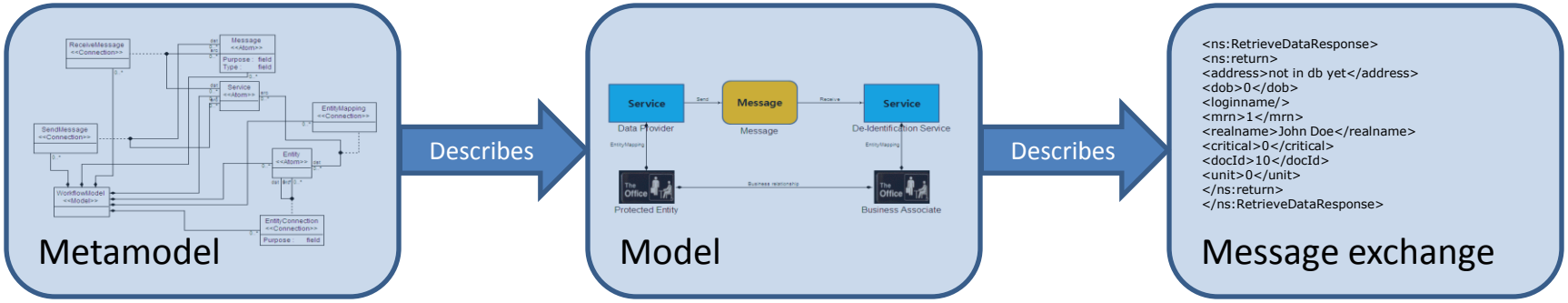
Integration of Clinical Workflows with Privacy Policies on a Common Semantic Platform

Jan Werner, Bradley Malin, Yonghwan Lee,
Akos Ledeczi, Janos Sztipanovits
Institute for Software Integrated System
Vanderbilt University, Nashville, TN

presented by Janos Mathe

2nd International Workshop on Model-Based Design of Trustworthy Health Information Systems

Model Based Design for Clinical Workflows



- Metamodel of a workflow language
- Description of the modeling abstractions eg. Messages, Services and Composition Rules
- Definition of a workflow domain

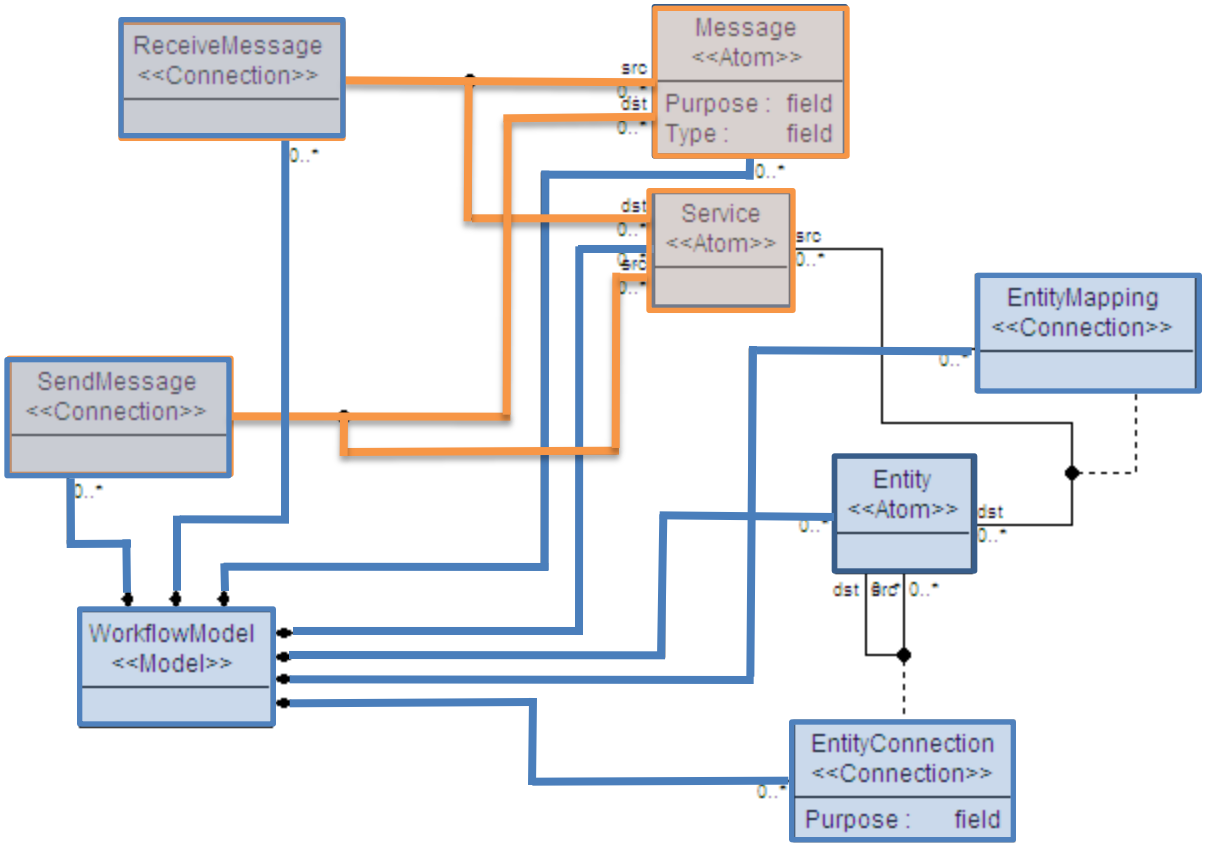
- Model of a workflow
- Representation of message exchange patterns, definition of services and messages in a clinical setting eg. Data Provider Service, Medical Record Message
- Definition of communication protocol

- Messages in runtime environment
- Service invocations and replies with requested data eg. Patient record of 'John Doe'
- Instance of communication pattern

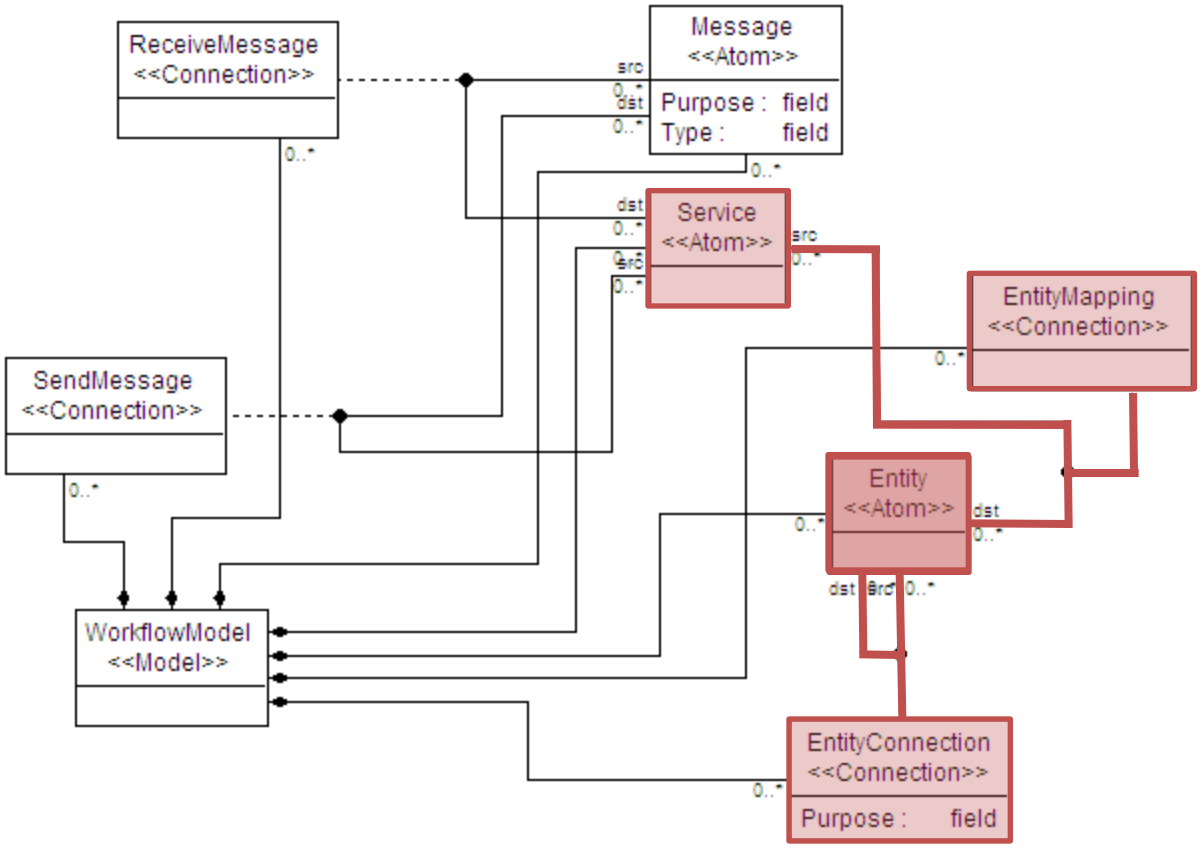
Privacy Policies

- Privacy Policies used in this presentation:
  - *A covered entity may send protected health information to a business partner for de-identification purposes only if there exists a contractual agreement between the communicating entities.*
  - *Access to the patient's medical record should only be granted to primary care physicians listed in medical record, or in case of emergency situation access should be provided to any physician following the "Break Glass" policy*

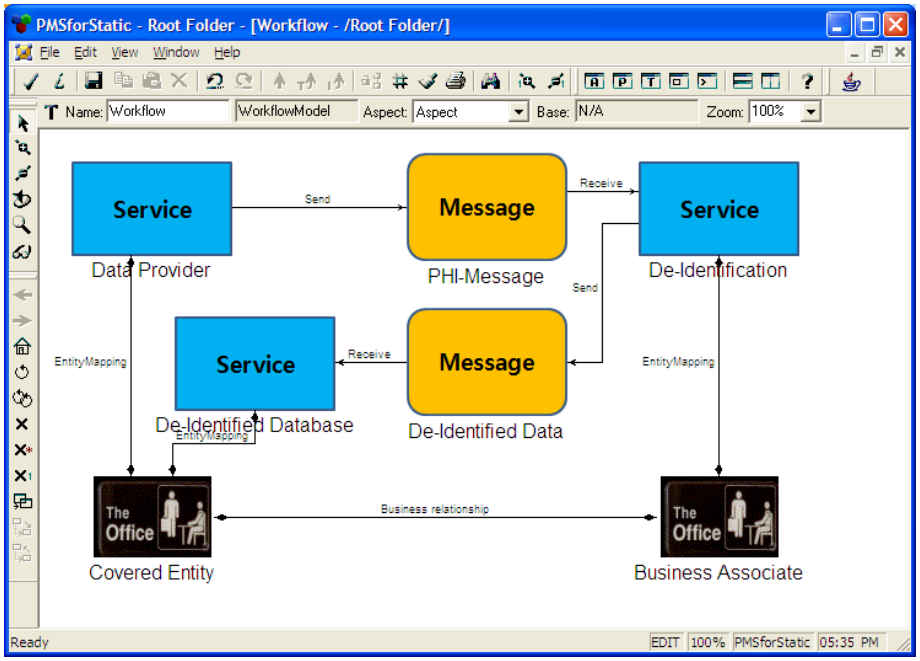
Design of a simple workflow language



Design of a simple workflow language



Model of a workflow



Workflow model

Data provider sends the sensitive data for de-identification. De-identified data is finally stored in local database

Privacy Policy

Covered Entity sends the Protected Health Information for de-identification to Business Associate and receives back the de-identified data A covered entity may send protected health information to a business partner for de-identification purposes only if there exists a contractual agreement between the communicating entities.

Integration using Structural Semantics Approach

- How to formally represent a domain?
- A domain D is given by
  - An alphabet Σ
  - A set of n-ary function symbols Y
  - A set of model realizations $R_{\mathrm{Y}} = \mathrm{H}(\Sigma, \mathrm{Y})$
  - A set of constraints C such that $r \in R_{\mathrm{Y}}, r \succ C, \rightarrow r \in D$
- Constraints are given as proofs

$$(r \succ C) \Leftrightarrow (\exists x \in r, r \cap C \rightarrow \mathit{wellform}(x))$$

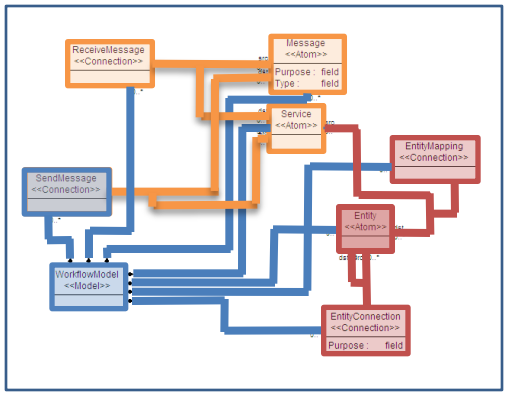
$$(r \succ C) \Leftrightarrow (\neg \exists x \in r, r \cap C \rightarrow \mathit{malform}(x))$$

Model transformation and interpretation

GME

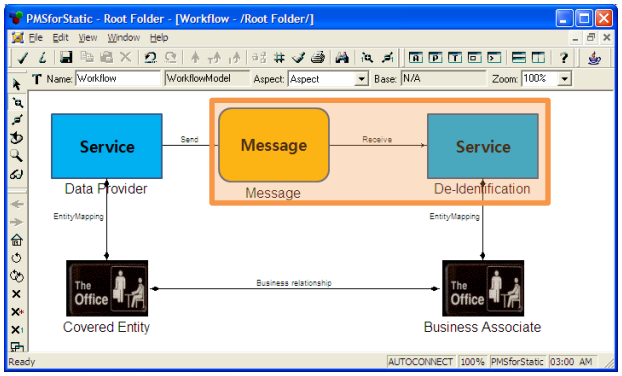
Horn domain

Meta level




```
canconn('receivemessage',X,Y) :-
    message(X), service(Y).
malform(receivemessage(N,X,Y)):-
    receivemessage(N,X,Y), \+canconn('receivemessage',X,Y)
cancontain(X,Y) :-
    sendmessage(X), workflowmodel(Y).
malform(purpose(Y,V)) :-
    purpose(Y,V), purpose(Y,W), (V \== W).
malform(purpose(Y,V)) :-
    purpose(Y,V), \+entityconnection(Y).
(…)
```

Model level

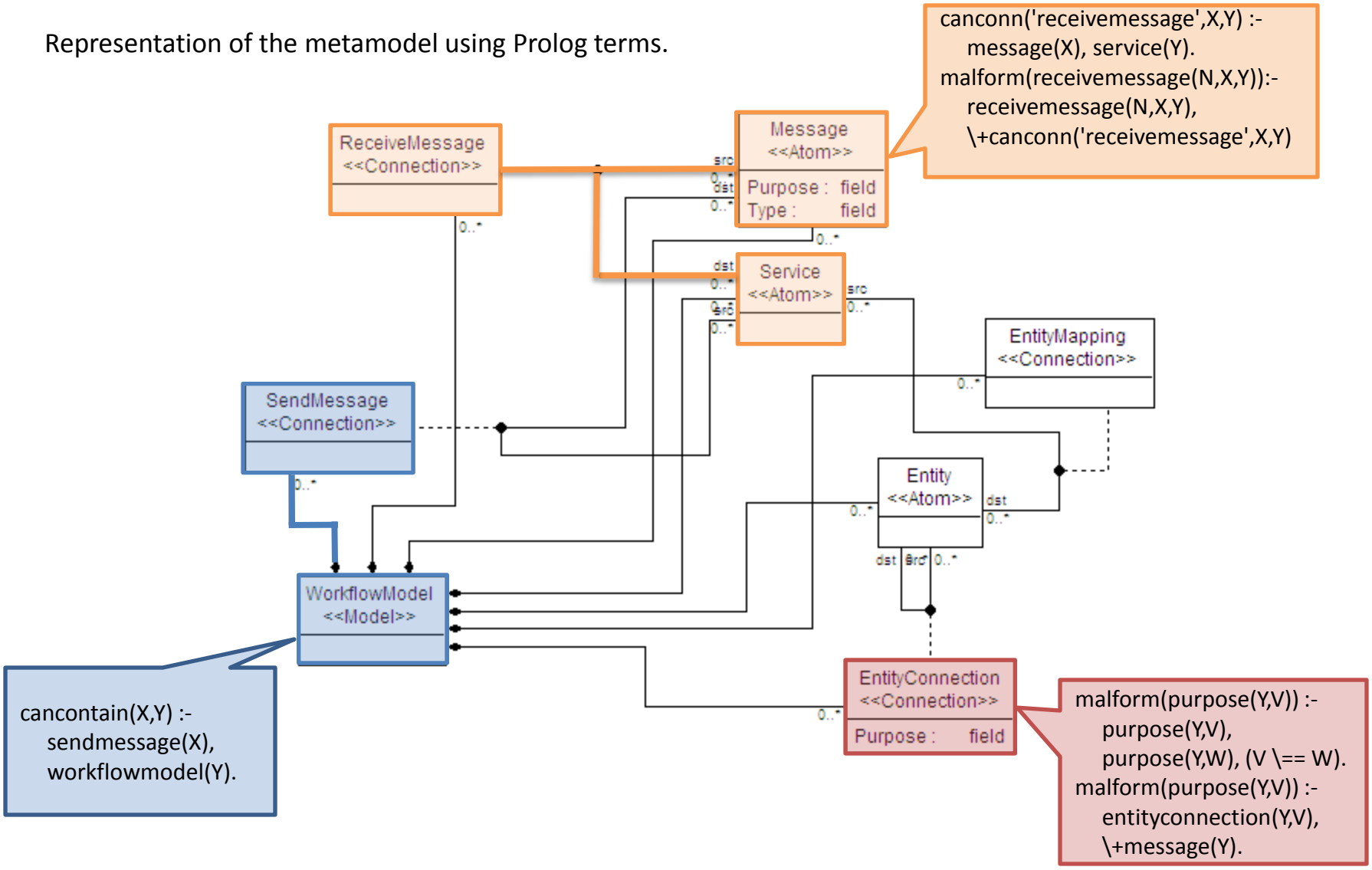



```
message('message_id-0066-00000004’).
service('de-identification _id-0066-00000003').
receivemessage('receivemessage_id-0068-00000009').
receivemessage('receivemessage_id-0068-00000009','message_id-
0066-00000004','de-identification_id-0066-00000003').

workflowmodel('workflow_id-0065-00000001').
sendmessage('sendmessage_id-0068-00000002').
contains('sendmessage_id-0068-00000002','workflow_id-0065-
00000001').
(…)
```

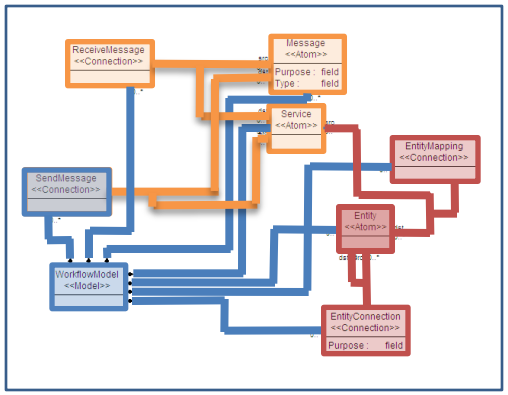
Translation of an example workflow metamodel

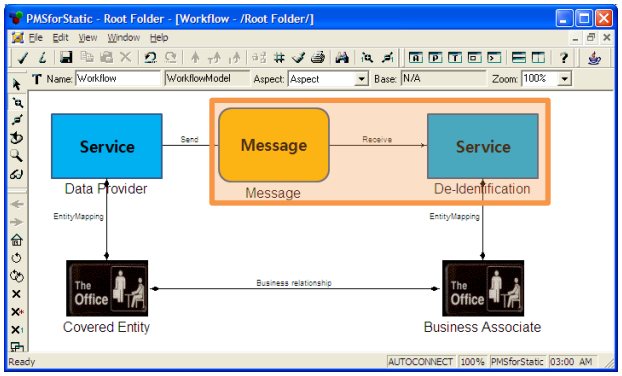
Representation of the metamodel using Prolog terms.



Model transformation and interpretation

	GME	Horn domain

Meta level		canconn('receivemessage',X,Y) :-
		  message(X), service(Y).
		malform(receivemessage(N,X,Y)):-
		  receivemessage(N,X,Y),
		  \+canconn('receivemessage',X,Y)
		cancontain(X,Y) :-
		  sendmessage(X), workflowmodel(Y).
		malform(purpose(Y,V)) :-
		  purpose(Y,V), purpose(Y,W), (V \== W).
		malform(purpose(Y,V)) :-
		  purpose(Y,V), \+entityconnection(Y).
		(…)

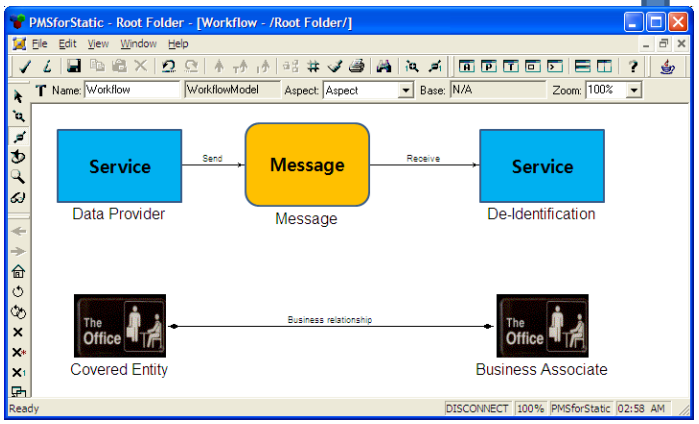
Model level		message('message_id-0066-00000004’). service('de-identification _id-0066-00000003'). receivemessage('receivemessage_id-0068-00000009'). receivemessage('receivemessage_id-0068-00000009','message_id-0066-00000004','de-identification_id-0066-00000003'). workflowmodel('workflow_id-0065-00000001'). sendmessage('sendmessage_id-0068-00000002'). contains('sendmessage_id-0068-00000002','workflow_id-0065-00000001'). (…)

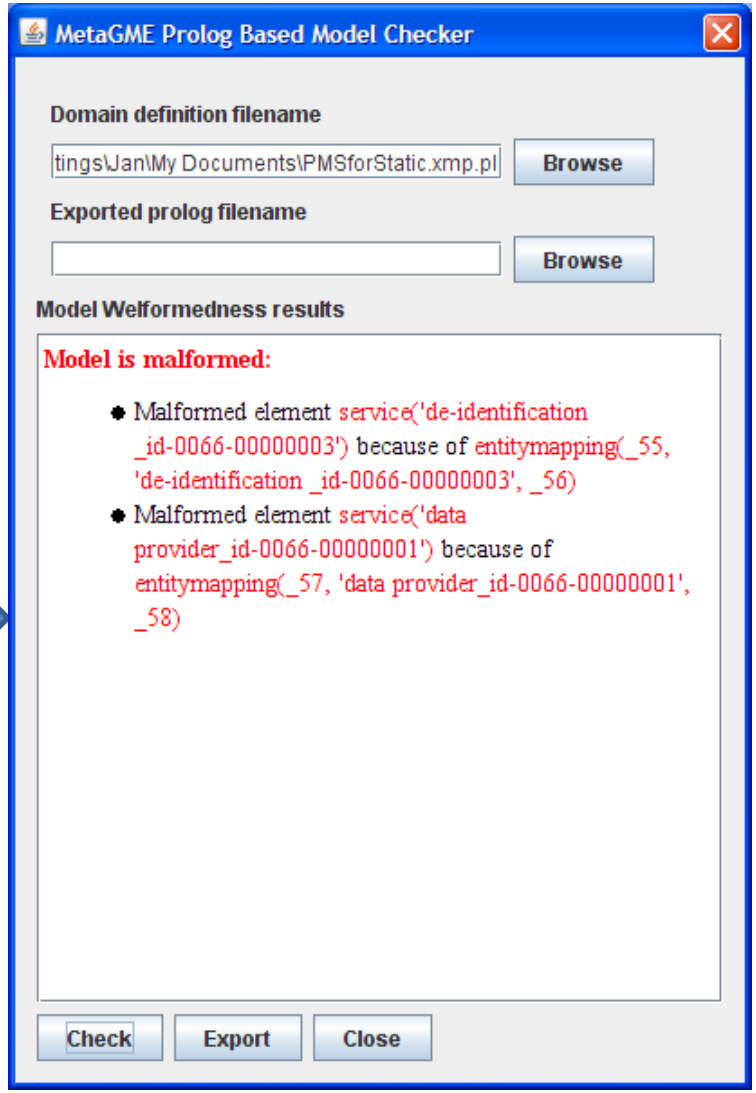
Verification of model wellformedness

Additional constraints
Services have to be mapped to the organizations

```
no_entity_mapping(S,R) :-
    R = entitymapping(_,S,_),
    \+entitymapping(X,S,_).
malform(service(S),R) :- service(S),
    no_entity_mapping(S,R).
```

Malformed model





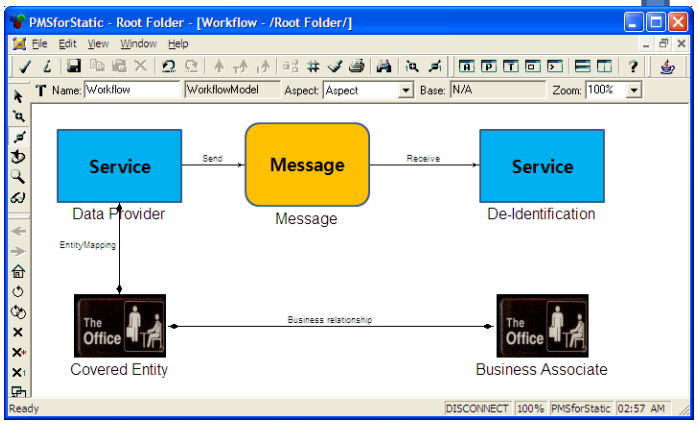
Verification of model wellformedness

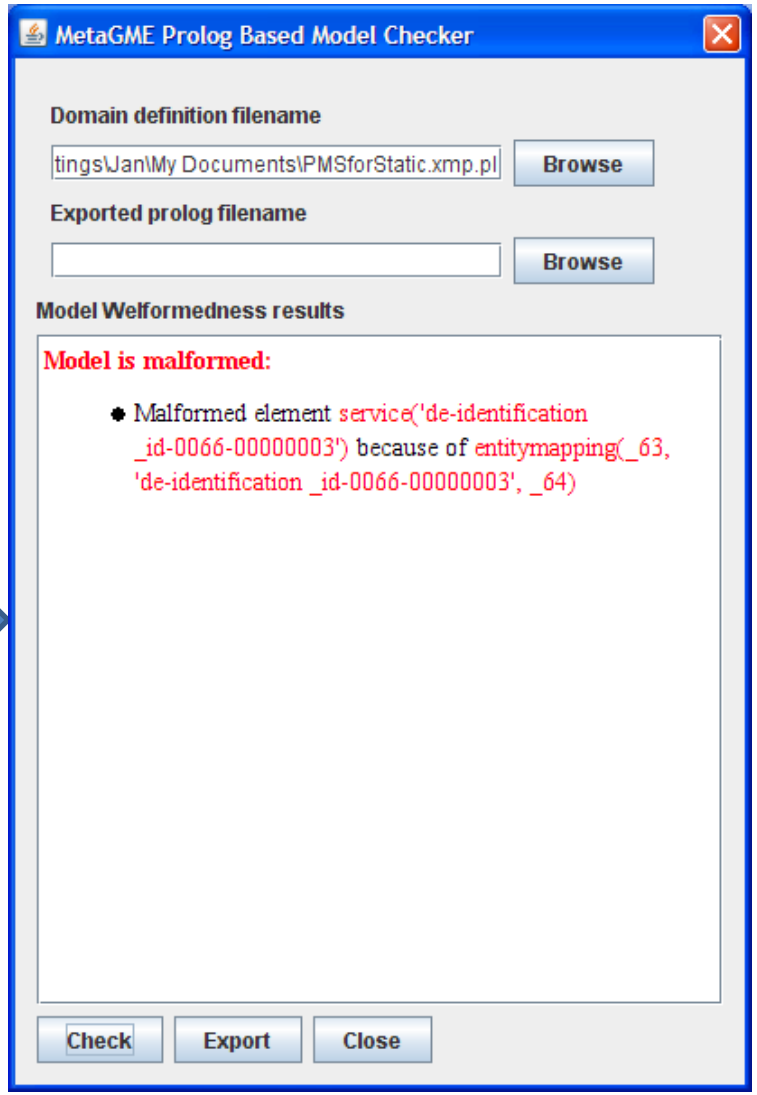
Additional constraints

Services have to be mapped to the organizations

```
no_entity_mapping(S,R) :-
    R = entitymapping(_,S,_),
    \+entitymapping(X,S,_).
malform(service(S),R) :- service(S),
    no_entity_mapping(S,R).
```

Malformed model





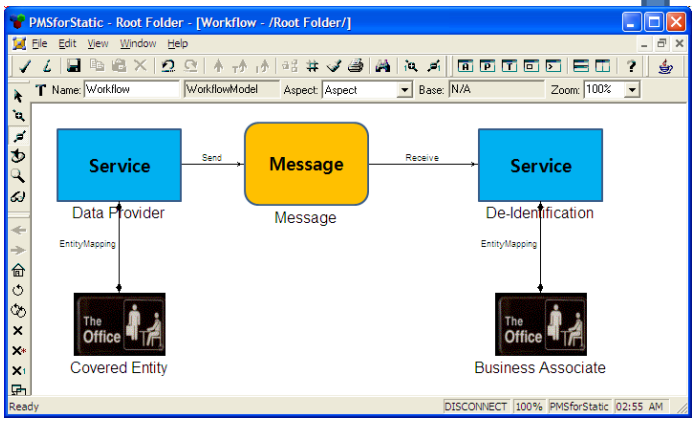
Privacy policy as model constraint

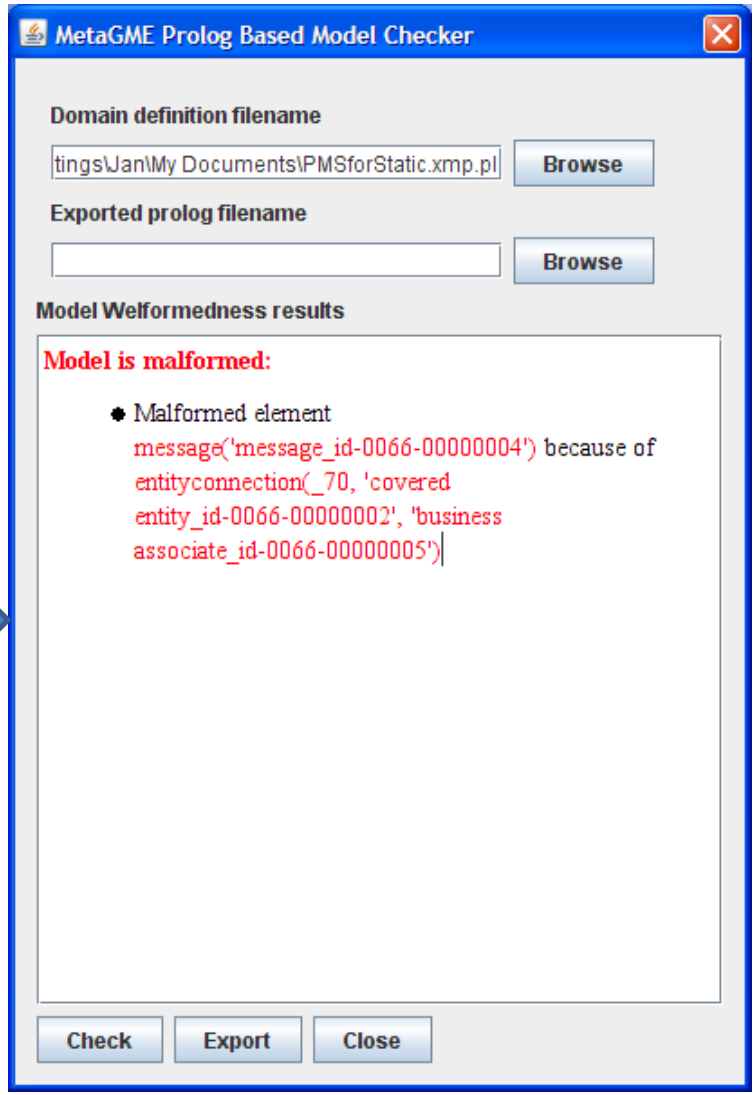
Additional constraints – privacy policy

Covered entity(E1) may send protected health information (M) to business partner (E2) for de-identification only if there exist partner link (EntityConnection) between the entity (E1) and business partner (E2)

```
no_entity_connection(E1,E2,R)  :-
    R = entityconnection(_,E1,E2), (E1\==E2),
    \+ entityconnection(X,E1,E2).
malform(message(M),R) :- message(M),
    sendmessage(MF,S1,M), receivemessage(MF2,M,S2),
    entitymapping(EM1,S1,E1), entitymapping(EM2,S2,E2),
    no_entity_connection(E1,E2,R).
```

Malformed model





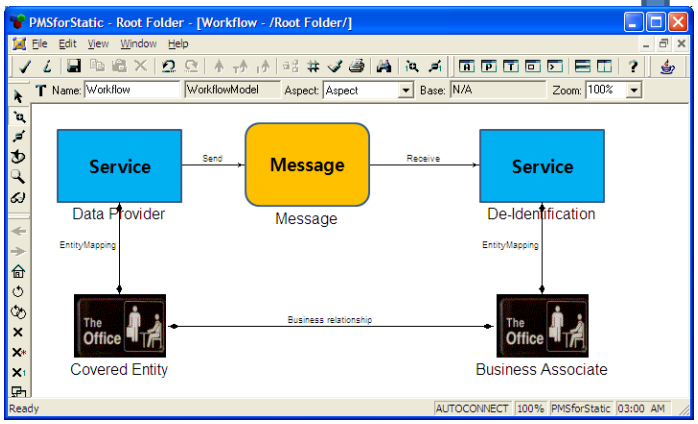
Privacy policy as model constraint

Additional constraints

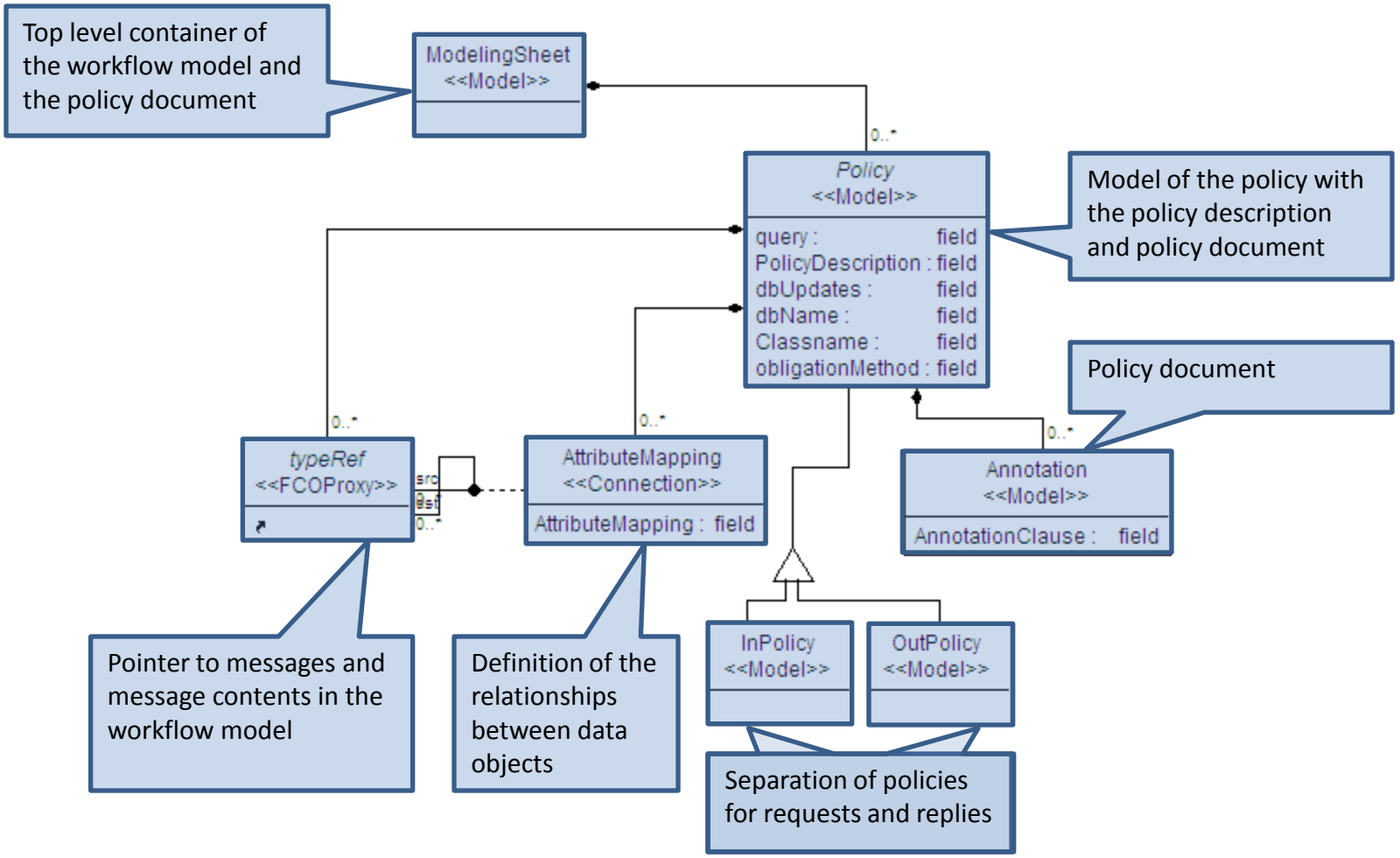
```
no_entity_mapping(S,R) :-
    R = entitymapping(_,S,_),
    \+entitymapping(X,S,_).
malform(service(S),R) :- service(S),
    no_entity_mapping(S,R).

no_entity_connection(E1,E2,R)  :-
    R = entityconnection(_,E1,E2), (E1\==E2),
    \+ entityconnection(X,E1,E2).
malform(message(M),R) :- message(M),
    sendmessage(MF,S1,M), receivemessage(MF2,M,S2),
    entitymapping(EM1,S1,E1), entitymapping(EM2,S2,E2),
    no_entity_connection(E1,E2,R).
```

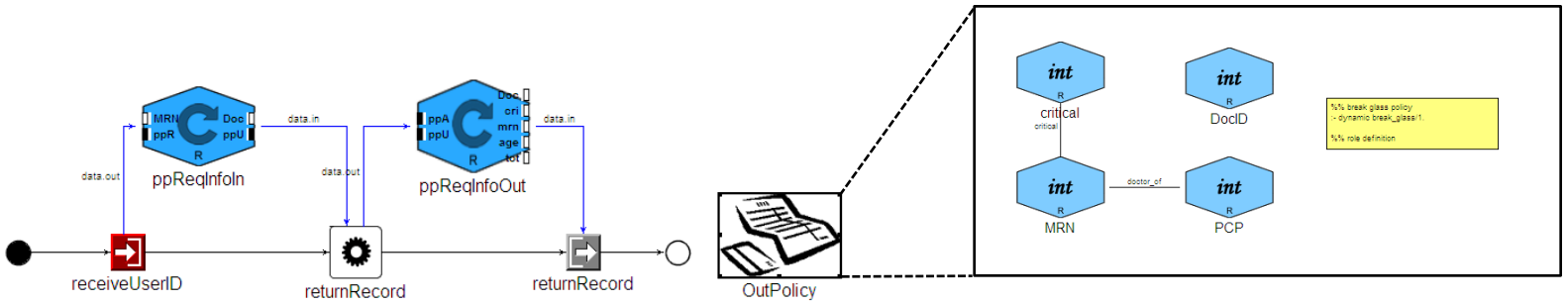
Wellformed model



Design of the policy language



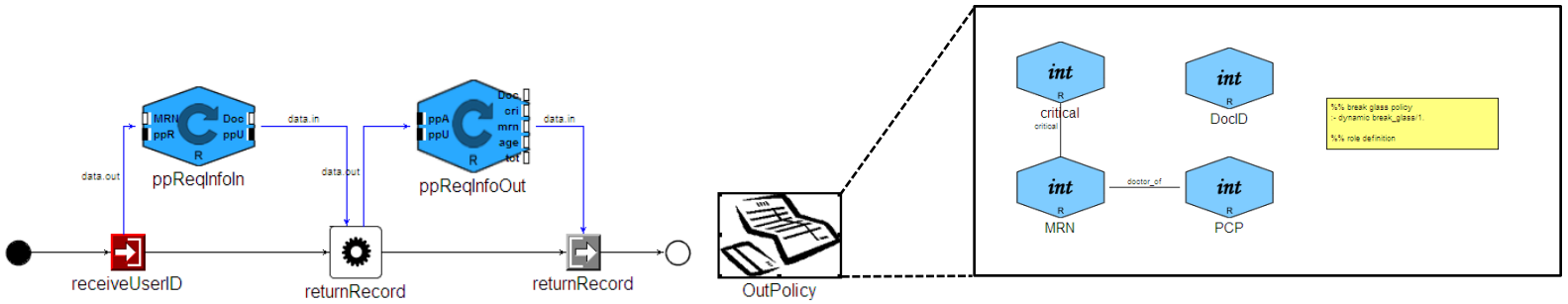
MICIS workflow model



- Workflow model of service providing patient's medical records
- Outgoing message privacy policy:
  - *Access to the patient's medical record should only be granted to primary care physicians listed in medical record, or in case of emergency situation access should be provided to any physician following the "Break Glass" policy*

Generation of runtime enforced policies

Model



Generated documents

Workflow documents

BPEL workflow description

WSDL Web Services description

Deployment Configuration

Policy Description:

Service identifier

Type of the policy (incoming / outgoing)

Description of fields from request required to evaluate the policy

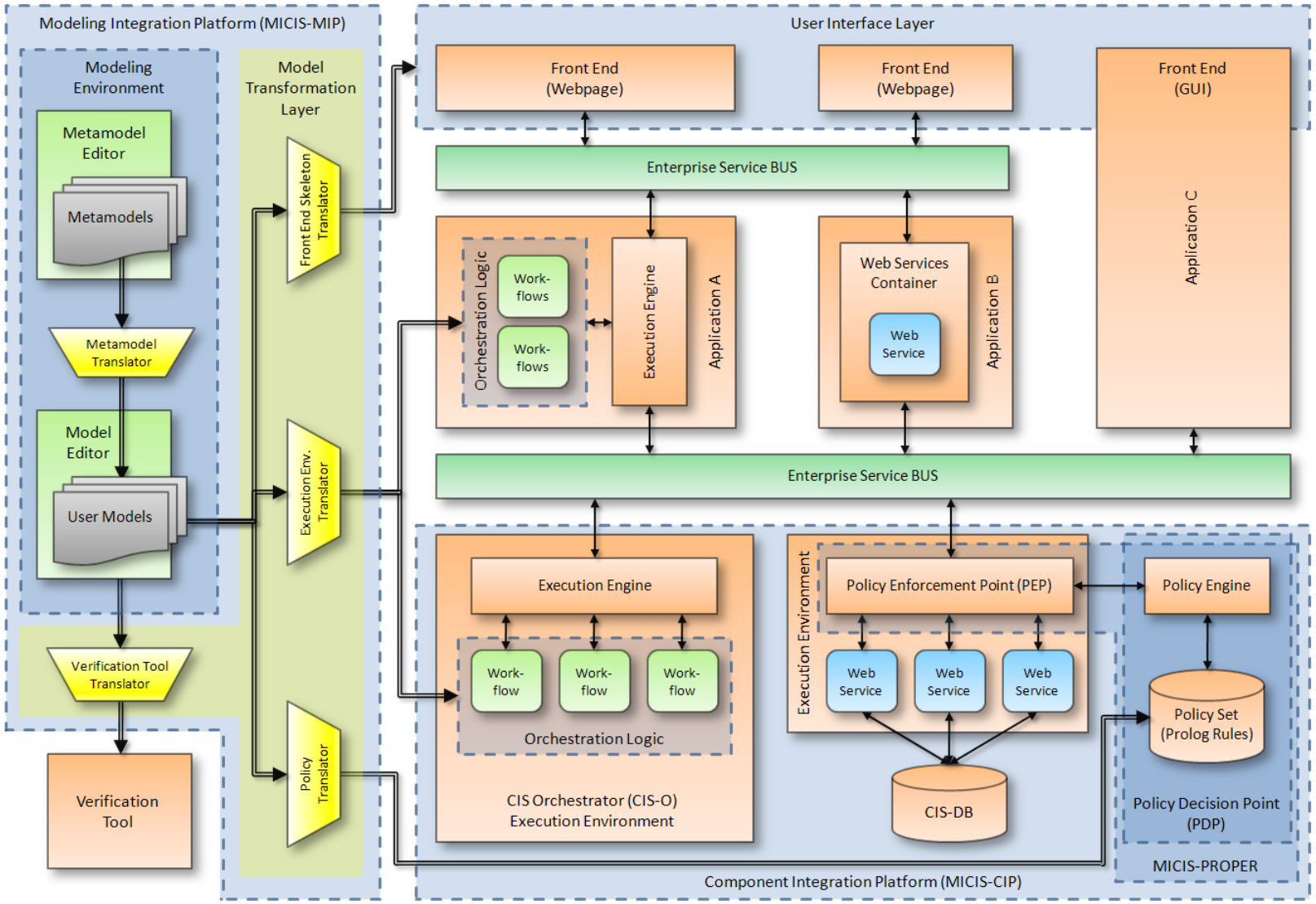
Information on the state of the Decision Engine

Obligations executed upon the service invocation

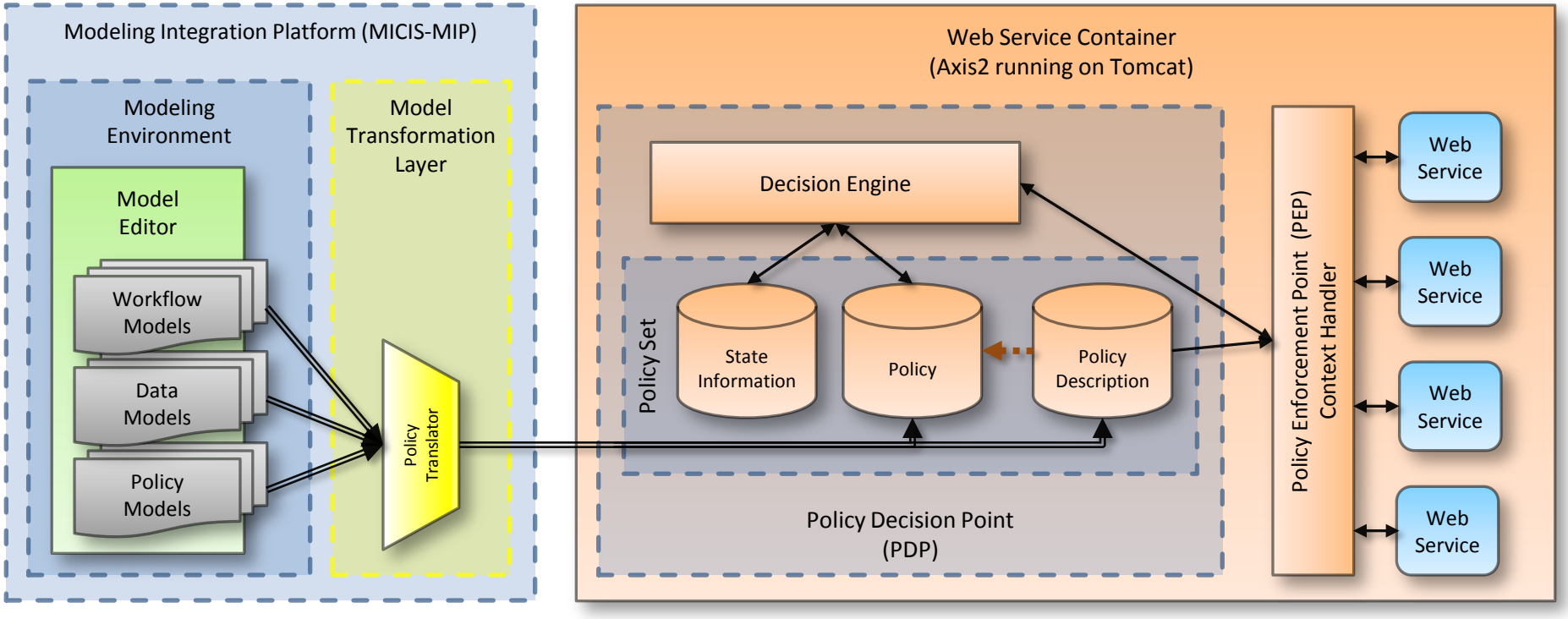
Policy Document:

```
:- dynamic break_glass/1.
:- dynamic treats/2.
:- dynamic critical/2.
:- dynamic retrievedata/2.
retrievedata(RecordNo,DocId):-
                treats(RecordNo, DocId);
                break_glass(RecordNo).
break_glass(RecordNo):-
                critical(RecordNo,X), X>0.
```

MICIS architecture

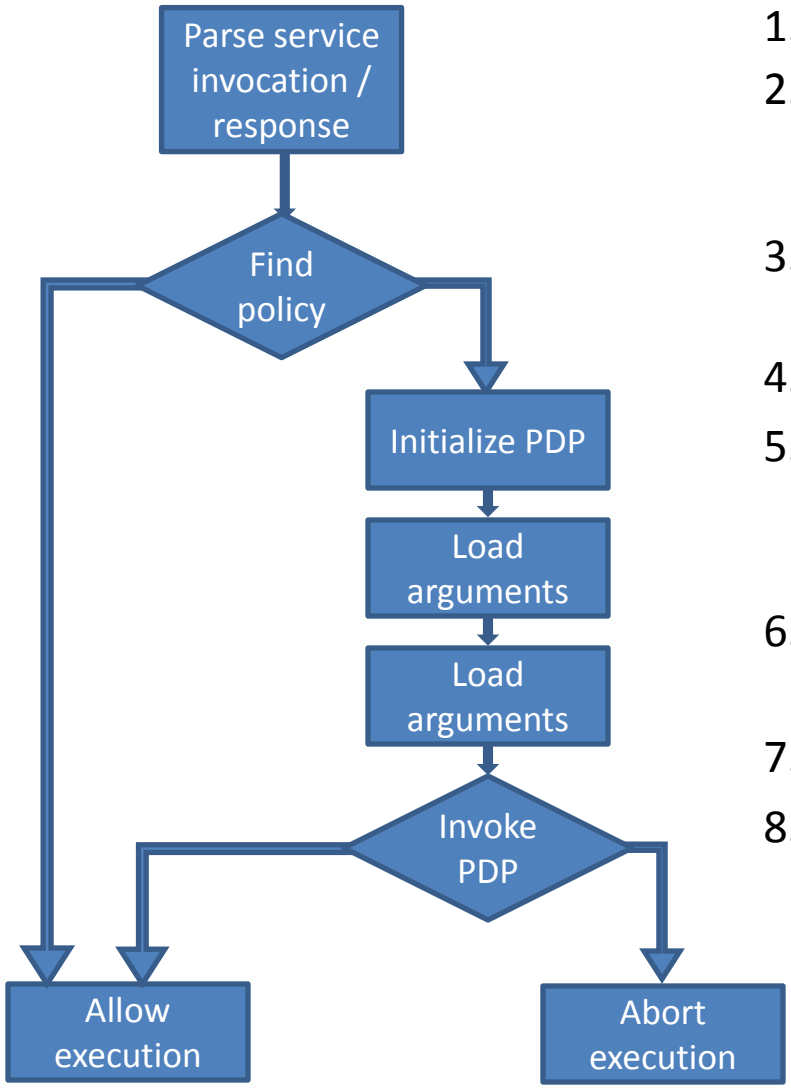


Policy Decision and Enforcement Point



- Invocation of protected services is guarded by the Web Service message interceptor implementing Policy Enforcement Point
- Policy Enforcement Point is driven by the configuration generated from the models (Policy Description).
- Decision Point loads the Policy Documents deployed from the policy models (Policy Store) and the saved state(State Information)

Step by step enforcement of a dynamic policy



1. Parse the service invocation (Req)
2. Using the service ID and the Policy Description (PD) find corresponding policies [Px,…, Py]
3. Based on the service ID initialize the appropriate state of the Decision Engine
4. Load policies into the Decision Engine
5. Based on PD indentify and load the arguments from the Req into the Decision Engine(Prolog)
6. Invoke the Decision Engine to decide on access to protected service
7. Save the new state of Decision Engine
8. Execute the obligations (if specified in PD)

Results

- Framework that unifies description of workflows and policies on common semantic platform
- Prolog Based tool for verification of the models integrated in GME modeling environment
- Policy Enforcement Engine integrated in a Service-Oriented Architecture platform

Future Work

- Classification of HIPAA rules to represent them using Structural constraints on the models
- Generation of the workflow models based on the set of rules and policies