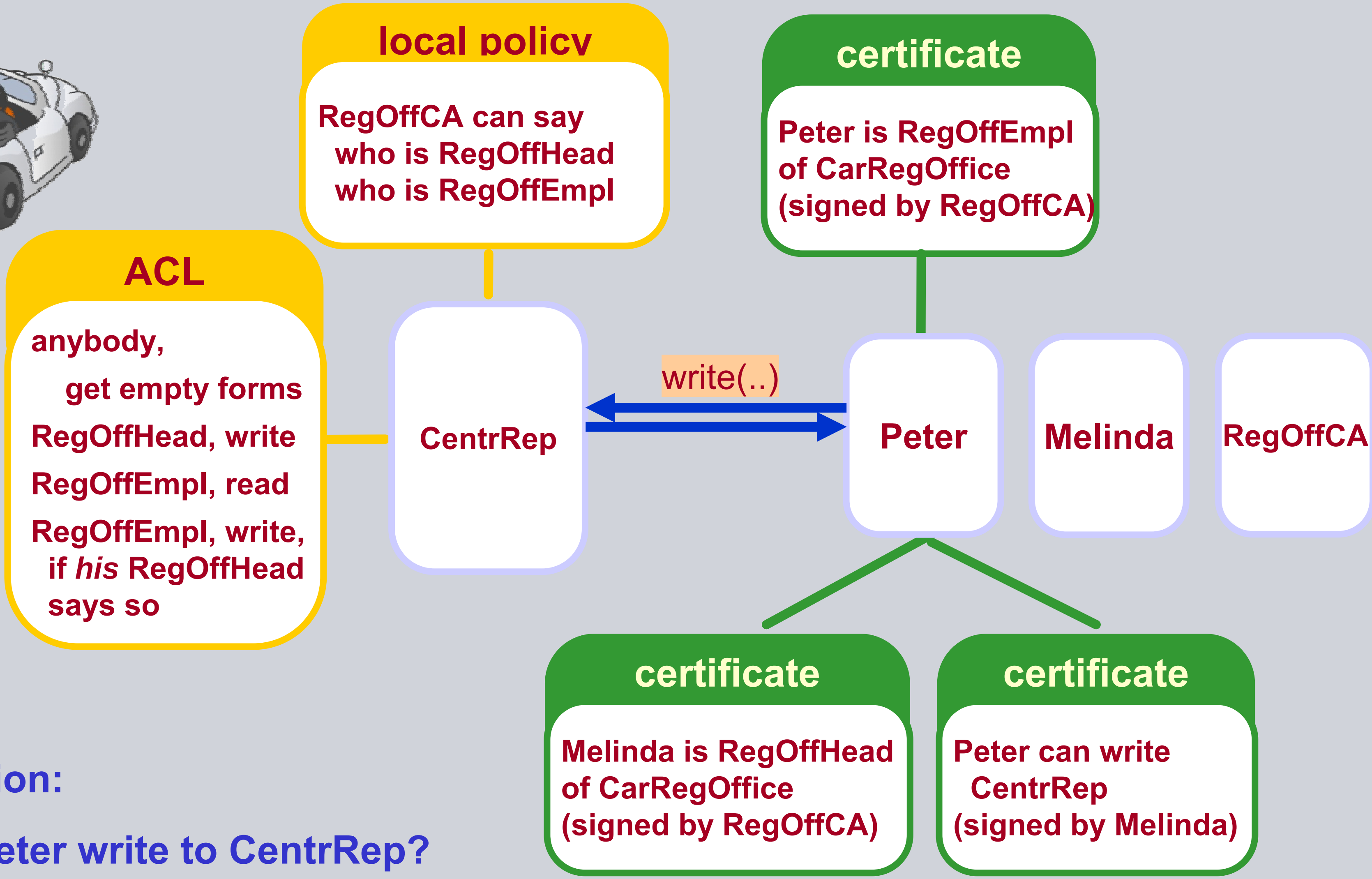
Example 2: ASLan++ model of PTD Application A2

```
entity A2 (Actor: agent, TS2: agent) {    % Application 2, connected with Trust Server 2
  symbols
    C0,C,A1: agent;
    CryptedOrder, Order, Details, Results, TaskHandle, ADT, MAC: message;
    SKey: symmetric_key;
  body { while (true) {
    select {
      % A2 receives (via some C0) a package from some A1. This package includes encrypted and
      % hashed information. A2 needs the corresponding key and the Authorization Decision Token.
      on (?C0 -> Actor: (?A1.Actor.?TaskHandle.?CryptedOrder).?MAC): {
        % A2 contacts its own ticket server (TS2) and requests the secret key SKey and the ADT.
        Actor *->* TS2: TaskHandle;
      }
        % A2 receives from A1 the SKey and checks if the decrypted data corresponds to the hashed data
      on (TS2 *->* Actor: (?ADT.?SKey).TaskHandle  & CryptedOrder = scrypt(SKey,?,?Details.?C)
           & MAC = hash(SKey, A1.Actor.TaskHandle.CryptedOrder)): {
        % A2 does the task requested by A1, then sends to A1 via C the results encrypted with the secret key.
        Results := fresh();  % in general, the result depends on Details etc.
        Actor -> C: Actor.C.A1. scrypt(SKey,Results);
   } } }
  goals
    authentic_C_A2_Details: C  *-> Actor: Details;
    secret_Order: secret (Order, {Actor, A1});
}
```


SIEMENS

Example 3: Electronic Car Registration policies





Question:

May Peter write to CentrRep?

Example 3: On-the-fly inferences via Horn clauses

DKAL-style trust inference, e.g. trust application:

```
trustapp(P,Q,AnyThing):
  P->knows(AnyThing) :-
    P->trusts(Q,AnyThing) &
    P->knows(Q->said(AnyThing));
```

Basic facts, e.g. the central repository fully trusts the CA

```
centrRepTrustCA(AnyThing):
  centrRep->trusts(theCA,AnyThing);
```

State-dependent (evolving) facts, e.g. department head manages a set of trusted employees:

```
trustedEmplsCanStoreDoc(Head): forall Empl.
  Head->knows(Empl->canStoreDoc) :-
    contains(TrustedEmpls, Empl);
```

Use of certificates, e.g. the central repository trusts the department head on employee's rights:

```
centrRepTrustHead(Head,Empl):
  centrRep->trusts(Head,Empl->canStoreDoc) :-
    centrRep->knows(theCA->said(Head->hasRole(head))) &
    centrRep->knows(theCA->said(Empl->hasRole(employee)));
```

Backup slides: ASLan

Semantics of channel goals as LTL formulas

A channel goal requiring authentication, directedness, freshness, and confidentiality:

```
secure_Alice_Payload_Bob: A *->>* B: Payload;
```

On the sender side: `Actor -> B: ...Payload...;`

```
witness(Actor,B,auth_Alice_Payload_Bob,Payload);
secret(Payload,secr_Alice_Payload_Bob,{Actor,B});
```

On the receiver side: `A -> Actor: ...?Payload...;`

```
request(Actor,A,auth_Alice_Payload_Bob,Payload,IID);
secret(Payload,secr_Alice_Payload_Bob,{A,Actor});
```

Semantics of the **authentication** and **directedness** part:

```
forall A,B,P,M,IID. [] (request(B,A,P,M,IID) =>
 (<-> (witness(A,B,P,M)) | (dishonest(A) & iknows(M)))
```

Semantics of the **freshness** (replay protection) part:

```
forall A,B,P,M,IID IID'. [] (request(B,A,P,M,IID) =>
 (!(<-> (request(B,A,P,M,IID') & !(IID=IID')) | dishonest(A)))
```

Semantics of the **confidentiality** part:

```
forall M,P,As. [] ((secret(M,P,As) & iknows(M)) => contains(i, As))
```


SIEMENS

Optimization: Merging transitions on translation

A series of transmission and internal computation ASLan++ commands like

```
receive(A, ?M);
N := fresh();
send(A, N);
```

could bet translated into individual ASLan transitions like:

```
state_entity(Actor, IID, 1, dummy, dummy) . iknows(M) =>
state_entity(Actor, IID, 2, M    , dummy)

state_entity(Actor, IID, 2, M    , dummy) =[exists N]=>
state_entity(Actor, IID, 3, M    , N    )

state_entity(Actor, IID, 3, M    , N    ) =>
state_entity(Actor, IID, 4, M    , N    ) . iknows(N)
```

but can be `compressed´ into a single atomic ASLan transition:

```
state_entity(Actor, IID, 1, dummy, dummy) . iknows(M) =[exists N]=>
state_entity(Actor, IID, 4, M    , N    ) . iknows(N)
```

Even internal computations containing loops etc. can be `glued together´ to avoid interleaving.
This dramatically reduces the search space because a lot of useless branching is avoided.

Example 1: ASLan model of NSPK (1): types, functions

```
% Specification: NSPK
% Channel model: CCM
% Goals as attack states: yes
% Orchestration client: N/A
% Horn clauses level: ALL
% Optimization level: LUMP
% Stripped output (no comments and line information): no
section signature:
    message > text
    ak : agent -> public_key
    ck : agent -> public_key
    defaultPseudonym : agent -> agent
    descendant : nat * nat -> fact
    dishonest : agent -> fact
    isAgent : agent -> fact
    pk : agent -> public_key
    secr_Alice_Bob_PayloadA_set : nat -> set(agent)
    secr_Bob_Alice_PayloadB_set : nat -> set(agent)
    secret_Na_set : nat -> set(agent)
    secret_Nb_set : nat -> set(agent)
    sign : private_key * message -> message
    state_Alice : agent * nat * nat * agent * text * text * text * text -> fact
    state_Bob : agent * nat * nat * agent * text * text * text * text -> fact
    state_Environment : agent * nat * nat -> fact
    state_Session : agent * nat * nat * agent * agent -> fact
```

SIEMENS

Example 1: ASLan model of NSPK (2): constants, variables

```
section types:
     A : agent
     Actor : agent
     Ak_arg_1 : agent
     B : agent
     Ck_arg_1 : agent
     Descendant_Closure_arg_1 : nat
     Descendant_Closure_arg_2 : nat
     Descendant_Closure_arg_3 : nat
     E_S_A_Actor : agent
     E_S_A_B : agent
     E_S_A_IID : nat
     E_S_A_SL : nat
     E_S_Actor : agent
     E_S_B_A : agent
     E_S_B_A_1 : agent
     E_S_B_A_2 : agent
     E_S_B_Actor : agent
     E_S_B_IID : nat
     E_S_B_Na : text
     E_S_B_Na_1 : text
     E_S_B_Nb : text
     E_S_B_Nb_1 : text
     E_S_B_PayloadA : text
     E_S_B_PayloadA_1 : text
E_S_B_PayloadB : text
E_S_B_PayloadB_1 : text
E_S_B_SL : nat
E_S_IID : nat
E_S_SL : nat
E_aABPA_IID : nat
E_aABPA_Msg : message
E_aABPA_Req : agent
E_aABPA_Wit : agent
E_aBAPB_IID : nat
E_aBAPB_Msg : message
E_sABPA_Knowers : set(agent)
E_sABPA_Msg : message
E_sBAPB_Knowers : set(agent)
E_sBAPB_Msg : message
E_sN_Knowers : set(agent)
E_sN_Msg : message
IID : nat
IID_1 : nat
IID_2 : nat
IID_3 : nat
IID_4 : nat
Knowers : set(agent)
Msg : message
Na : text
Na_1 : text
Nb : text
Nb_1 : text
PayloadA : text
PayloadA_1 : text
PayloadB : text
PayloadB_1 : text
Pk_arg_1 : agent
Req : agent
SL : nat
Sign_arg_1 : private_key
Sign_arg_2 : message
Wit : agent
a : agent
atag : text
auth_Alice_Bob_PayloadA
       : protocol_id
auth_Bob_Alice_PayloadB
       : protocol_id
b : agent
ctag : text
dummy_agent : agent
dummy_nat : nat
dummy_text : text
false : fact
secr_Alice_Bob_PayloadA
       : protocol_id
secr_Bob_Alice_PayloadB
       : protocol_id
secret_Na : protocol_id
secret_Nb : protocol_id
stag : text
true : fact
```

Example 1: ASLan model of NSPK (3): initial state, clauses

```
section inits:

initial_state init :=
    dishonest(i).
    iknows(a).
    iknows(atag).
    iknows(b).
    iknows(ctag).
    iknows(i).
    iknows(inv(ak(i))).
    iknows(inv(ck(i))).
    iknows(inv(pk(i))).
    iknows(stag).
    isAgent(a).
    isAgent(b).
    isAgent(i).
    state_Environment(
        dummy_agent,
        dummy_nat, 1).
    true
```

```
section hornClauses:

    hc public_ck(Ck_arg_1) :=
        iknows(ck(Ck_arg_1)) :-
            iknows(Ck_arg_1)

    hc public_ak(Ak_arg_1) :=
        iknows(ak(Ak_arg_1)) :-
            iknows(Ak_arg_1)

    hc public_pk(Pk_arg_1) :=
        iknows(pk(Pk_arg_1)) :-
            iknows(Pk_arg_1)

    hc public_sign(Sign_arg_1, Sign_arg_2) :=
        iknows(sign(Sign_arg_1, Sign_arg_2)) :-
            iknows(Sign_arg_1),
            iknows(Sign_arg_2)

    hc inv_sign(Sign_arg_1, Sign_arg_2) :=
        iknows(Sign_arg_2) :-
            iknows(sign(Sign_arg_1, Sign_arg_2))

    hc descendant_closure(Descendant_Closure_arg_1,
      Descendant_Closure_arg_2, Descendant_Closure_arg_3) :=
        descendant(Descendant_Closure_arg_1,
                   Descendant_Closure_arg_3) :-
            descendant(Descendant_Closure_arg_1,
                       Descendant_Closure_arg_2),
            descendant(Descendant_Closure_arg_2,
                       Descendant_Closure_arg_3)
```


Example 1: ASLan model of NSPK (4): transition rules

```
section rules:

% line 75
% new instance
%     new Session(a,b)
% lumped line 76 (skipped step label 2)
% new instance
%     new Session(a,i)
step step_1_Environment__line_75(Actor, IID, IID_1, IID_2) :=
      state_Environment(Actor, IID, 1)
      =[exists IID_1, IID_2]=>
      descendant(IID, IID_1).
      descendant(IID, IID_2).
      state_Environment(Actor, IID, 3).
      state_Session(dummy_agent, IID_1, 1, a, b).
      state_Session(dummy_agent, IID_2, 1, a, i)

% line 62
% guard
%     !dishonest(A)
% lumped line 63 (skipped step label 2)
% new instance
%     new Alice(A,B)
step step_2_Session__line_62(A, B, E_S_Actor, E_S_IID, IID_3) :=
      not(dishonest(A)).
      state_Session(E_S_Actor, E_S_IID, 1, A, B)
      =[exists IID_3]=>
      descendant(E_S_IID, IID_3).
      state_Alice(A, IID_3, 1, B, dummy_text, dummy_text, dummy_text, dummy_text).
      state_Session(E_S_Actor, E_S_IID, 3, A, B)
```

... (some 5 more pages of rules)

Example 1: ASLan model of NSPK (5): goals

```
section goals:

attack_state auth_Alice_Bob_PayloadA(E_aABPA_IID, E_aABPA_Msg, E_aABPA_Req, E_aABPA_Wit) :=
      not(witness(E_aABPA_Wit, E_aABPA_Req, auth_Alice_Bob_PayloadA, E_aABPA_Msg)).
      request(E_aABPA_Req, E_aABPA_Wit, auth_Alice_Bob_PayloadA, E_aABPA_Msg, E_aABPA_IID) &
      not(equal(i, E_aABPA_Wit))

attack_state auth_Bob_Alice_PayloadB(E_aBAPB_IID, E_aBAPB_Msg, Req, Wit) :=
      not(witness(Wit, Req, auth_Bob_Alice_PayloadB, E_aBAPB_Msg)).
      request(Req, Wit, auth_Bob_Alice_PayloadB, E_aBAPB_Msg, E_aBAPB_IID) &
      not(equal(i, Wit))

attack_state secr_Alice_Bob_PayloadA(E_sABPA_Knowers, E_sABPA_Msg) :=
      iknows(E_sABPA_Msg).
      not(contains(i, E_sABPA_Knowers)).
      secret(E_sABPA_Msg, secr_Alice_Bob_PayloadA, E_sABPA_Knowers)

attack_state secr_Bob_Alice_PayloadB(E_sBAPB_Knowers, E_sBAPB_Msg) :=
      iknows(E_sBAPB_Msg).
      not(contains(i, E_sBAPB_Knowers)).
      secret(E_sBAPB_Msg, secr_Bob_Alice_PayloadB, E_sBAPB_Knowers)

attack_state secret_Na(Knowers, Msg) :=
      iknows(Msg).
      not(contains(i, Knowers)).
      secret(Msg, secret_Na, Knowers)

attack_state secret_Nb(E_sN_Knowers, E_sN_Msg) :=
      iknows(E_sN_Msg).
      not(contains(i, E_sN_Knowers)).
      secret(E_sN_Msg, secret_Nb, E_sN_Knowers)
```

