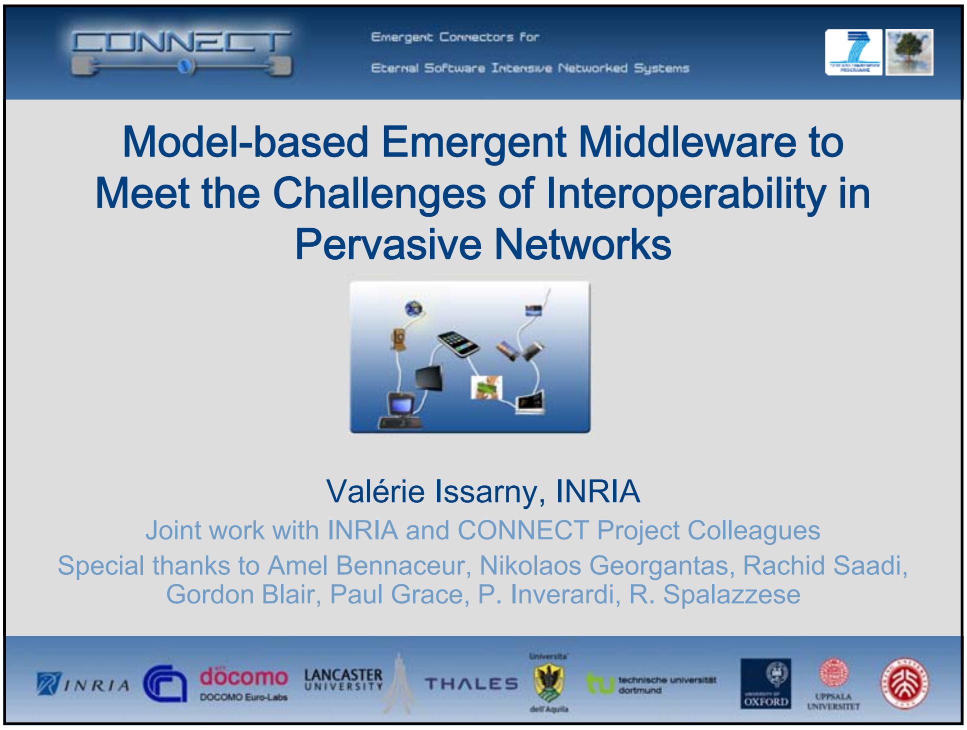


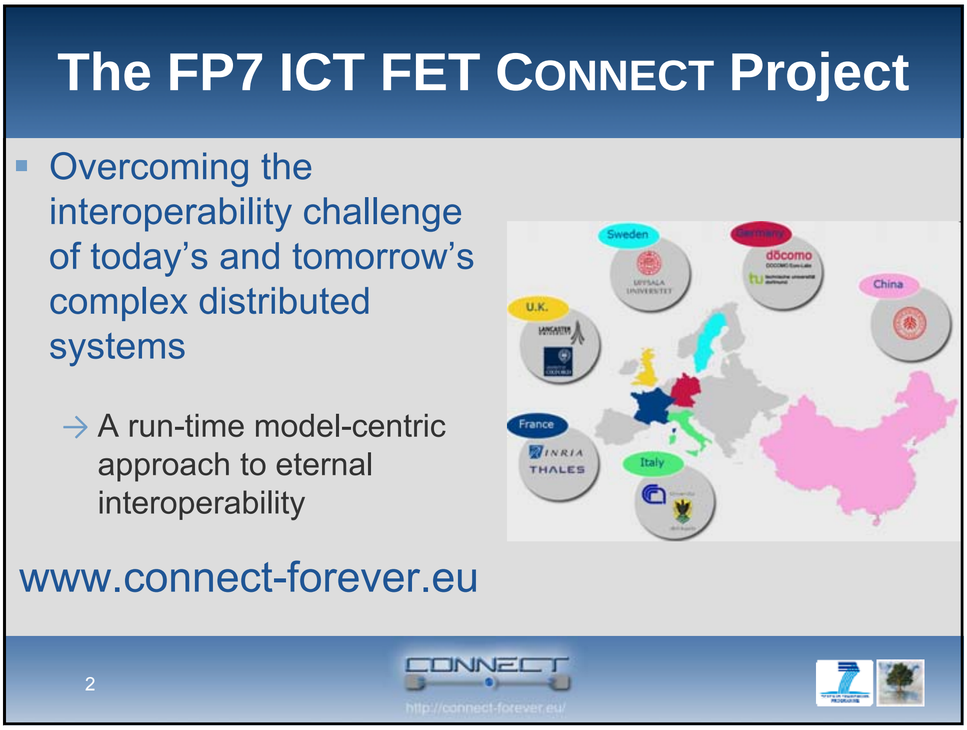
Emergent Connectors for Eternal Software Intensive Networked Systems

Model-based Emergent Middleware to Meet the Challenges of Interoperability in Pervasive Networks

Valérie Issarny, INRIA

Joint work with INRIA and CONNECT Project Colleagues

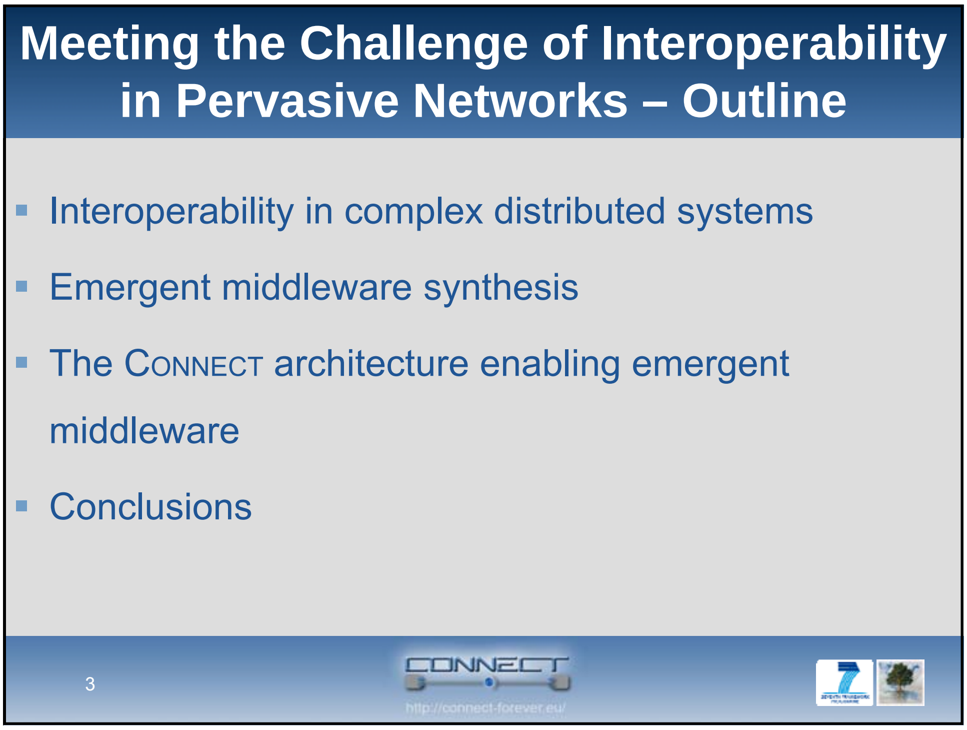
Special thanks to Amel Bennaceur, Nikolaos Georgantas, Rachid Saadi, Gordon Blair, Paul Grace, P. Inverardi, R. Spalazzese



The FP7 ICT FET CONNECT Project

- Overcoming the interoperability challenge of today's and tomorrow's complex distributed systems
  - → A run-time model-centric approach to eternal interoperability

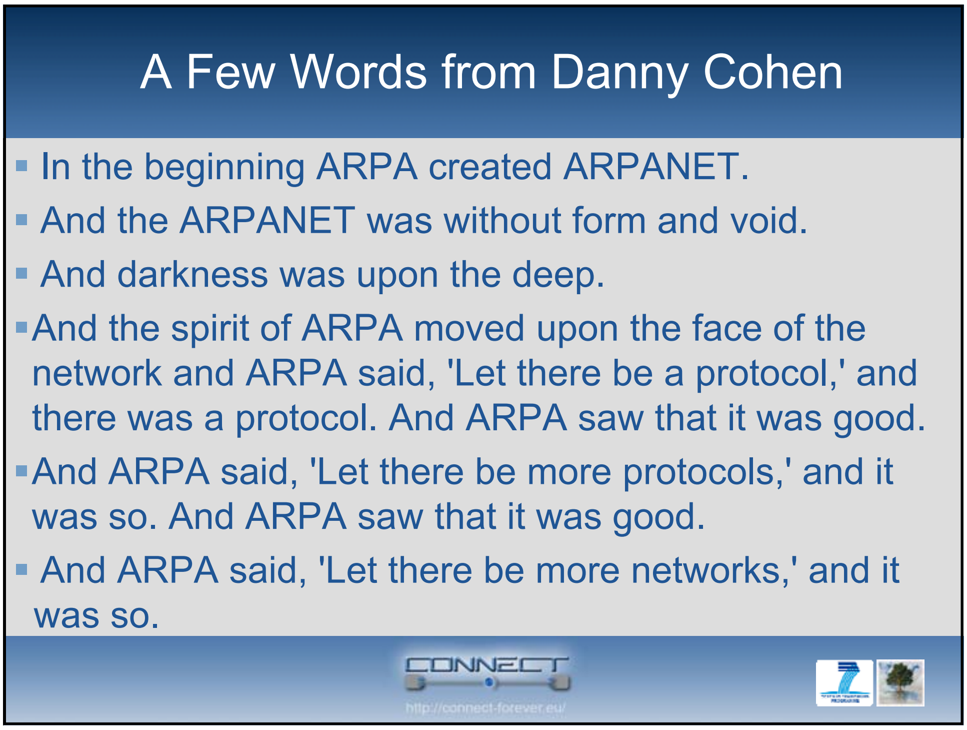
www.connect-forever.eu



Meeting the Challenge of Interoperability in Pervasive Networks – Outline

- Interoperability in complex distributed systems
- Emergent middleware synthesis
- The CONNECT architecture enabling emergent middleware
- Conclusions

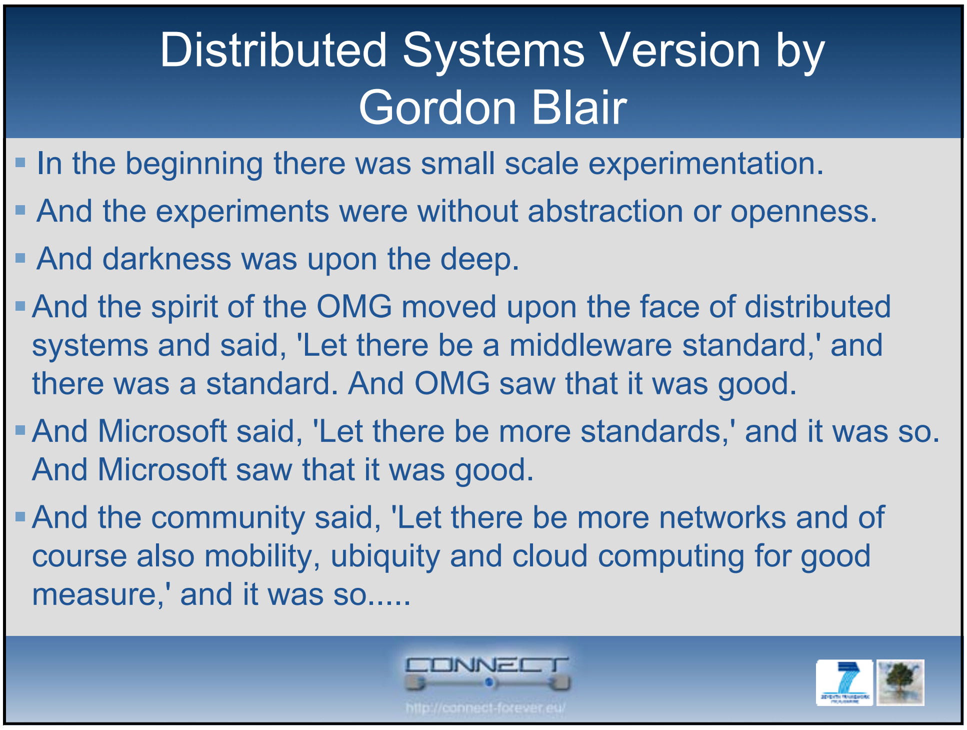
http://connect-forever.eu/



A Few Words from Danny Cohen

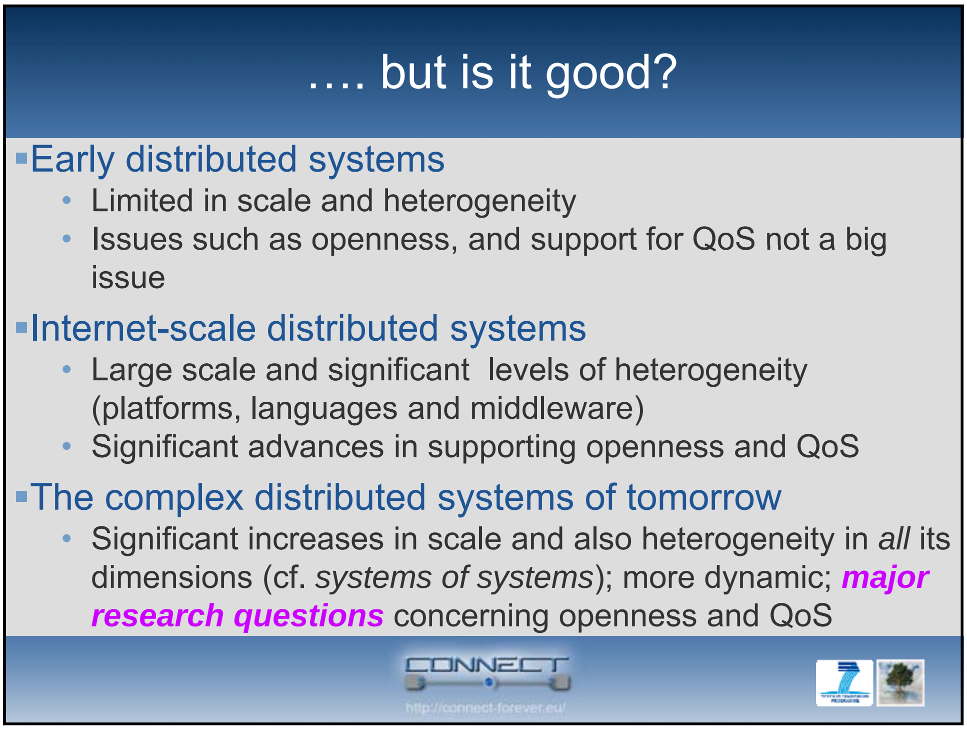
- In the beginning ARPA created ARPANET.
- And the ARPANET was without form and void.
- And darkness was upon the deep.
- And the spirit of ARPA moved upon the face of the network and ARPA said, 'Let there be a protocol,' and there was a protocol. And ARPA saw that it was good.
- And ARPA said, 'Let there be more protocols,' and it was so. And ARPA saw that it was good.
- And ARPA said, 'Let there be more networks,' and it was so.

http://connect-forever.eu/



Distributed Systems Version by Gordon Blair

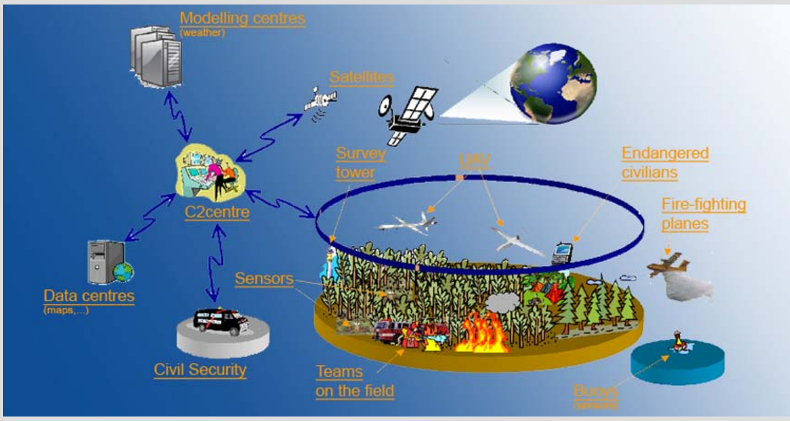
- In the beginning there was small scale experimentation.
- And the experiments were without abstraction or openness.
- And darkness was upon the deep.
- And the spirit of the OMG moved upon the face of distributed systems and said, 'Let there be a middleware standard,' and there was a standard. And OMG saw that it was good.
- And Microsoft said, 'Let there be more standards,' and it was so. And Microsoft saw that it was good.
- And the community said, 'Let there be more networks and of course also mobility, ubiquity and cloud computing for good measure,' and it was so.....



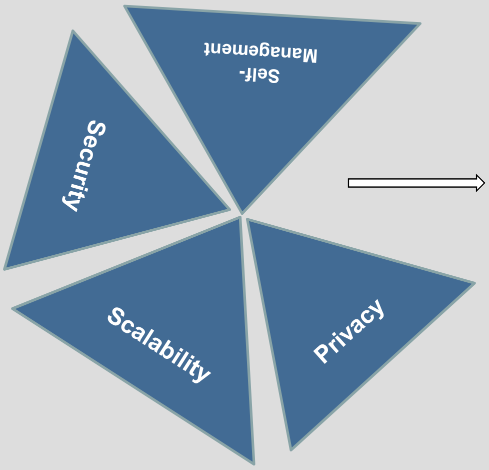
…. but is it good?

- Early distributed systems
  - Limited in scale and heterogeneity
  - Issues such as openness, and support for QoS not a big issue
- Internet-scale distributed systems
  - Large scale and significant levels of heterogeneity (platforms, languages and middleware)
  - Significant advances in supporting openness and QoS
- The complex distributed systems of tomorrow
  - Significant increases in scale and also heterogeneity in *all* its dimensions (cf. *systems of systems*); more dynamic; ***major research questions*** concerning openness and QoS

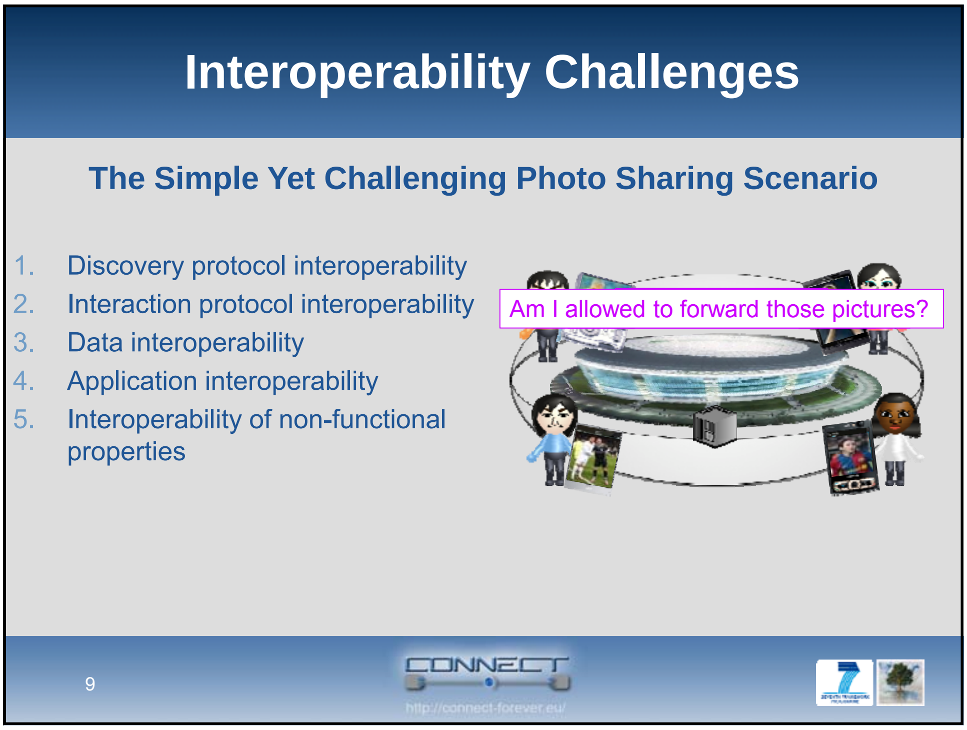
Illustrating the challenges
Global Monitoring for Environment & Security



Interoperability Focus



Tanenbaum & Van Steen:
"the extent by which two implementations of systems from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard".

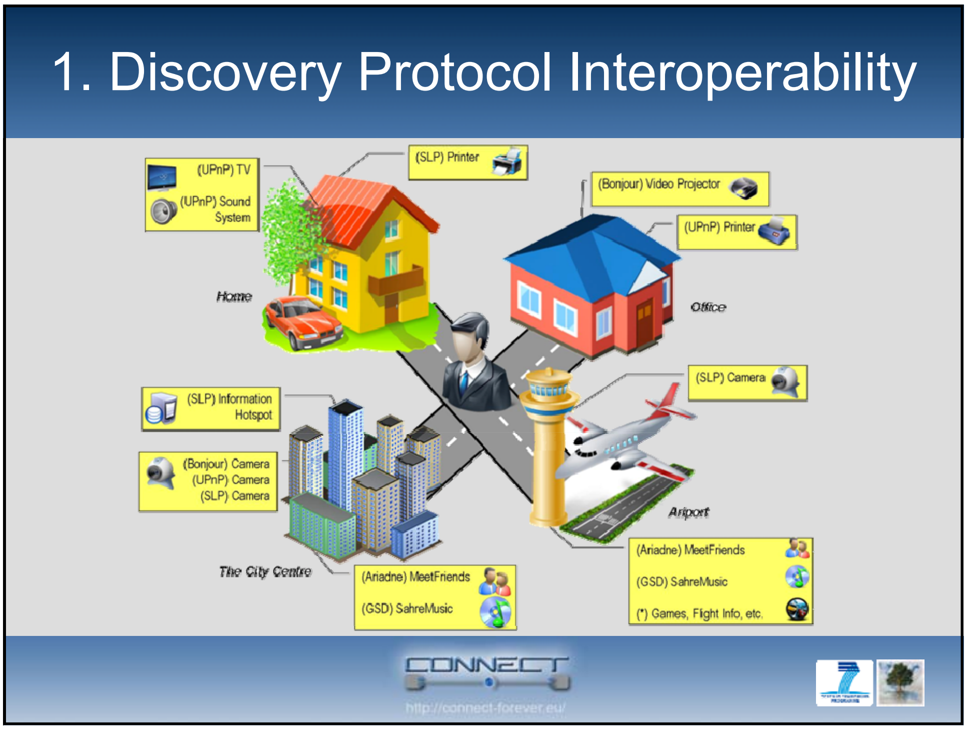


Interoperability Challenges

The Simple Yet Challenging Photo Sharing Scenario

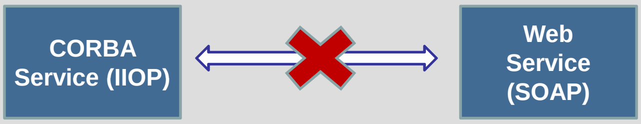
1. Discovery protocol interoperability
2. Interaction protocol interoperability
3. Data interoperability
4. Application interoperability
5. Interoperability of non-functional properties

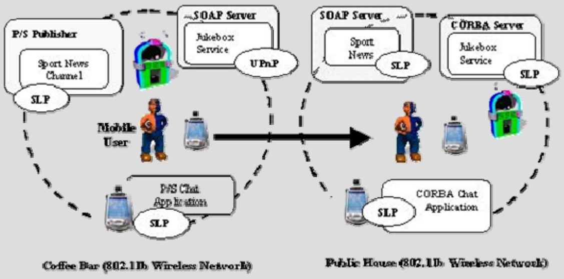
Am I allowed to forward those pictures?



1. Discovery Protocol Interoperability

2. Middleware Protocol Interoperability



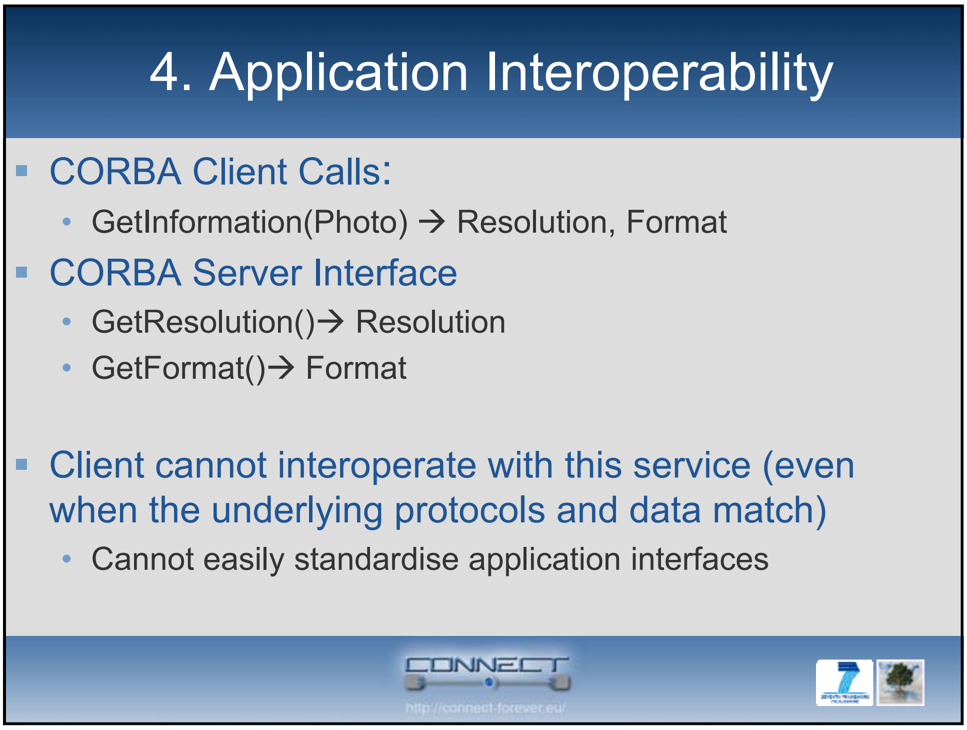


3. Data Interoperability

<photo> <resolution> low </resolution> <format> png </format> </photo>	<photo> <resolution> <value> 72x72 </value> <unit> DPI </unit> </resolution> <format> jpg</format> </photo>

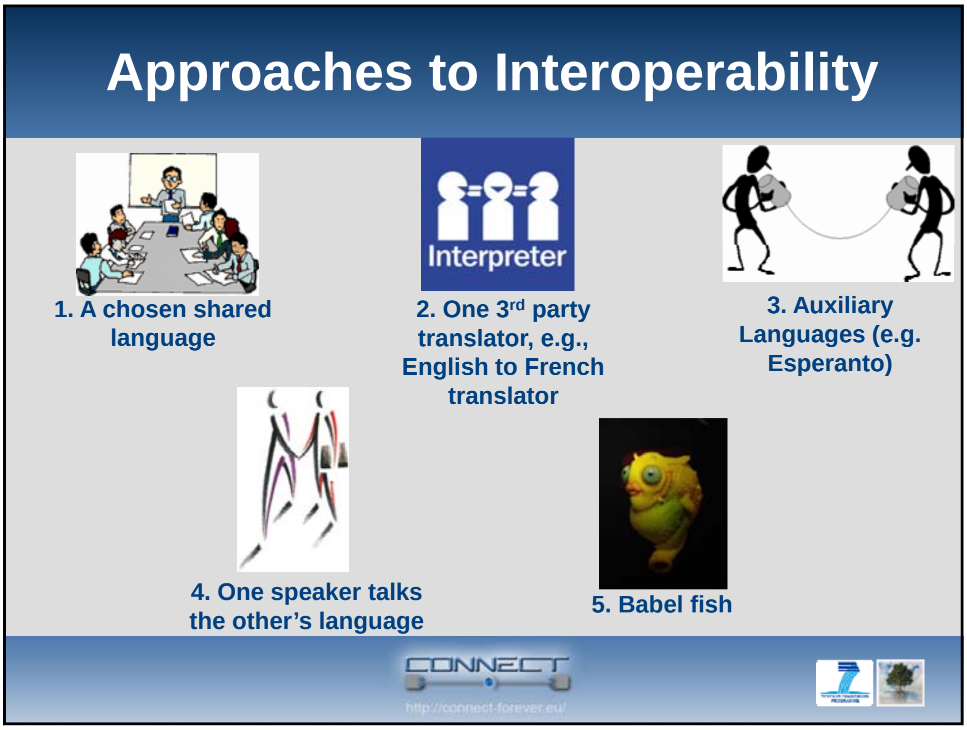
<photo> <resolution> low </resolution> <format> png</format> </photo>	photo(low, png)

<photo> <resolution> low </resolution> <format> png</format> </photo>	<picture> <resolution> low</ resolution> <filetype> png</ filetype> </picture>



4. Application Interoperability

- CORBA Client Calls:
  - GetInformation(Photo) → Resolution, Format
- CORBA Server Interface
  - GetResolution()→ Resolution
  - GetFormat()→ Format
- Client cannot interoperate with this service (even when the underlying protocols and data match)
  - Cannot easily standardise application interfaces



Approaches to Interoperability

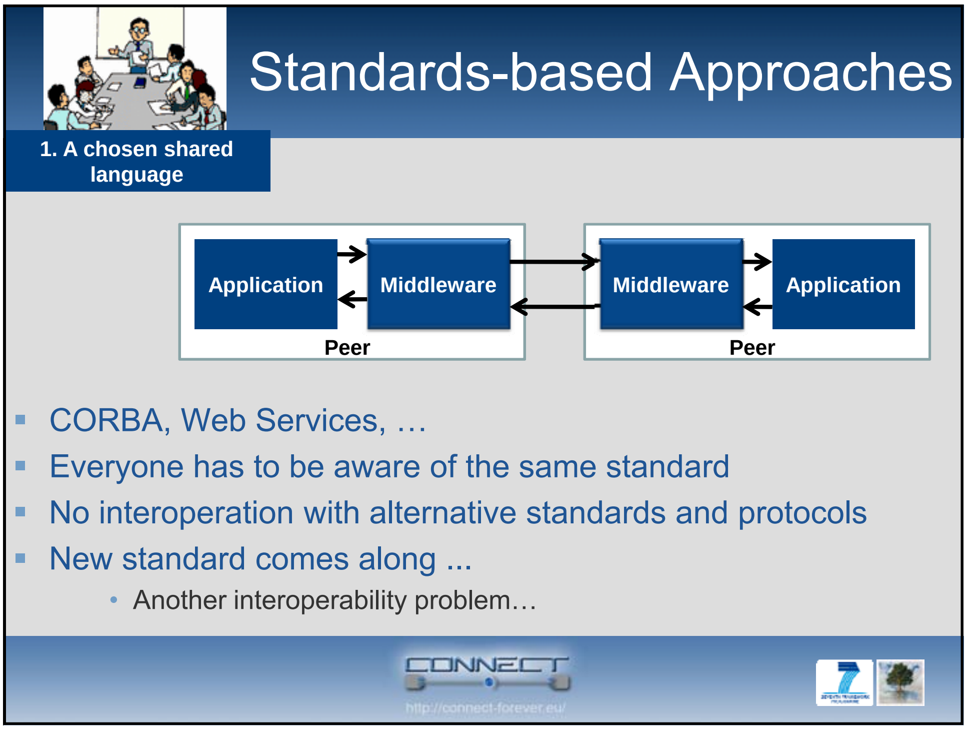
1. A chosen shared language

2. One 3rd party translator, e.g., English to French translator

3. Auxiliary Languages (e.g. Esperanto)

4. One speaker talks the other's language

5. Babel fish



Standards-based Approaches

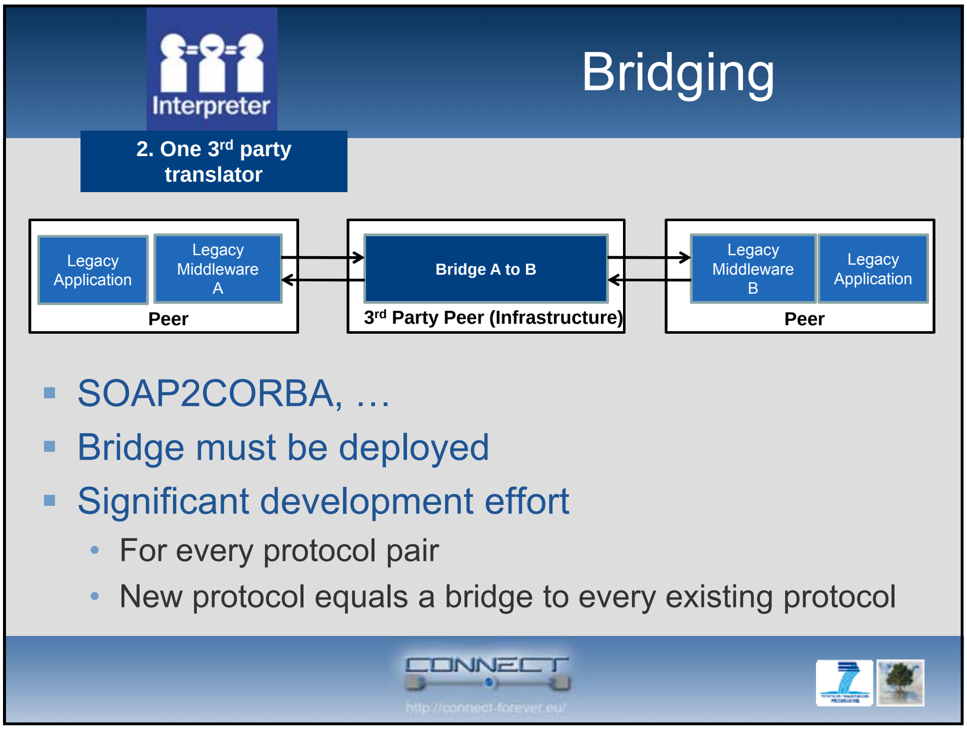
1. A chosen shared language

Application **Middleware** **Peer**

Middleware **Application** **Peer**

- CORBA, Web Services, …
- Everyone has to be aware of the same standard
- No interoperation with alternative standards and protocols
- New standard comes along ...
  - Another interoperability problem…

http://connect-forever.eu/



Bridging

Interpreter

2. One 3rd party translator

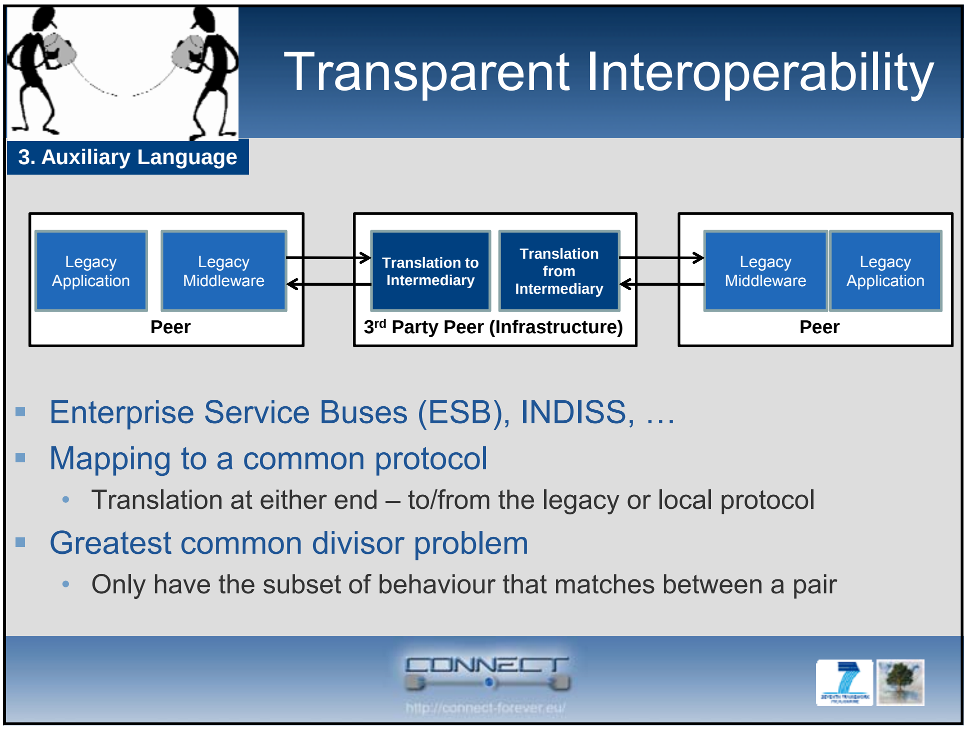
Legacy Application | Legacy Middleware A | **Peer**

Bridge A to B | **3rd Party Peer (Infrastructure)**

Legacy Middleware B | Legacy Application | **Peer**

- SOAP2CORBA, …
- Bridge must be deployed
- Significant development effort
  - For every protocol pair
  - New protocol equals a bridge to every existing protocol

http://connect-forever.eu/



Transparent Interoperability

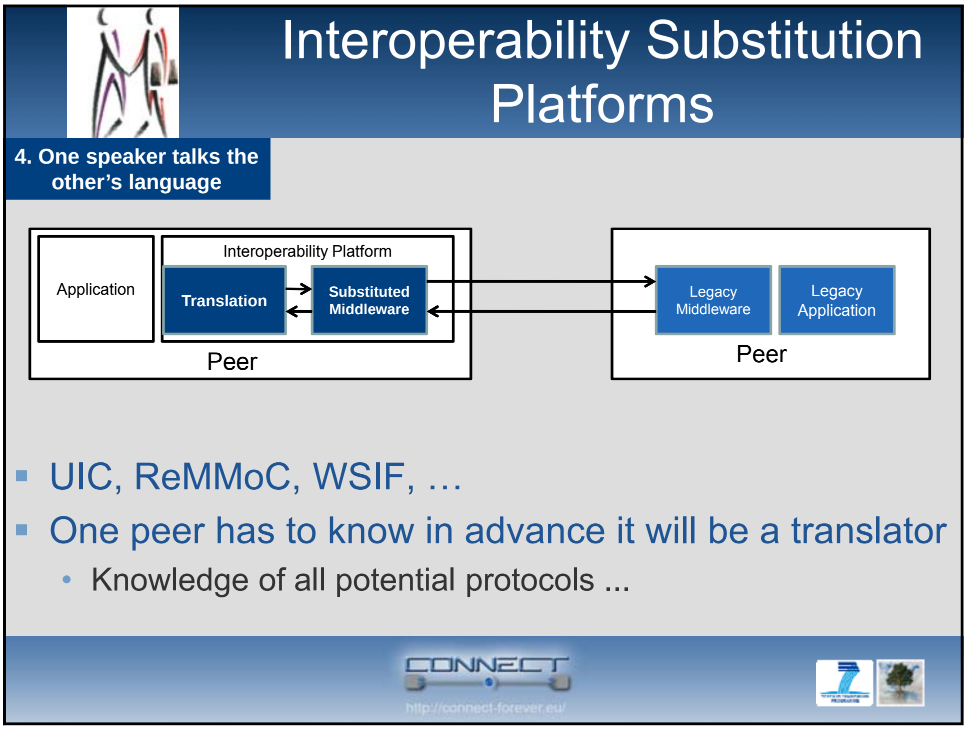
3. Auxiliary Language

Legacy Application | Legacy Middleware — Peer

Translation to Intermediary | Translation from Intermediary — 3rd Party Peer (Infrastructure)

Legacy Middleware | Legacy Application — Peer

- Enterprise Service Buses (ESB), INDISS, …
- Mapping to a common protocol
  - Translation at either end – to/from the legacy or local protocol
- Greatest common divisor problem
  - Only have the subset of behaviour that matches between a pair



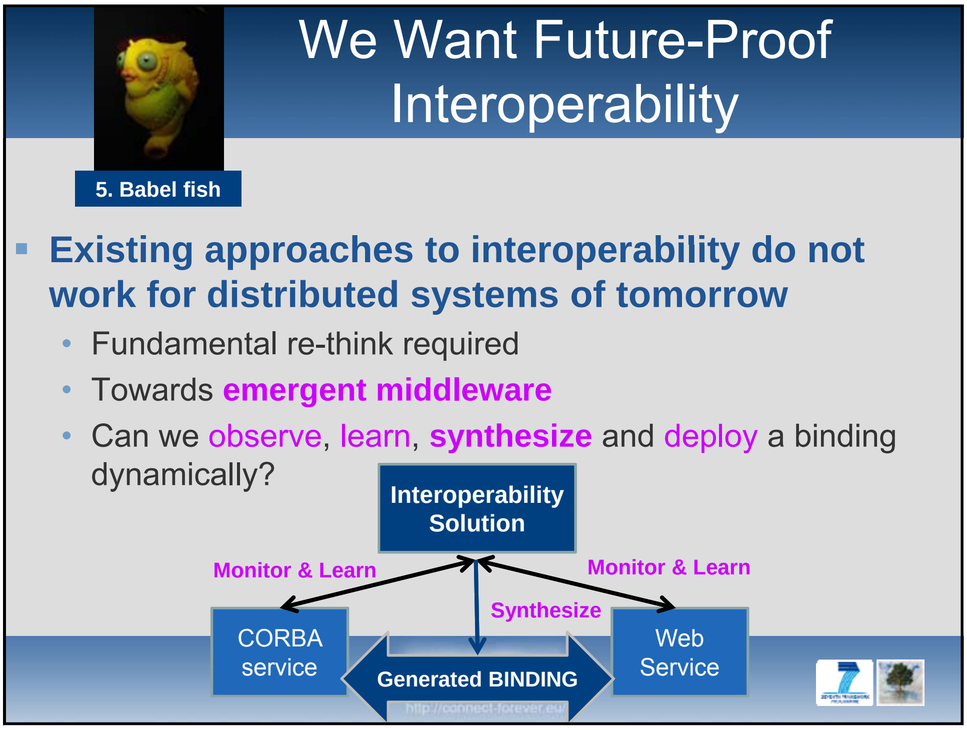
Interoperability Substitution Platforms

4. One speaker talks the other's language

Application | Interoperability Platform: Translation, Substituted Middleware — Peer

Legacy Middleware | Legacy Application — Peer

- UIC, ReMMoC, WSIF, …
- One peer has to know in advance it will be a translator
  - Knowledge of all potential protocols ...



We Want Future-Proof Interoperability

5. Babel fish

- **Existing approaches to interoperability do not work for distributed systems of tomorrow**
  - Fundamental re-think required
  - Towards **emergent middleware**
  - Can we observe, learn, **synthesize** and deploy a binding dynamically?

Interoperability Solution

Monitor & Learn

Monitor & Learn

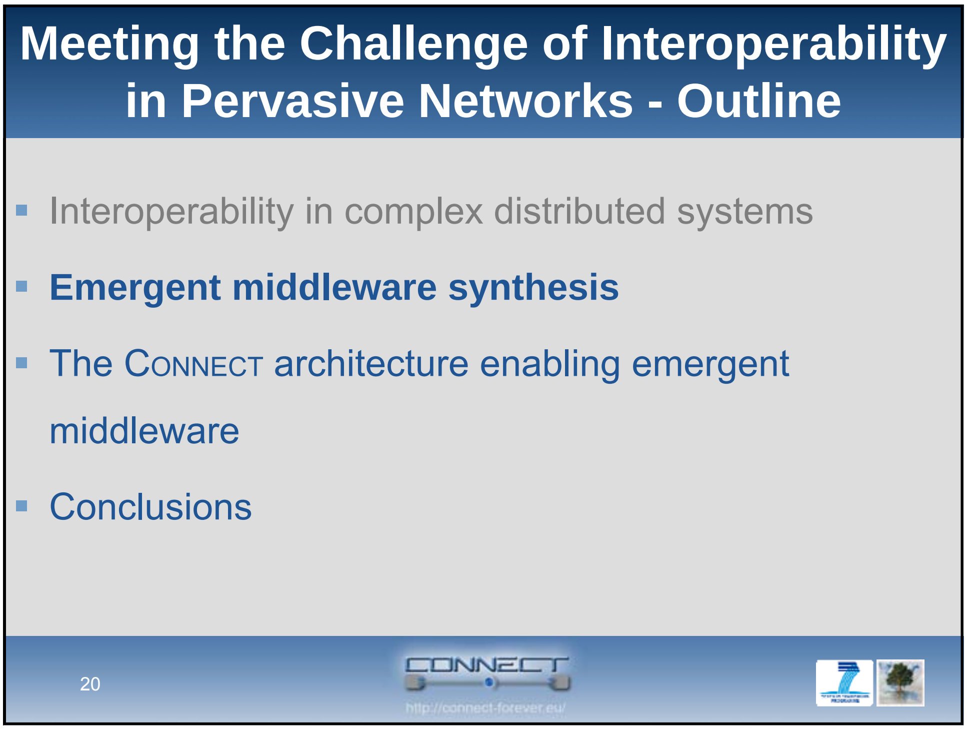
Synthesize

CORBA service

Generated BINDING

Web Service

http://connect-forever.eu/



Meeting the Challenge of Interoperability in Pervasive Networks - Outline

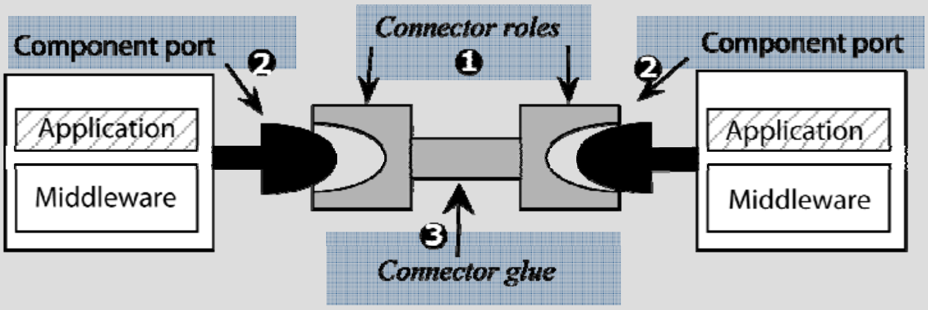
- Interoperability in complex distributed systems
- **Emergent middleware synthesis**
- The CONNECT architecture enabling emergent middleware
- Conclusions

20

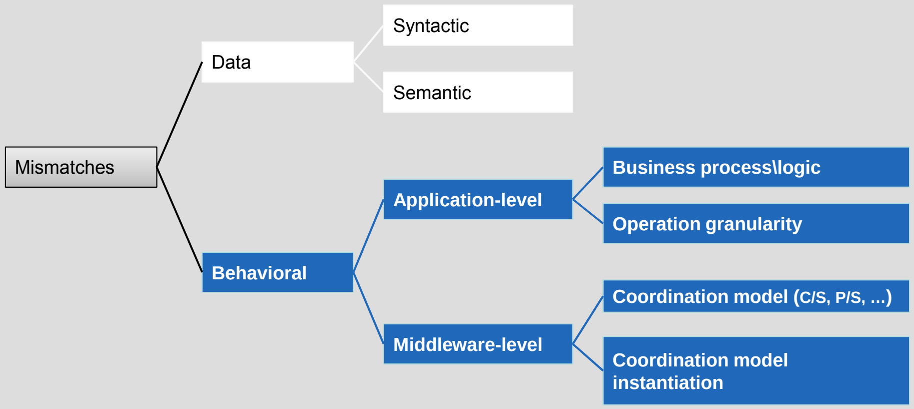
CONNECT

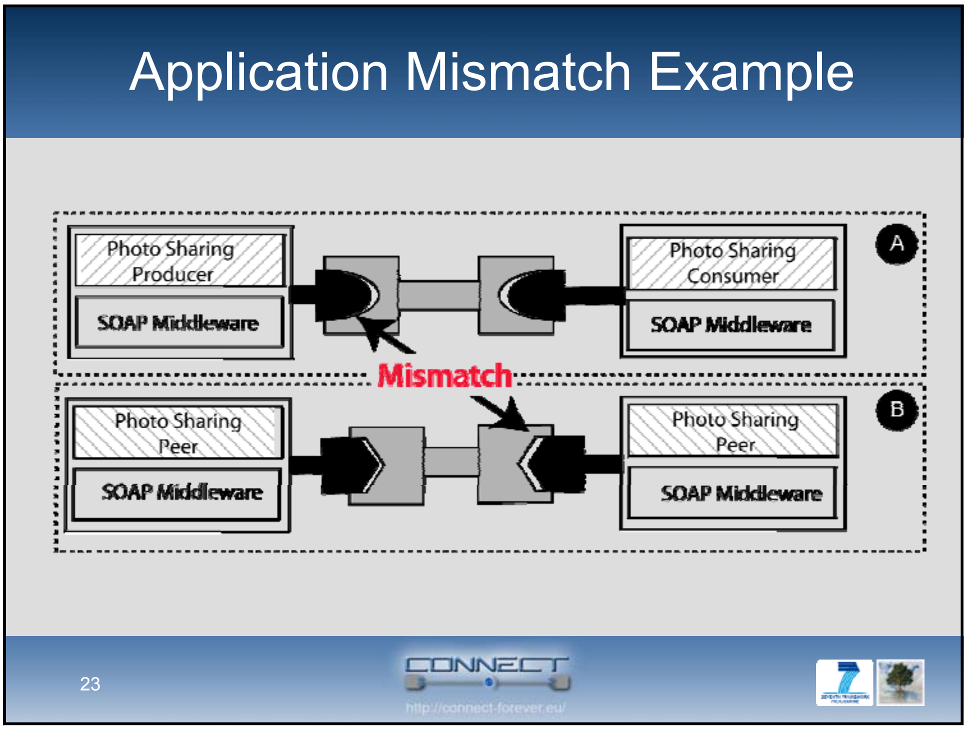
http://connect-forever.eu/

Connecting Systems



Classifying Connection Mismatches





Application Mismatch Example

Photo Sharing Producer

SOAP Middleware

Photo Sharing Consumer

SOAP Middleware

A

Mismatch

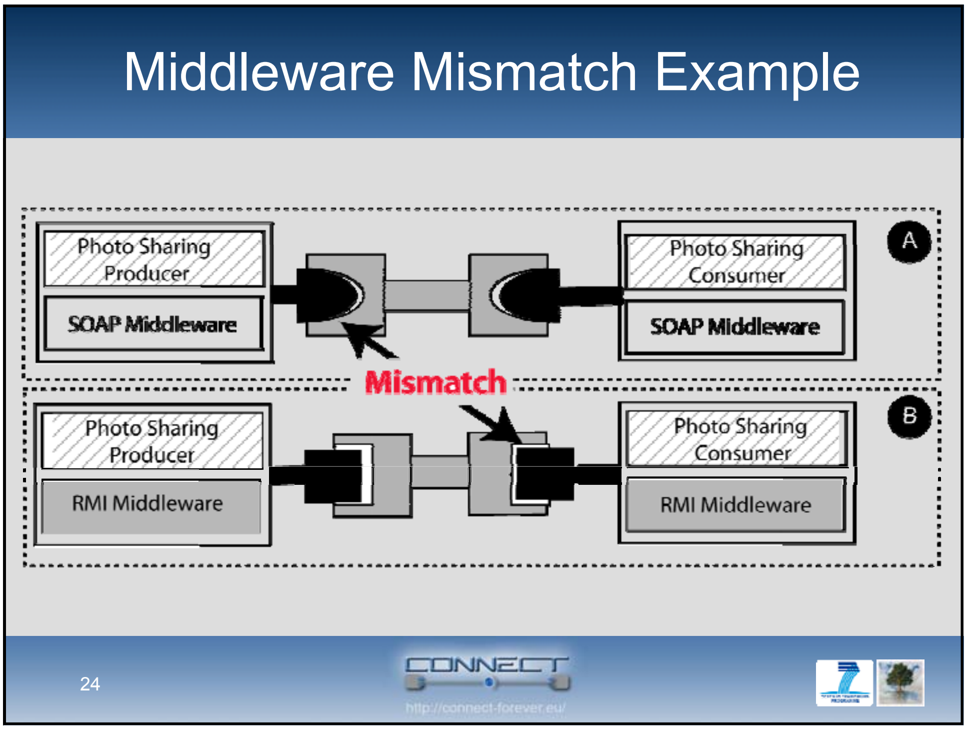
Photo Sharing Peer

SOAP Middleware

Photo Sharing Peer

SOAP Middleware

B



Middleware Mismatch Example

Photo Sharing Producer

SOAP Middleware

Photo Sharing Consumer

SOAP Middleware

A

Mismatch

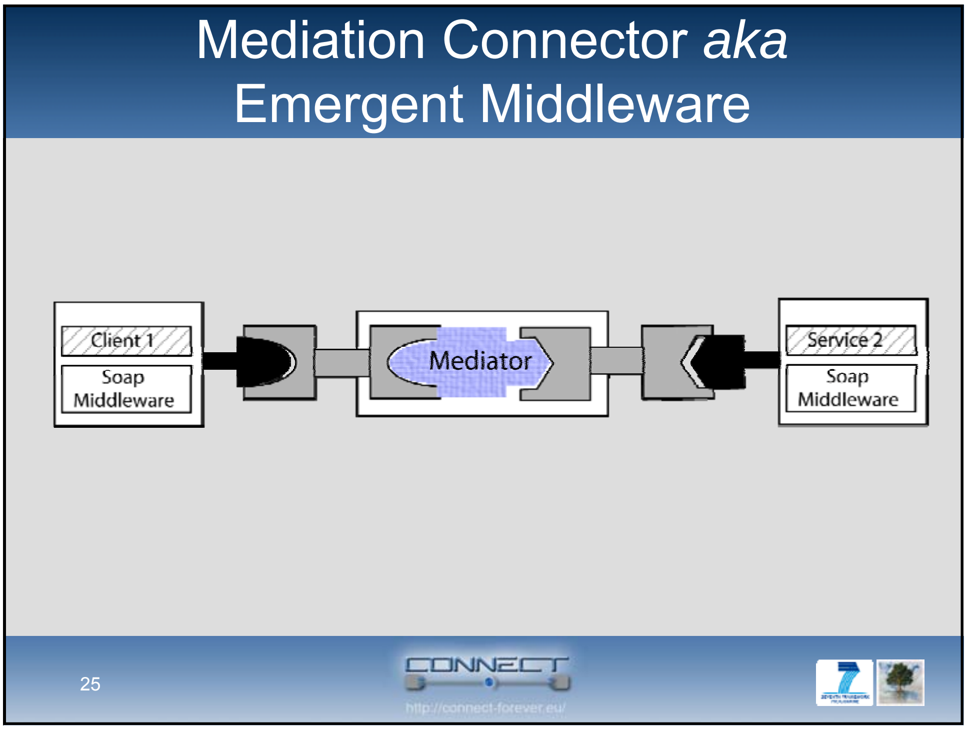
Photo Sharing Producer

RMI Middleware

Photo Sharing Consumer

RMI Middleware

B



Mediation Connector *aka* Emergent Middleware

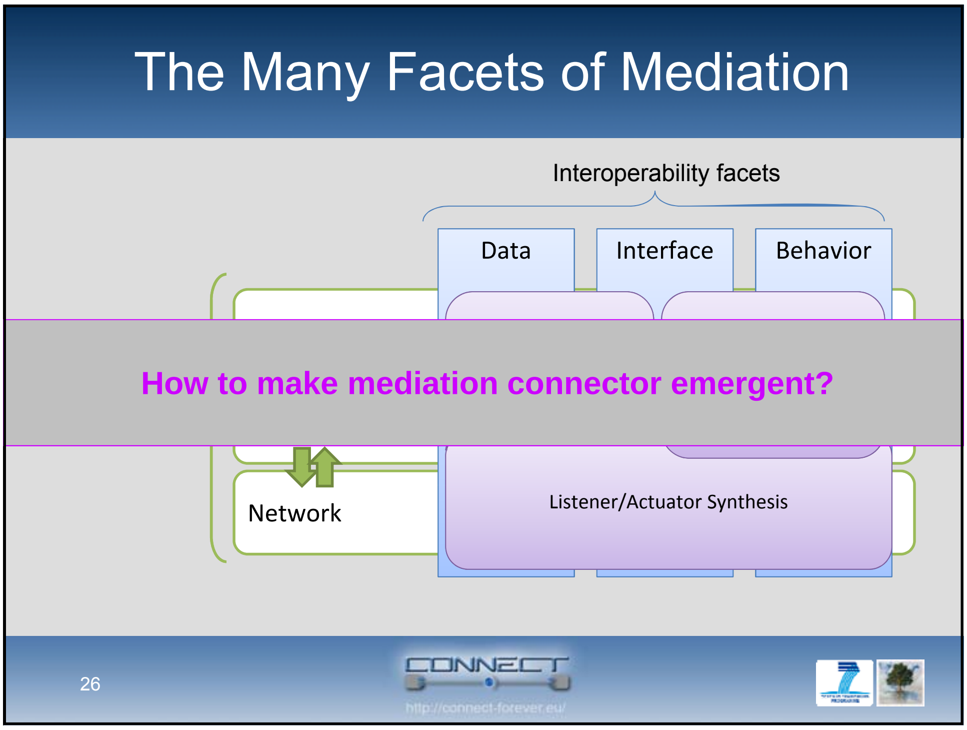
Client 1

Soap Middleware

Mediator

Service 2

Soap Middleware



The Many Facets of Mediation

Interoperability facets

Data

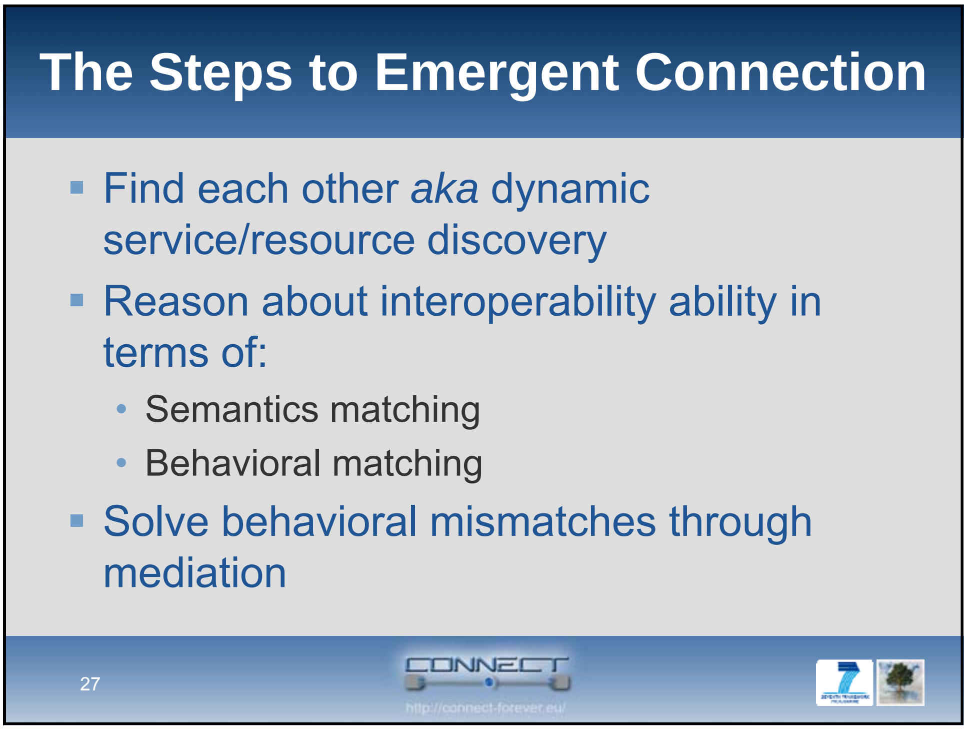
Interface

Behavior

How to make mediation connector emergent?

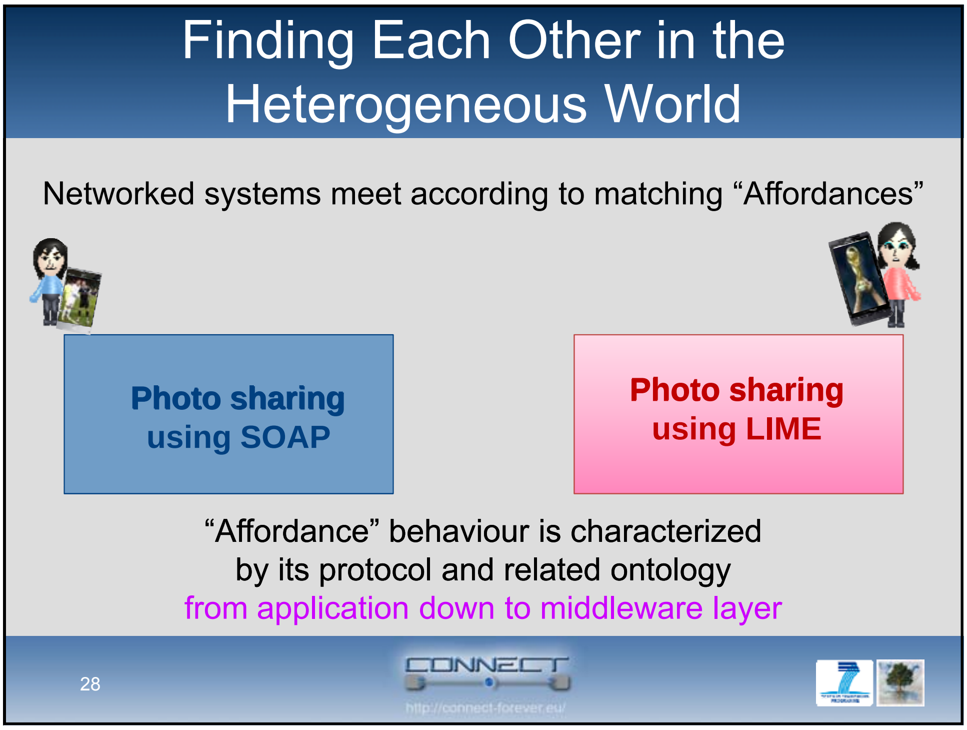
Network

Listener/Actuator Synthesis



The Steps to Emergent Connection

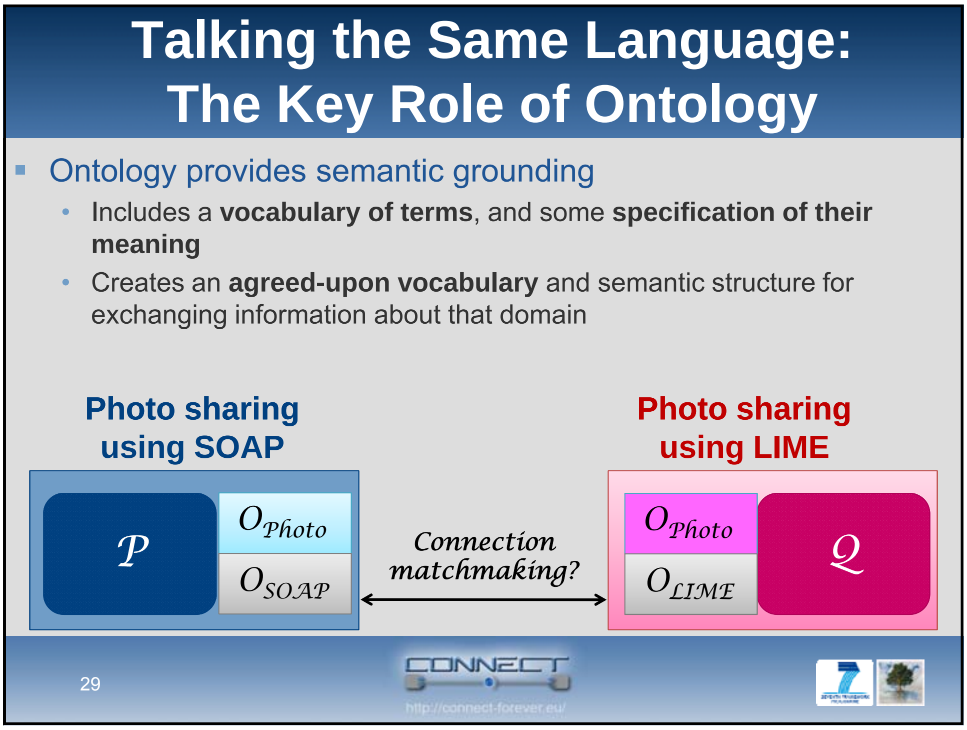
- Find each other *aka* dynamic service/resource discovery
- Reason about interoperability ability in terms of:
  - Semantics matching
  - Behavioral matching
- Solve behavioral mismatches through mediation



Finding Each Other in the Heterogeneous World

Networked systems meet according to matching “Affordances”

“Affordance” behaviour is characterized by its protocol and related ontology from application down to middleware layer



Talking the Same Language: The Key Role of Ontology

- Ontology provides semantic grounding
  - Includes a **vocabulary of terms**, and some **specification of their meaning**
  - Creates an **agreed-upon vocabulary** and semantic structure for exchanging information about that domain

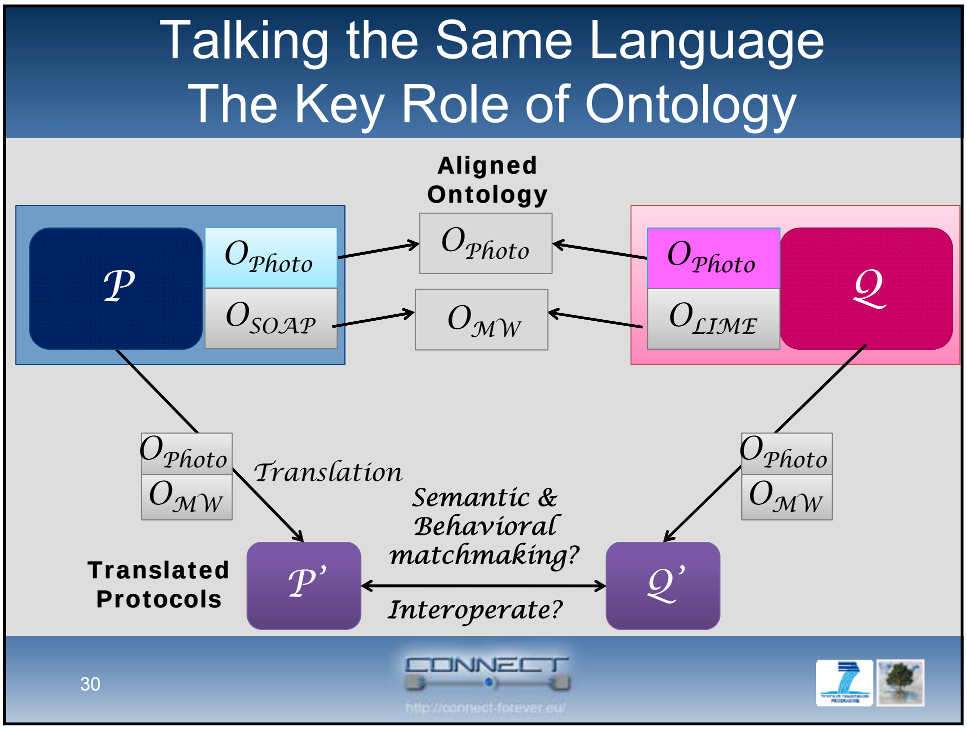
Photo sharing using SOAP

Photo sharing using LIME

$\mathcal{P}$ O_{Photo} O_{SOAP}

Connection matchmaking?

O_{Photo} O_{LIME} $\mathcal{Q}$



Talking the Same Language The Key Role of Ontology

Aligned Ontology

O_{Photo}

O_{MW}

$\mathcal{P}$ O_{Photo} O_{SOAP}

O_{Photo} O_{LIME} $\mathcal{Q}$

O_{Photo} O_{MW}

Translation

O_{Photo} O_{MW}

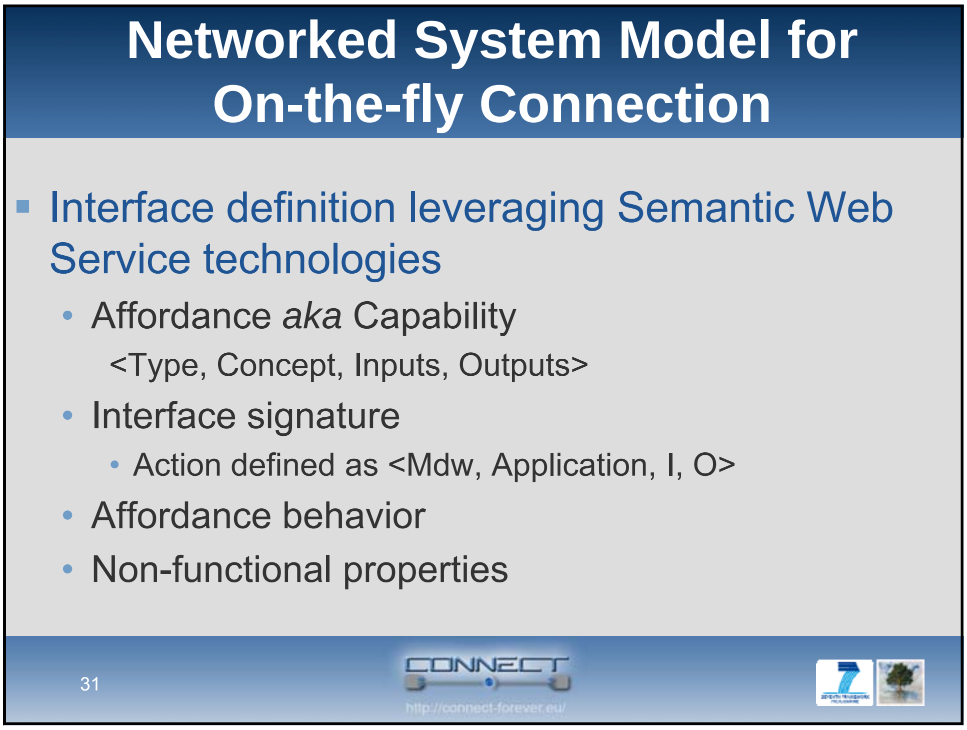
Semantic & Behavioral matchmaking?

Translated Protocols

$\mathcal{P}'$

$\mathcal{Q}'$

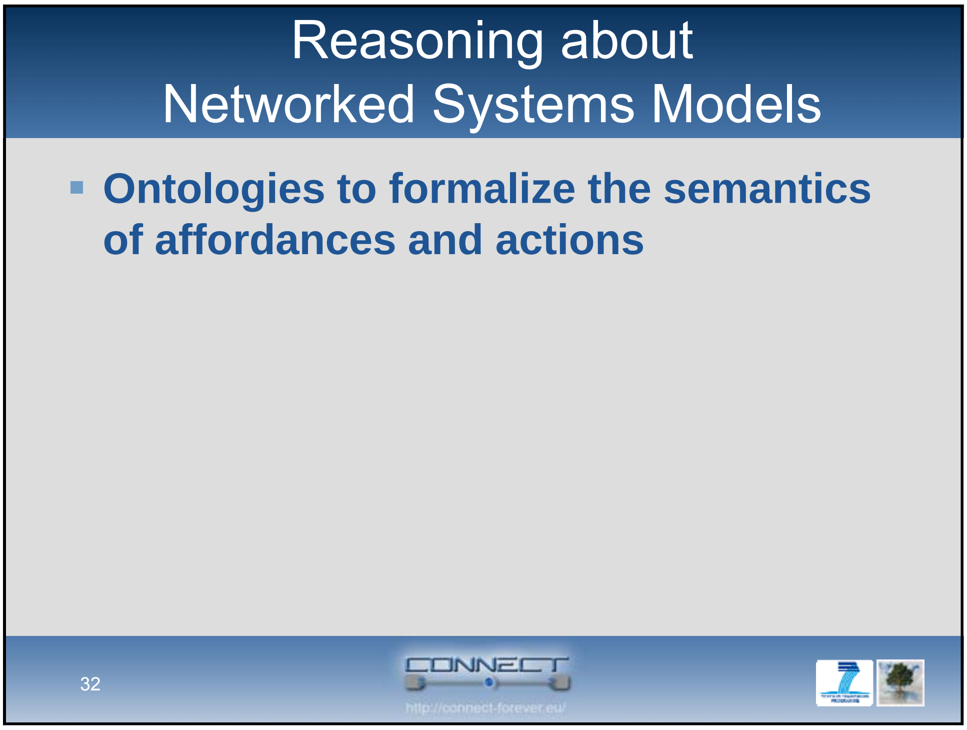
Interoperate?



Networked System Model for On-the-fly Connection

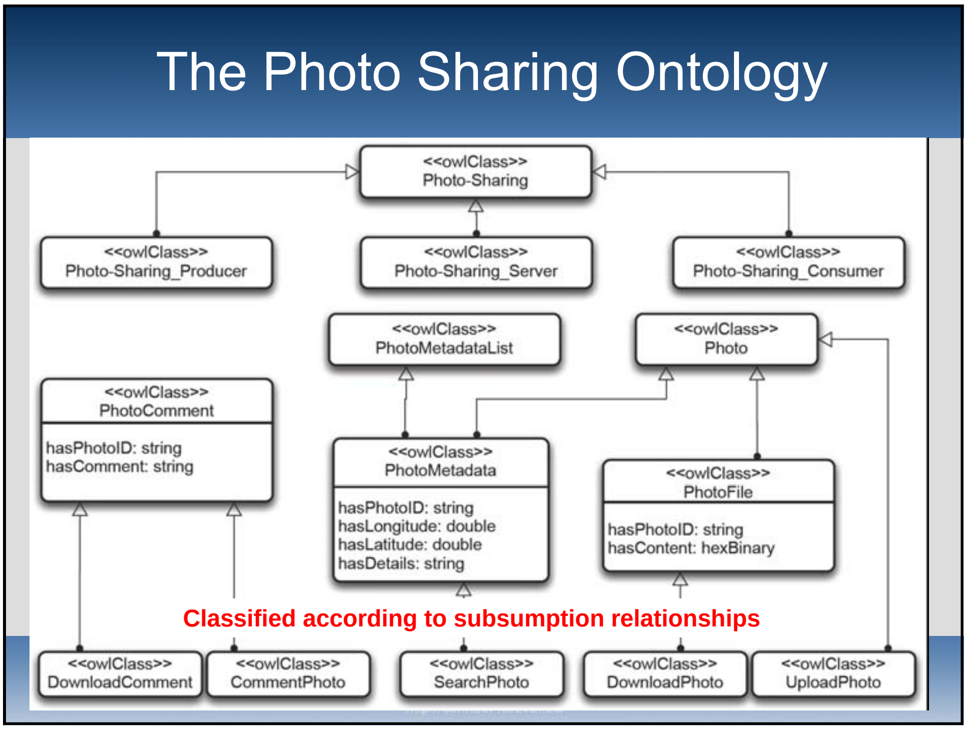
- Interface definition leveraging Semantic Web Service technologies
  - Affordance *aka* Capability

    <Type, Concept, Inputs, Outputs>
  - Interface signature
    - Action defined as <Mdw, Application, I, O>
  - Affordance behavior
  - Non-functional properties



Reasoning about Networked Systems Models

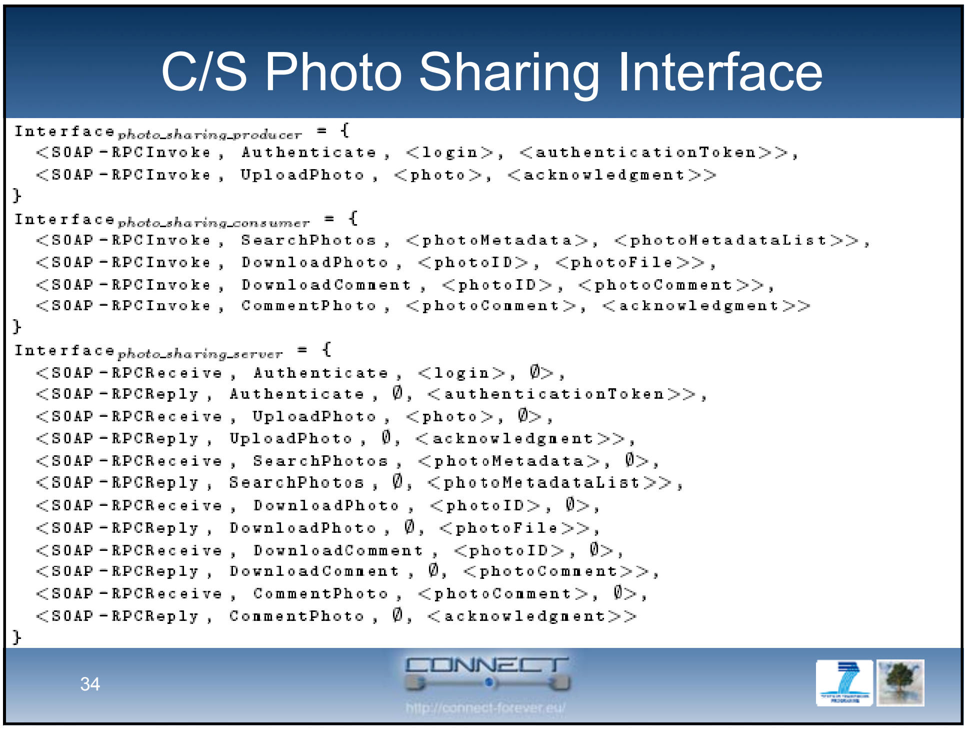
- **Ontologies to formalize the semantics of affordances and actions**



The Photo Sharing Ontology

<<owlClass>> Photo-Sharing
<<owlClass>> Photo-Sharing_Producer
<<owlClass>> Photo-Sharing_Server
<<owlClass>> Photo-Sharing_Consumer
<<owlClass>> PhotoMetadataList
<<owlClass>> Photo
<<owlClass>> PhotoComment
hasPhotoID: string
hasComment: string
<<owlClass>> PhotoMetadata
hasPhotoID: string
hasLongitude: double
hasLatitude: double
hasDetails: string
<<owlClass>> PhotoFile
hasPhotoID: string
hasContent: hexBinary
<<owlClass>> DownloadComment
<<owlClass>> CommentPhoto
<<owlClass>> SearchPhoto
<<owlClass>> DownloadPhoto
<<owlClass>> UploadPhoto

Classified according to subsumption relationships



C/S Photo Sharing Interface

```
Interface_photo_sharing_producer = {
  <SOAP-RPCInvoke, Authenticate, <login>, <authenticationToken>>,
  <SOAP-RPCInvoke, UploadPhoto, <photo>, <acknowledgment>>
}
Interface_photo_sharing_consumer = {
  <SOAP-RPCInvoke, SearchPhotos, <photoMetadata>, <photoMetadataList>>,
  <SOAP-RPCInvoke, DownloadPhoto, <photoID>, <photoFile>>,
  <SOAP-RPCInvoke, DownloadComment, <photoID>, <photoComment>>,
  <SOAP-RPCInvoke, CommentPhoto, <photoComment>, <acknowledgment>>
}
Interface_photo_sharing_server = {
  <SOAP-RPCReceive, Authenticate, <login>, ∅>,
  <SOAP-RPCReply, Authenticate, ∅, <authenticationToken>>,
  <SOAP-RPCReceive, UploadPhoto, <photo>, ∅>,
  <SOAP-RPCReply, UploadPhoto, ∅, <acknowledgment>>,
  <SOAP-RPCReceive, SearchPhotos, <photoMetadata>, ∅>,
  <SOAP-RPCReply, SearchPhotos, ∅, <photoMetadataList>>,
  <SOAP-RPCReceive, DownloadPhoto, <photoID>, ∅>,
  <SOAP-RPCReply, DownloadPhoto, ∅, <photoFile>>,
  <SOAP-RPCReceive, DownloadComment, <photoID>, ∅>,
  <SOAP-RPCReply, DownloadComment, ∅, <photoComment>>,
  <SOAP-RPCReceive, CommentPhoto, <photoComment>, ∅>,
  <SOAP-RPCReply, CommentPhoto, ∅, <acknowledgment>>
}
```

P2P Photo Sharing Interface

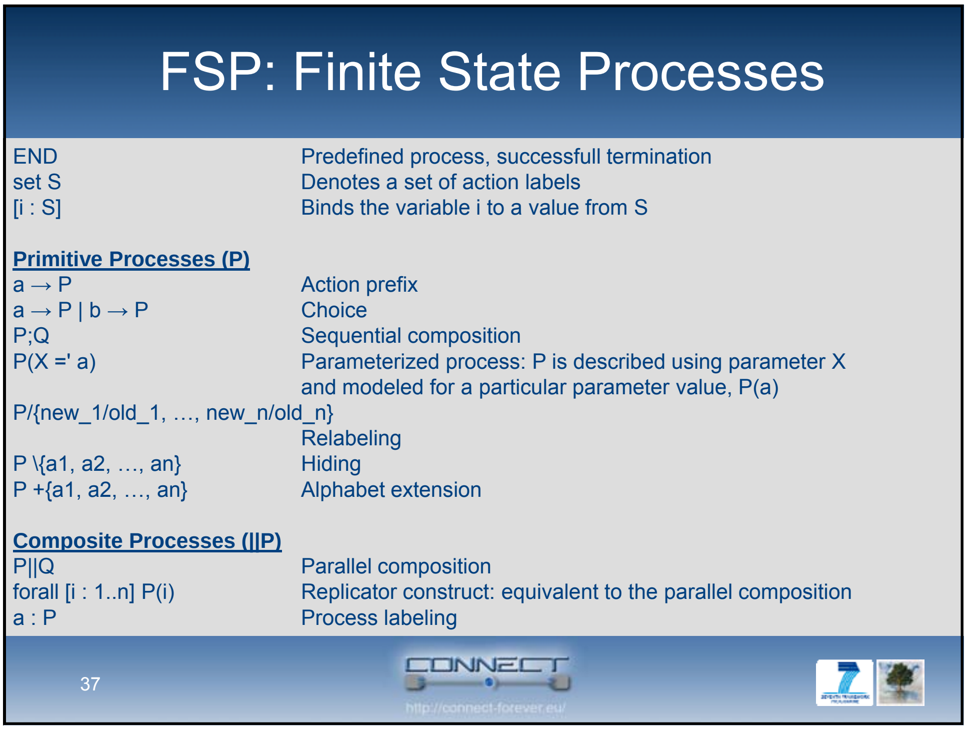
$\text{Interface}_{\text{photo\_sharing}}$ = {

< Out, PhotoMetadata, ∅, < photoMetadata >>,
< Out, PhotoFile, ∅, < photoFile >>,
< Rdg, PhotoMetadata,< photoMetadata >, < photoMetadataList >>,
< Rd, PhotoFile, < photoID >, < photoFile >>,
< Rd, PhotoComment, < photoID >, < photoComment >>,
< Out, PhotoComment, ∅, < photoComment >>,
< In, PhotoComment, < photoID >, < photoComment >>,
< Rd, PhotoComment, < photoID >, < photoComment >>

}

Reasoning about Networked Systems Models

- Ontologies to formalize the semantics of affordances and actions
- **Finite state processes to formalize the behavior of affordances**
  - LTS semantics
  - LTSA tool for automated model checking



FSP: Finite State Processes

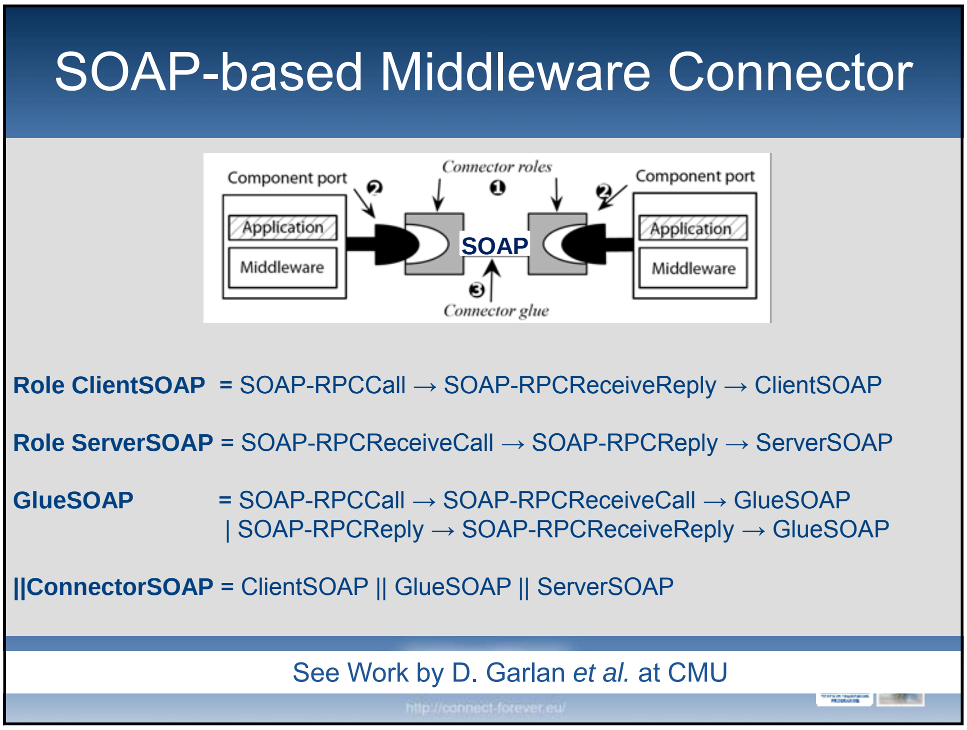
END	Predefined process, successfull termination
set S	Denotes a set of action labels
[i : S]	Binds the variable i to a value from S

Primitive Processes (P)

a → P	Action prefix
a → P | b → P	Choice
P;Q	Sequential composition
P(X =' a)	Parameterized process: P is described using parameter X and modeled for a particular parameter value, P(a)
P/{new_1/old_1, …, new_n/old_n}	Relabeling
P \{a1, a2, …, an}	Hiding
P +{a1, a2, …, an}	Alphabet extension

Composite Processes (||P)

P||Q	Parallel composition
forall [i : 1..n] P(i)	Replicator construct: equivalent to the parallel composition
a : P	Process labeling



SOAP-based Middleware Connector

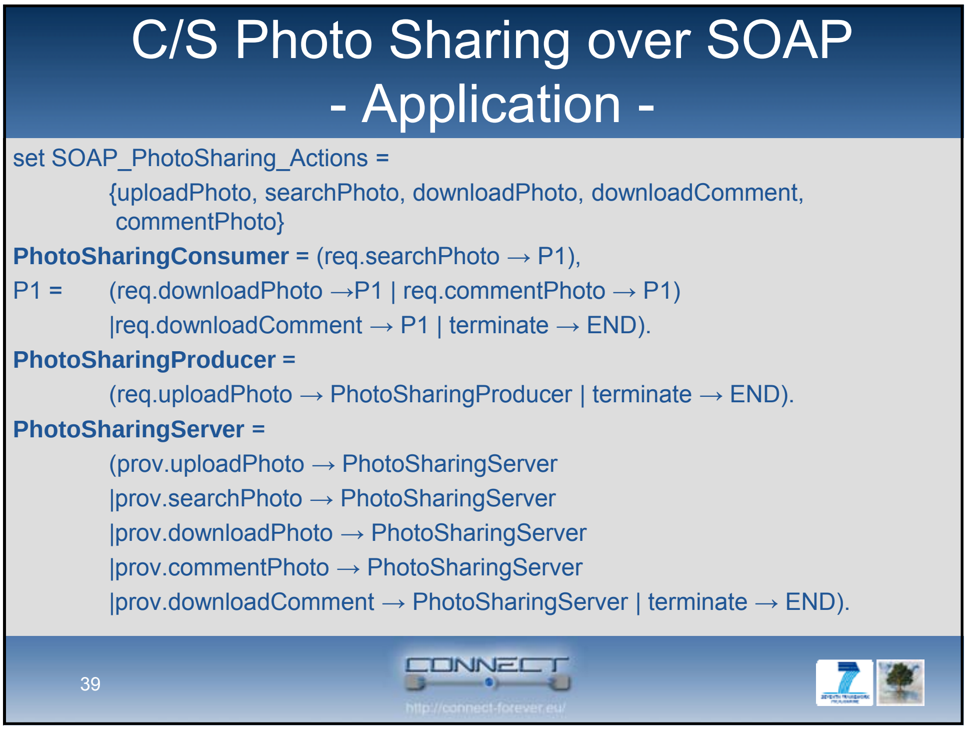
Role ClientSOAP = SOAP-RPCCall → SOAP-RPCReceiveReply → ClientSOAP

Role ServerSOAP = SOAP-RPCReceiveCall → SOAP-RPCReply → ServerSOAP

GlueSOAP = SOAP-RPCCall → SOAP-RPCReceiveCall → GlueSOAP
| SOAP-RPCReply → SOAP-RPCReceiveReply → GlueSOAP

||ConnectorSOAP = ClientSOAP || GlueSOAP || ServerSOAP

See Work by D. Garlan *et al.* at CMU



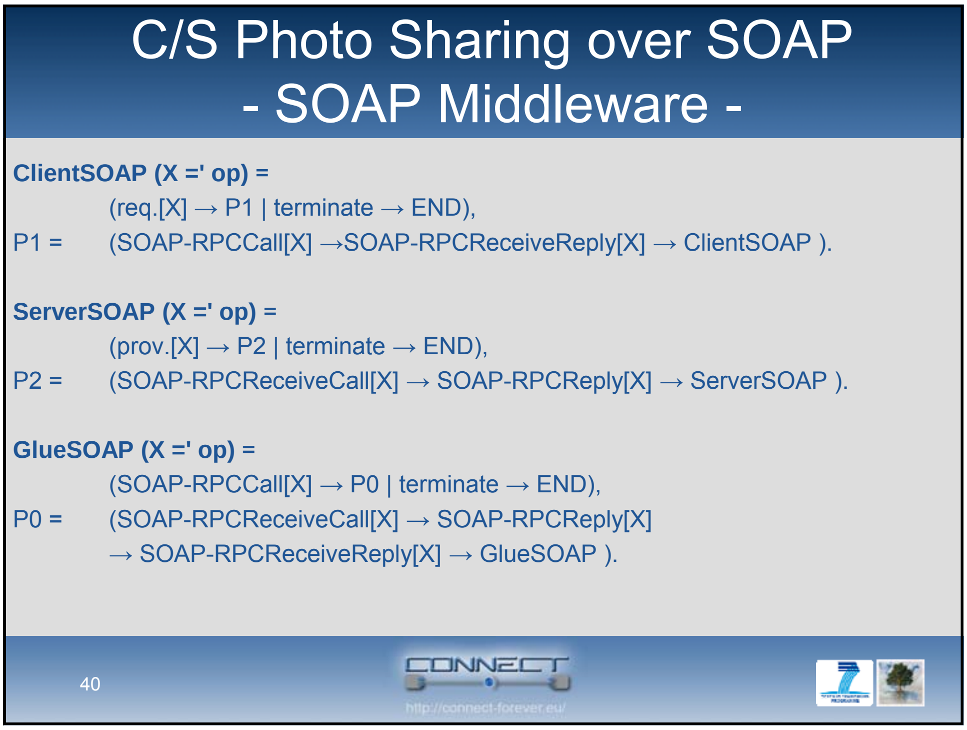
C/S Photo Sharing over SOAP - Application -

set SOAP_PhotoSharing_Actions =
{uploadPhoto, searchPhoto, downloadPhoto, downloadComment, commentPhoto}

PhotoSharingConsumer = (req.searchPhoto → P1),
P1 = (req.downloadPhoto →P1 | req.commentPhoto → P1)
|req.downloadComment → P1 | terminate → END).

PhotoSharingProducer =
(req.uploadPhoto → PhotoSharingProducer | terminate → END).

PhotoSharingServer =
(prov.uploadPhoto → PhotoSharingServer
|prov.searchPhoto → PhotoSharingServer
|prov.downloadPhoto → PhotoSharingServer
|prov.commentPhoto → PhotoSharingServer
|prov.downloadComment → PhotoSharingServer | terminate → END).

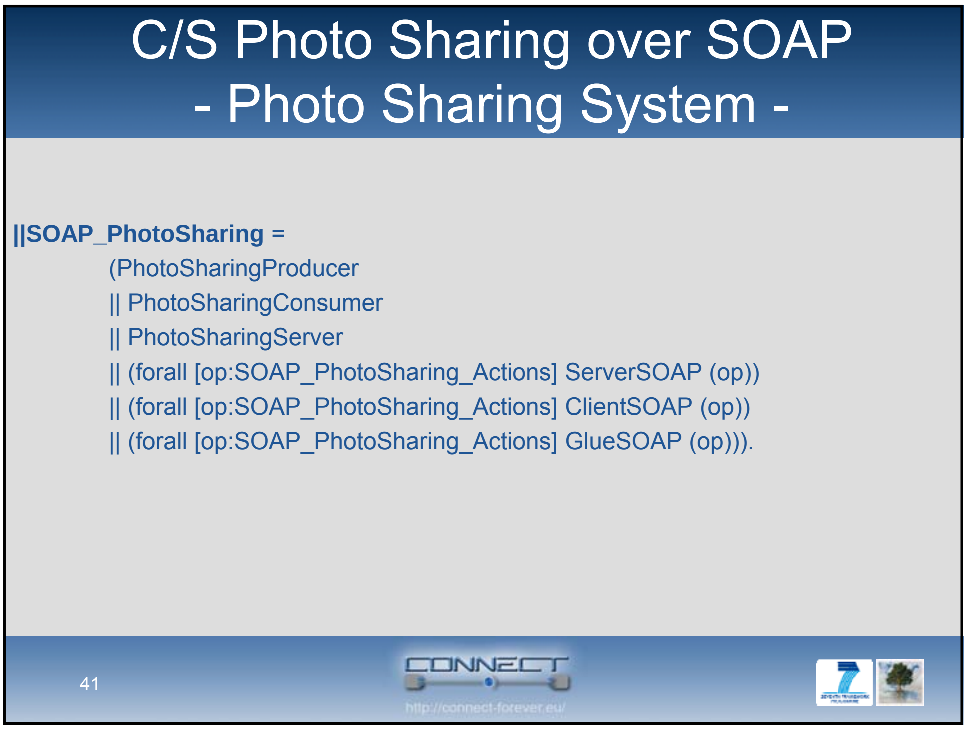


C/S Photo Sharing over SOAP - SOAP Middleware -

ClientSOAP (X =' op) =
(req.[X] → P1 | terminate → END),
P1 = (SOAP-RPCCall[X] →SOAP-RPCReceiveReply[X] → ClientSOAP).

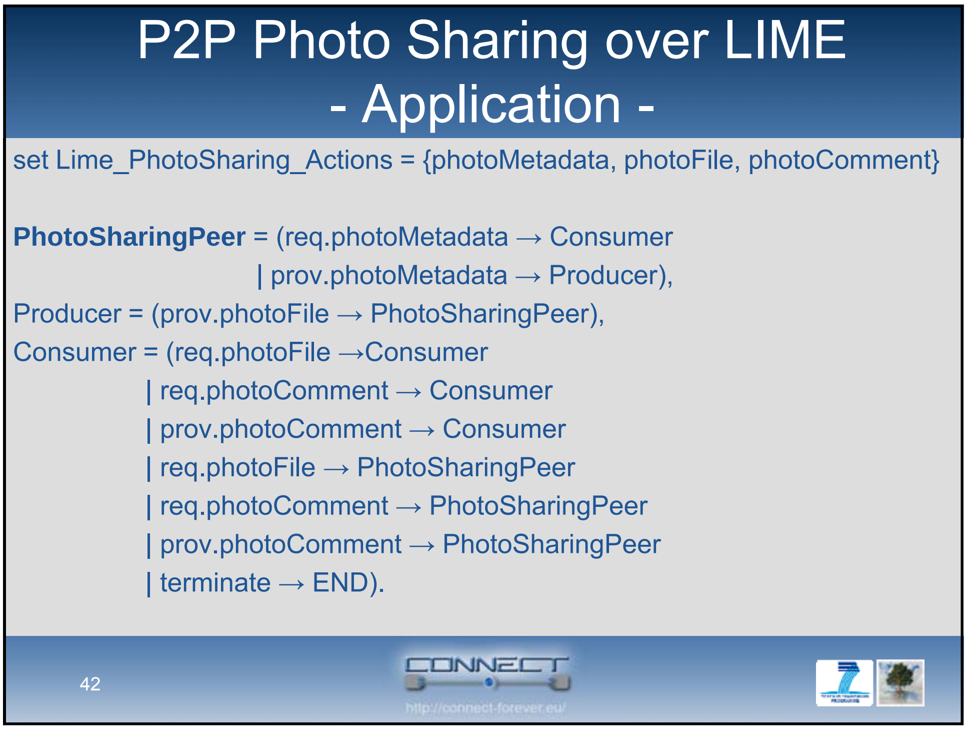
ServerSOAP (X =' op) =
(prov.[X] → P2 | terminate → END),
P2 = (SOAP-RPCReceiveCall[X] → SOAP-RPCReply[X] → ServerSOAP).

GlueSOAP (X =' op) =
(SOAP-RPCCall[X] → P0 | terminate → END),
P0 = (SOAP-RPCReceiveCall[X] → SOAP-RPCReply[X]
→ SOAP-RPCReceiveReply[X] → GlueSOAP).



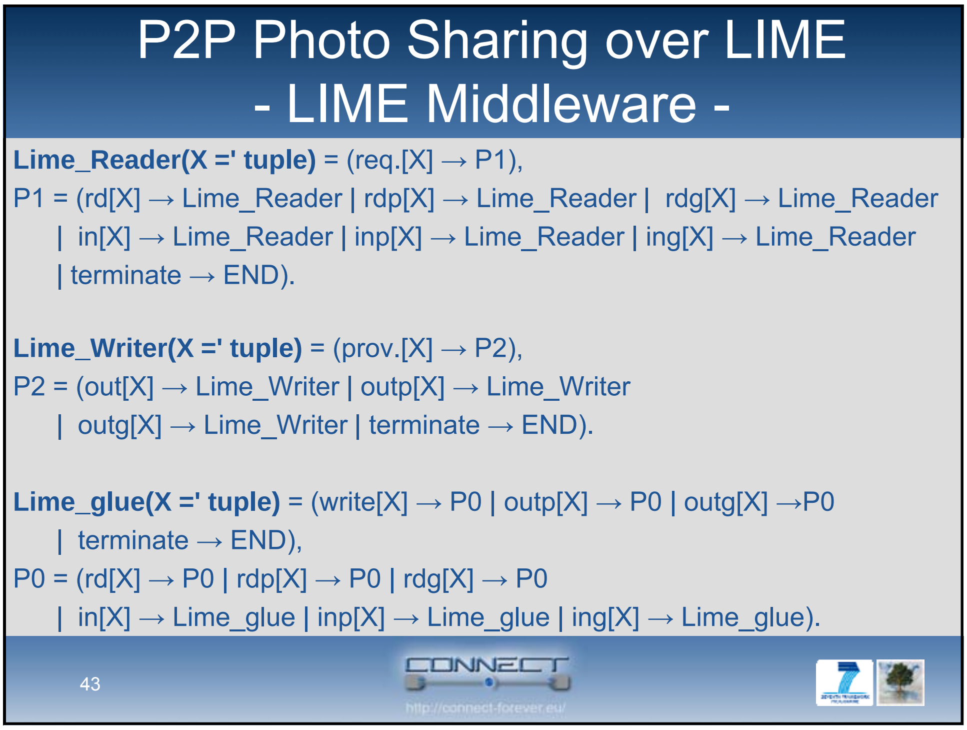
C/S Photo Sharing over SOAP - Photo Sharing System -

```
||SOAP_PhotoSharing =
        (PhotoSharingProducer
        || PhotoSharingConsumer
        || PhotoSharingServer
        || (forall [op:SOAP_PhotoSharing_Actions] ServerSOAP (op))
        || (forall [op:SOAP_PhotoSharing_Actions] ClientSOAP (op))
        || (forall [op:SOAP_PhotoSharing_Actions] GlueSOAP (op))).
```



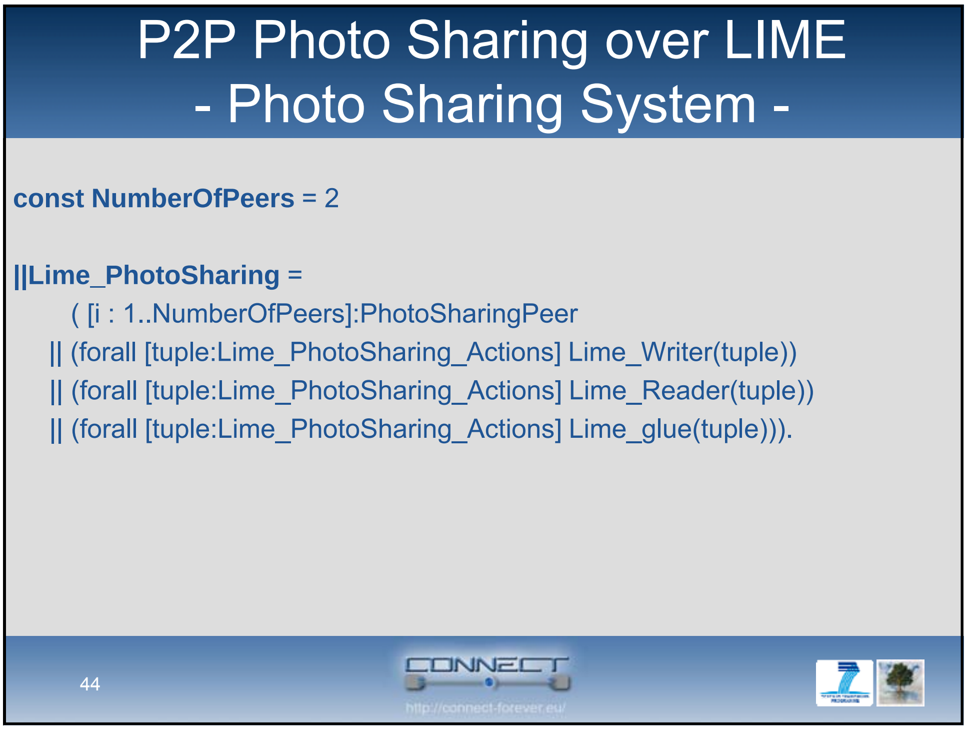
P2P Photo Sharing over LIME - Application -

```
set Lime_PhotoSharing_Actions = {photoMetadata, photoFile, photoComment}

PhotoSharingPeer = (req.photoMetadata → Consumer
                   | prov.photoMetadata → Producer),
Producer = (prov.photoFile → PhotoSharingPeer),
Consumer = (req.photoFile →Consumer
           | req.photoComment → Consumer
           | prov.photoComment → Consumer
           | req.photoFile → PhotoSharingPeer
           | req.photoComment → PhotoSharingPeer
           | prov.photoComment → PhotoSharingPeer
           | terminate → END).
```



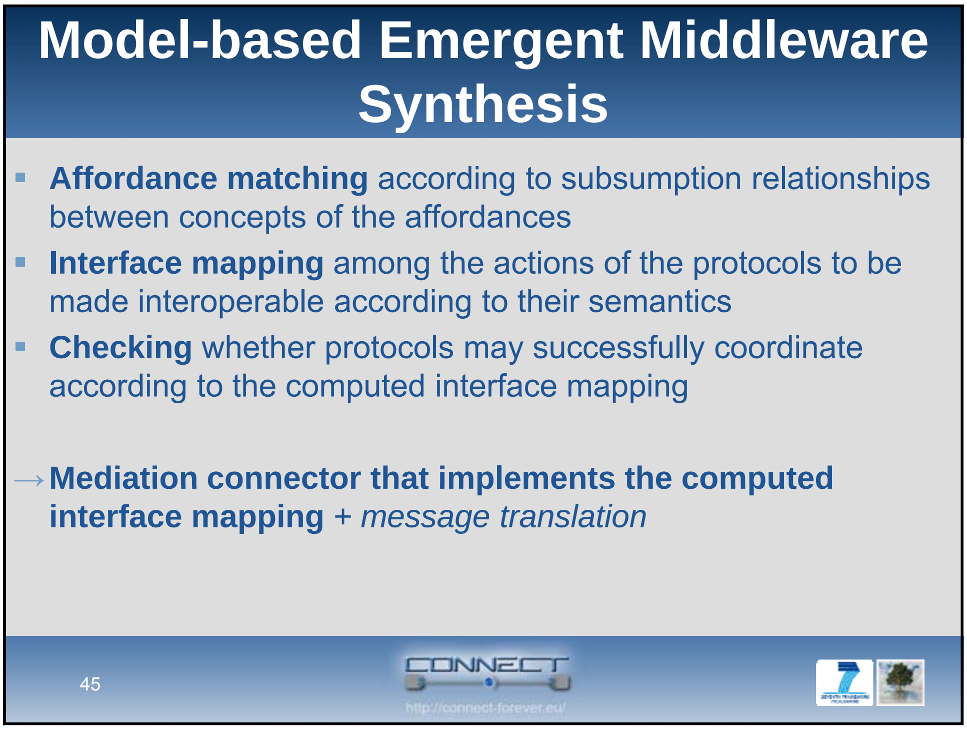
P2P Photo Sharing over LIME - LIME Middleware -

```
Lime_Reader(X =' tuple) = (req.[X] → P1),
P1 = (rd[X] → Lime_Reader | rdp[X] → Lime_Reader |  rdg[X] → Lime_Reader
    |  in[X] → Lime_Reader | inp[X] → Lime_Reader | ing[X] → Lime_Reader
    | terminate → END).

Lime_Writer(X =' tuple) = (prov.[X] → P2),
P2 = (out[X] → Lime_Writer | outp[X] → Lime_Writer
    |  outg[X] → Lime_Writer | terminate → END).

Lime_glue(X =' tuple) = (write[X] → P0 | outp[X] → P0 | outg[X] →P0
    |  terminate → END),
P0 = (rd[X] → P0 | rdp[X] → P0 | rdg[X] → P0
    |  in[X] → Lime_glue | inp[X] → Lime_glue | ing[X] → Lime_glue).
```



P2P Photo Sharing over LIME - Photo Sharing System -

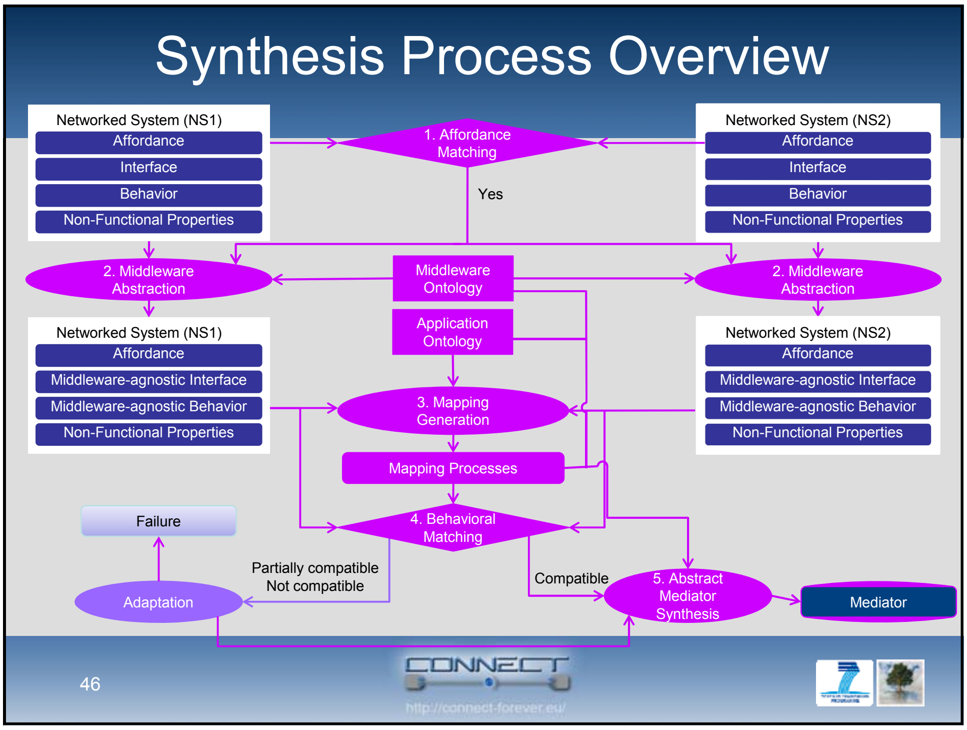
```
const NumberOfPeers = 2

||Lime_PhotoSharing =
      ( [i : 1..NumberOfPeers]:PhotoSharingPeer
    || (forall [tuple:Lime_PhotoSharing_Actions] Lime_Writer(tuple))
    || (forall [tuple:Lime_PhotoSharing_Actions] Lime_Reader(tuple))
    || (forall [tuple:Lime_PhotoSharing_Actions] Lime_glue(tuple))).
```



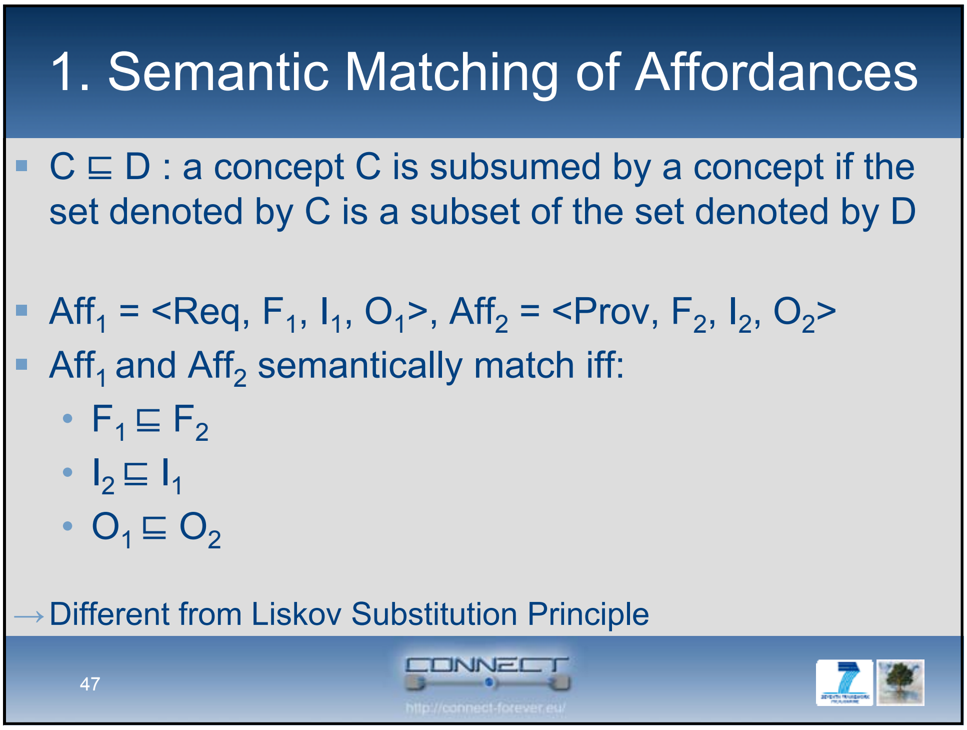
Model-based Emergent Middleware Synthesis

- **Affordance matching** according to subsumption relationships between concepts of the affordances
- **Interface mapping** among the actions of the protocols to be made interoperable according to their semantics
- **Checking** whether protocols may successfully coordinate according to the computed interface mapping

→ **Mediation connector that implements the computed interface mapping** + *message translation*



Synthesis Process Overview

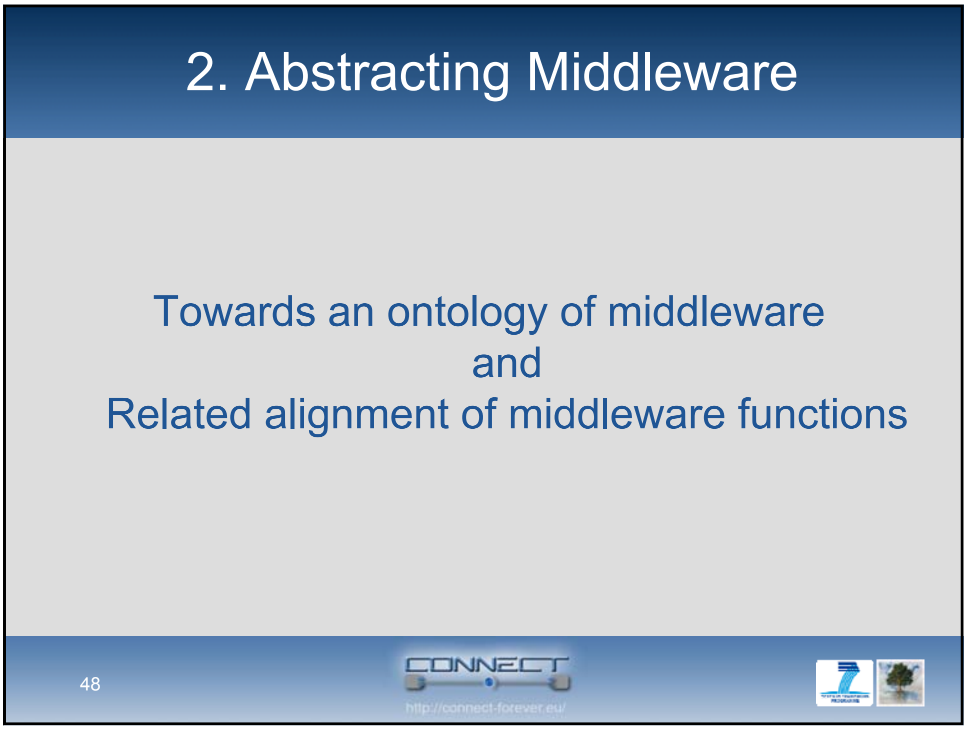


1. Semantic Matching of Affordances

- $C \sqsubseteq D$: a concept C is subsumed by a concept if the set denoted by C is a subset of the set denoted by D

- Aff_1 = <Req, F_1, I_1, O_1>, Aff_2 = <Prov, F_2, I_2, O_2>
- Aff_1 and Aff_2 semantically match iff:
  - $F_1 \sqsubseteq F_2$
  - $I_2 \sqsubseteq I_1$
  - $O_1 \sqsubseteq O_2$

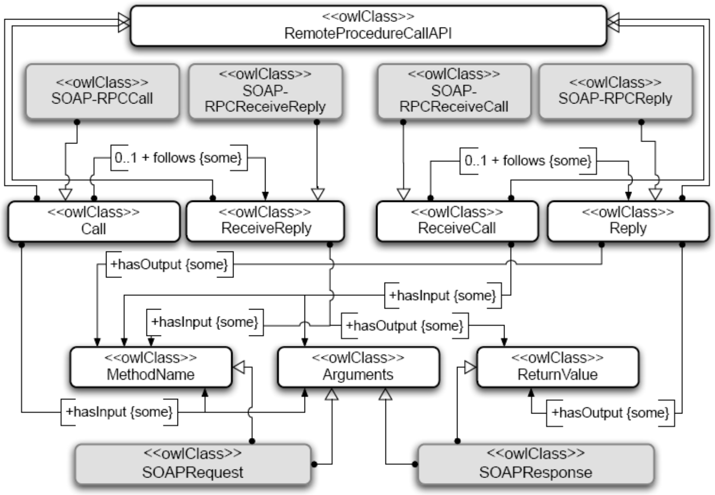
→ Different from Liskov Substitution Principle



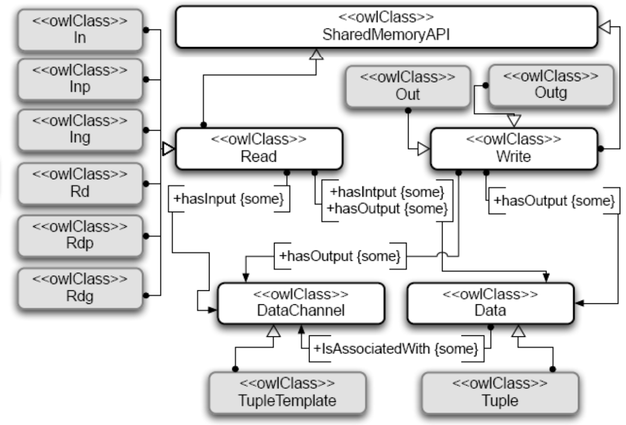
2. Abstracting Middleware

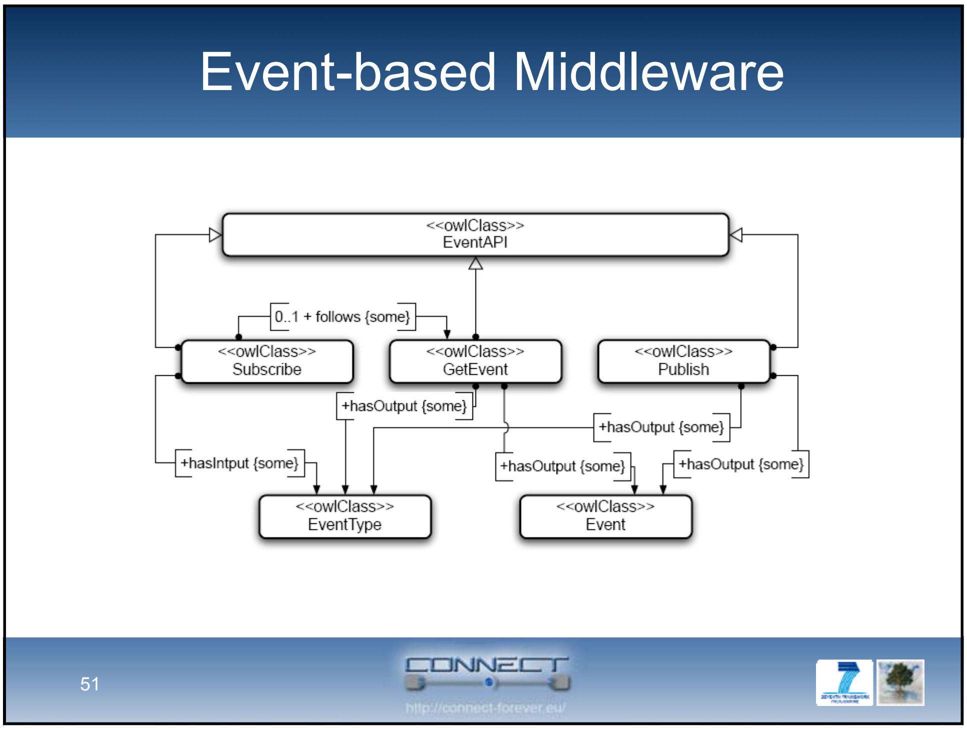
Towards an ontology of middleware
and
Related alignment of middleware functions

RPC Middleware

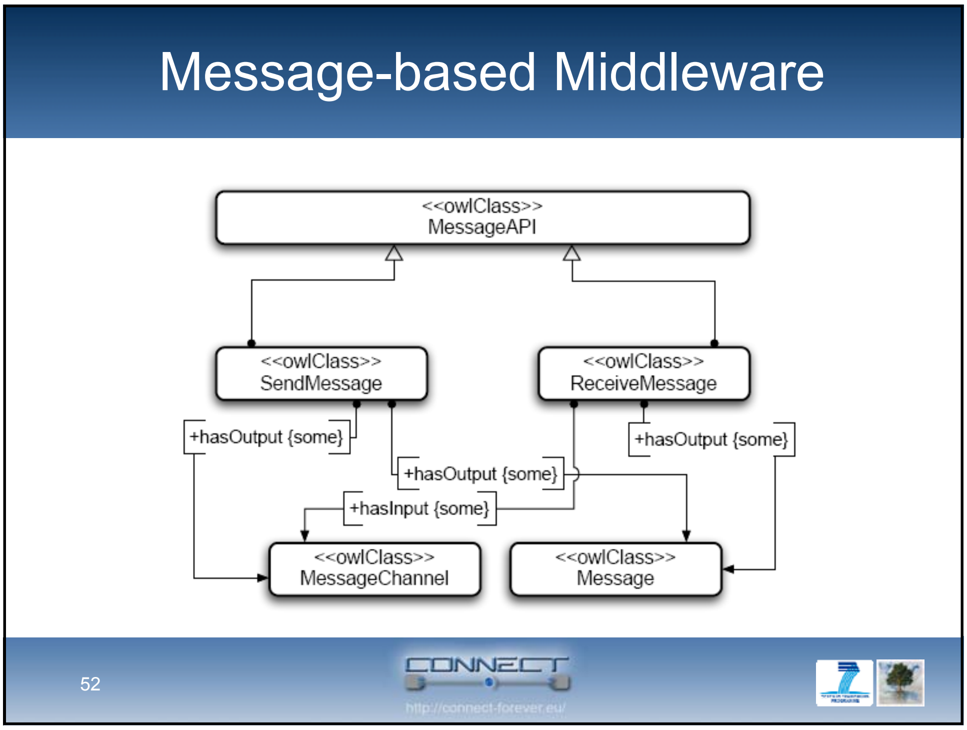


Shared Memory Middleware

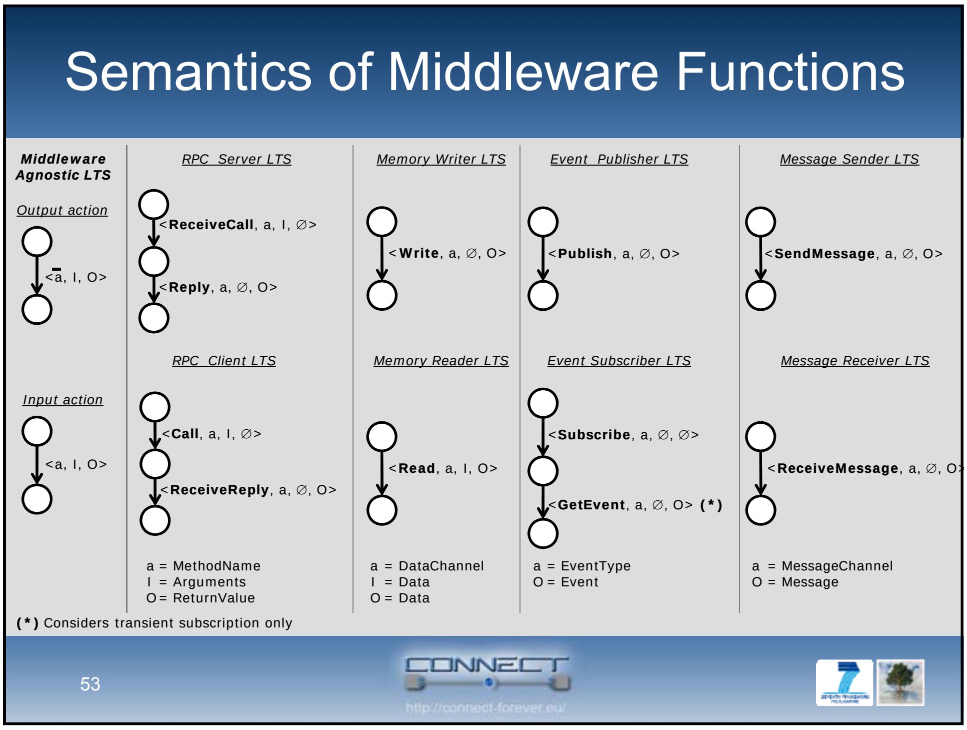




Event-based Middleware



Message-based Middleware



Semantics of Middleware Functions

Middleware Agnostic LTS

Output action

<$\overline{a}$, I, O>

Input action

<a, I, O>

RPC Server LTS

<**ReceiveCall**, a, I, ∅>

<**Reply**, a, ∅, O>

RPC Client LTS

<**Call**, a, I, ∅>

<**ReceiveReply**, a, ∅, O>

a = MethodName
I = Arguments
O = ReturnValue

Memory Writer LTS

<**Write**, a, ∅, O>

Memory Reader LTS

<**Read**, a, I, O>

a = DataChannel
I = Data
O = Data

Event Publisher LTS

<**Publish**, a, ∅, O>

Event Subscriber LTS

<**Subscribe**, a, ∅, ∅>

<**GetEvent**, a, ∅, O> **(*)**

a = EventType
O = Event

Message Sender LTS

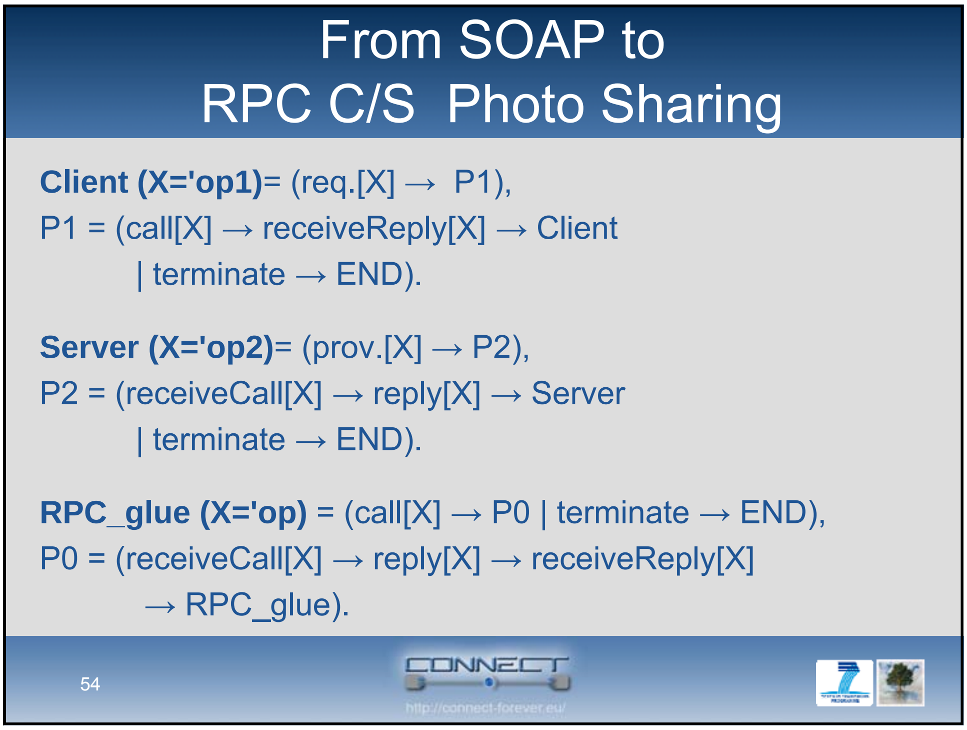
<**SendMessage**, a, ∅, O>

Message Receiver LTS

<**ReceiveMessage**, a, ∅, O

a = MessageChannel
O = Message

(*) Considers transient subscription only



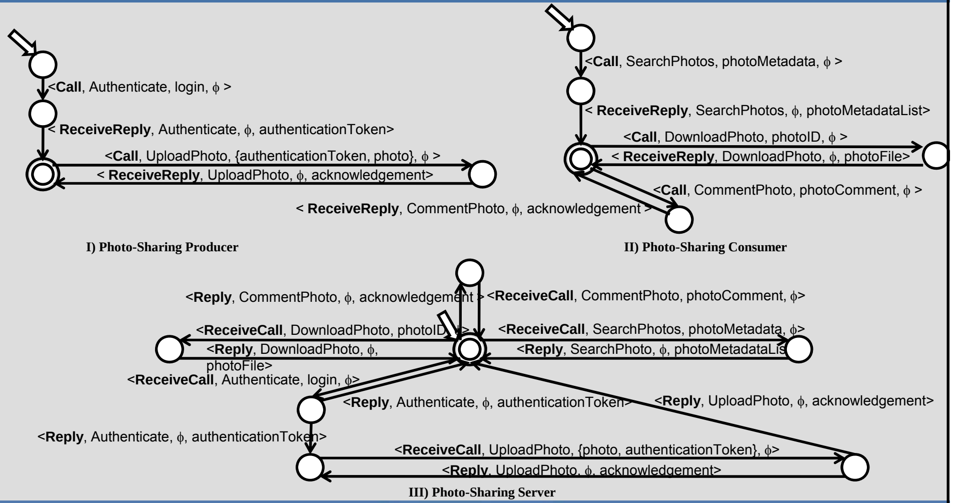
From SOAP to RPC C/S Photo Sharing

Client (X='op1)= (req.[X] → P1),
P1 = (call[X] → receiveReply[X] → Client
| terminate → END).

Server (X='op2)= (prov.[X] → P2),
P2 = (receiveCall[X] → reply[X] → Server
| terminate → END).

RPC_glue (X='op) = (call[X] → P0 | terminate → END),
P0 = (receiveCall[X] → reply[X] → receiveReply[X]
→ RPC_glue).

RPC-based Photo Sharing LTS Semantics


<**Call**, Authenticate, login, ϕ >
< **ReceiveReply**, Authenticate, ϕ, authenticationToken>
<**Call**, UploadPhoto, {authenticationToken, photo}, ϕ >
< **ReceiveReply**, UploadPhoto, ϕ, acknowledgement>
<**Call**, SearchPhotos, photoMetadata, ϕ >
< **ReceiveReply**, SearchPhotos, ϕ, photoMetadataList>
<**Call**, DownloadPhoto, photoID, ϕ >
< **ReceiveReply**, DownloadPhoto, ϕ, photoFile>
<**Call**, CommentPhoto, photoComment, ϕ >
< **ReceiveReply**, CommentPhoto, ϕ, acknowledgement >
<**Reply**, CommentPhoto, ϕ, acknowledgement >
<**ReceiveCall**, CommentPhoto, photoComment, ϕ>
<**ReceiveCall**, DownloadPhoto, photoID, ϕ>
<**Reply**, DownloadPhoto, ϕ, photoFile>
<**ReceiveCall**, SearchPhotos, photoMetadata, ϕ>
<**Reply**, SearchPhoto, ϕ, photoMetadataList>
<**ReceiveCall**, Authenticate, login, ϕ>
<**Reply**, Authenticate, ϕ, authenticationToken>
<**Reply**, UploadPhoto, ϕ, acknowledgement>
<**Reply**, Authenticate, ϕ, authenticationToken>
<**ReceiveCall**, UploadPhoto, {photo, authenticationToken}, ϕ>
<**Reply**, UploadPhoto, ϕ, acknowledgement>

I) Photo-Sharing Producer

II) Photo-Sharing Consumer

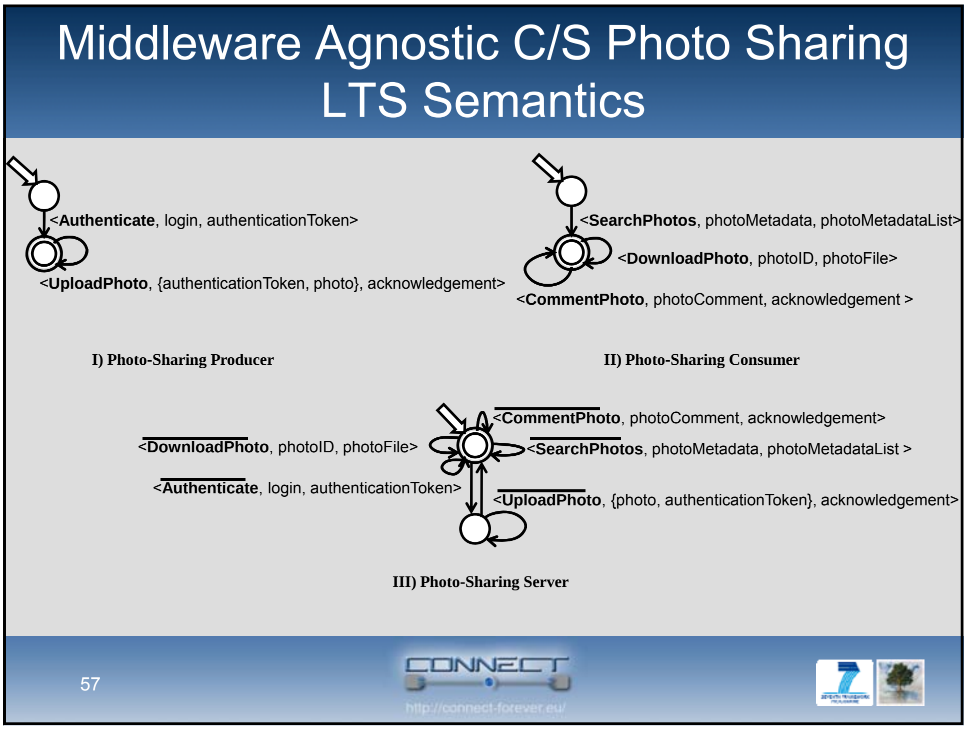
III) Photo-Sharing Server

To Middleware Agnostic C/S Photo Sharing

Client (X='op1) = (req.[X] → P1),
P1 = (input[X] → Client | terminate → END).

Server (X='op2) = (prov.[X] → P2),
P2 = (output[X] → Server | terminate → END).

RPC_glue (X='op) =
(output[X] → P0 | terminate → END),
P0 = (input[X] → RPC_glue).



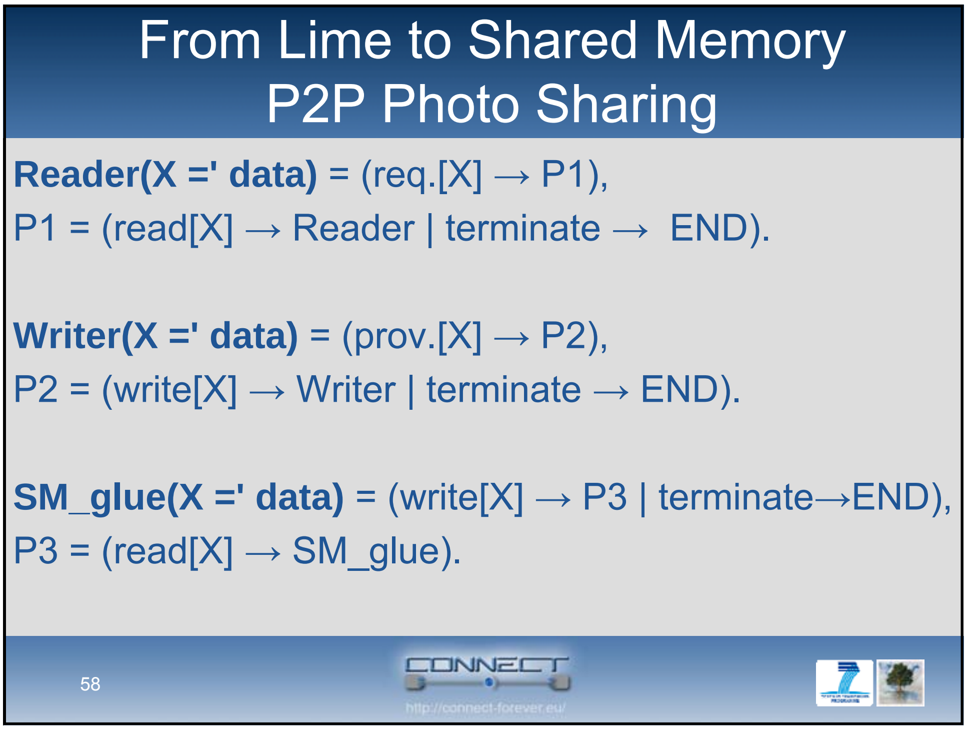
Middleware Agnostic C/S Photo Sharing LTS Semantics

<**Authenticate**, login, authenticationToken>
<**UploadPhoto**, {authenticationToken, photo}, acknowledgement>
<**SearchPhotos**, photoMetadata, photoMetadataList>
<**DownloadPhoto**, photoID, photoFile>
<**CommentPhoto**, photoComment, acknowledgement >
<$\overline{\textbf{CommentPhoto}}$, photoComment, acknowledgement>
<$\overline{\textbf{DownloadPhoto}}$, photoID, photoFile>
<$\overline{\textbf{SearchPhotos}}$, photoMetadata, photoMetadataList >
<$\overline{\textbf{Authenticate}}$, login, authenticationToken>
<$\overline{\textbf{UploadPhoto}}$, {photo, authenticationToken}, acknowledgement>

I) Photo-Sharing Producer

II) Photo-Sharing Consumer

III) Photo-Sharing Server

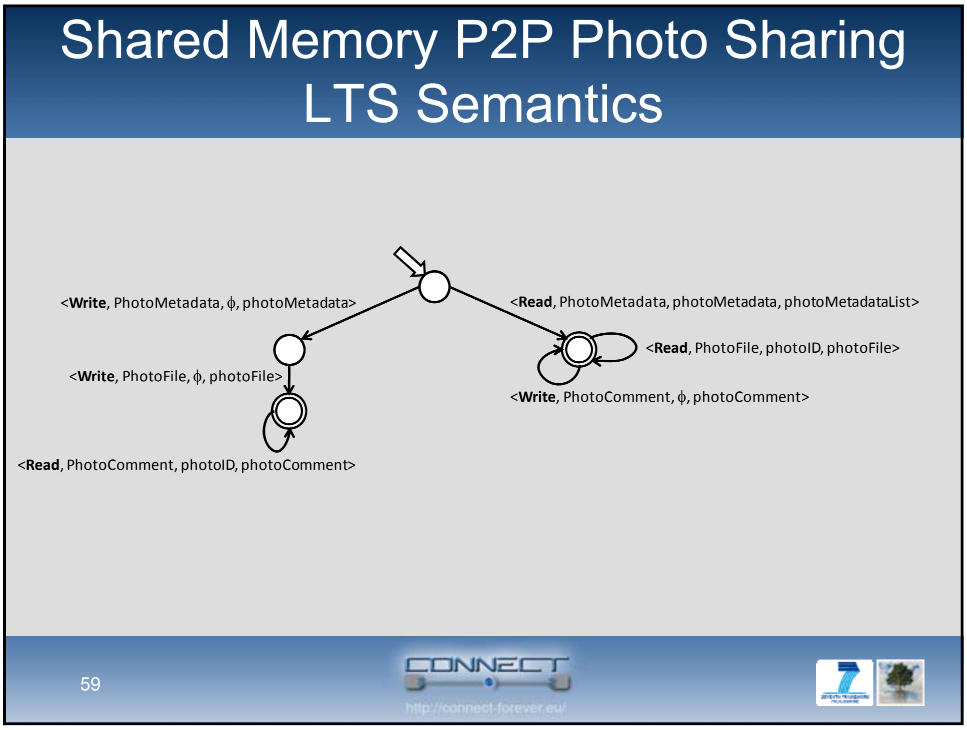


From Lime to Shared Memory P2P Photo Sharing

Reader(X =' data) = (req.[X] → P1),
P1 = (read[X] → Reader | terminate → END).

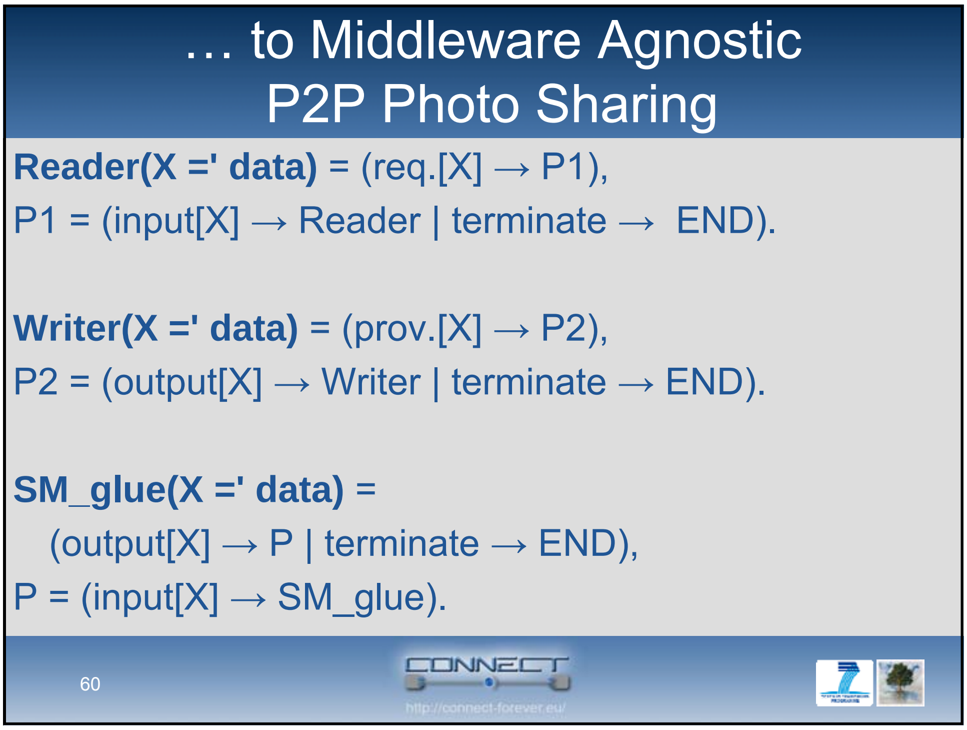
Writer(X =' data) = (prov.[X] → P2),
P2 = (write[X] → Writer | terminate → END).

SM_glue(X =' data) = (write[X] → P3 | terminate→END),
P3 = (read[X] → SM_glue).



Shared Memory P2P Photo Sharing LTS Semantics

<**Write**, PhotoMetadata, ϕ, photoMetadata>
<**Read**, PhotoMetadata, photoMetadata, photoMetadataList>
<**Read**, PhotoFile, photoID, photoFile>
<**Write**, PhotoFile, ϕ, photoFile>
<**Write**, PhotoComment, ϕ, photoComment>
<**Read**, PhotoComment, photoID, photoComment>



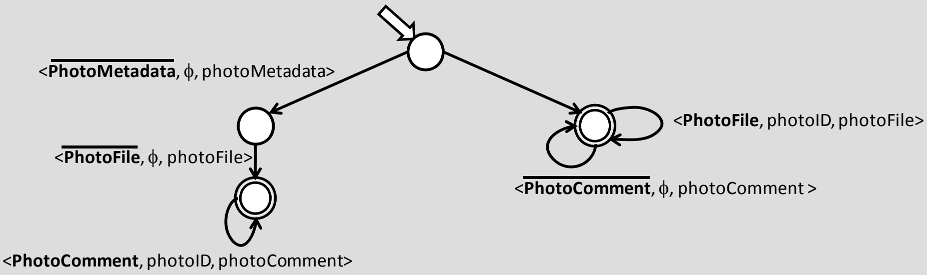
… to Middleware Agnostic P2P Photo Sharing

Reader(X =' data) = (req.[X] → P1),
P1 = (input[X] → Reader | terminate → END).

Writer(X =' data) = (prov.[X] → P2),
P2 = (output[X] → Writer | terminate → END).

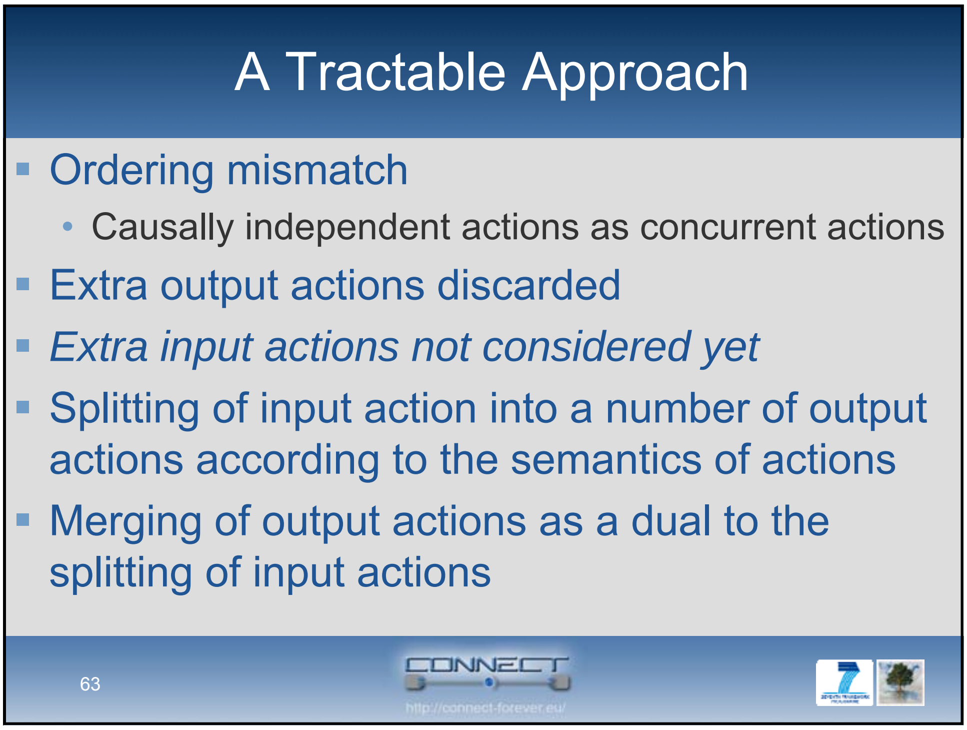
SM_glue(X =' data) =
(output[X] → P | terminate → END),
P = (input[X] → SM_glue).

Middleware Agnostic P2P Photo Sharing LTS Semantics


<$\overline{\textbf{PhotoMetadata}}$, ϕ, photoMetadata>
<$\overline{\textbf{PhotoFile}}$, ϕ, photoFile>
<**PhotoComment**, photoID, photoComment>
<**PhotoFile**, photoID, photoFile>
<$\overline{\textbf{PhotoComment}}$, ϕ, photoComment >

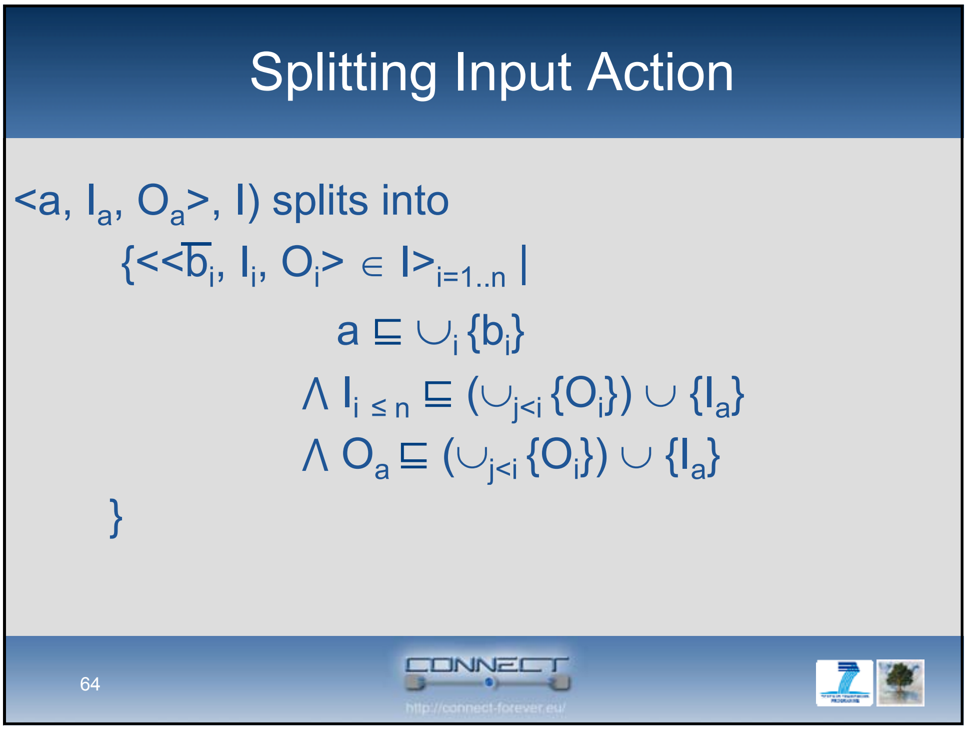
3. Interface Mapping

- Solving behavioral mismatches for input and output actions
  - Input actions must be synchronized with output actions
  - Associated mediator synthesis known as a computationally hard problem
  - Focus on basic mediation patterns
    - Ordering mismatches
    - Extra output actions
    - Extra input actions
    - Splitting of actions
    - Merging of actions



A Tractable Approach

- Ordering mismatch
  - Causally independent actions as concurrent actions
- Extra output actions discarded
- *Extra input actions not considered yet*
- Splitting of input action into a number of output actions according to the semantics of actions
- Merging of output actions as a dual to the splitting of input actions



Splitting Input Action

$<a, I_a, O_a>, I)$ splits into

$\{<<\overline{b}_i, I_i, O_i> \in I>_{i=1..n} \mid$

$a \sqsubseteq \cup_i \{b_i\}$

$\wedge\ I_{i \leq n} \sqsubseteq (\cup_{j<i} \{O_j\}) \cup \{I_a\}$

$\wedge\ O_a \sqsubseteq (\cup_{j<i} \{O_j\}) \cup \{I_a\}$

$\}$

Computing Interface Mapping

$$Map_I(I_{A_1}, I_{A_2}) = \cup_{<a,\, I,\, O> \in IA_1} \{<a, I, O> \rightarrow map(<a, I, O>, I_{A_2})\} \cup$$
$$\cup_{<a',\, I',\, O'> \in IA_2} \{<a', I', O'> \rightarrow map(<a', I', O'>, I_{A_1})\}$$

with:

$$map(<a, I_a, O_a>, I) = \{<<b_i, I_i, O_i> \in I>_{i=1..n} \mid$$
$$a \sqsubseteq \cup_i \{b_i\}$$
$$\wedge\ I_{i \leq n} \sqsubseteq (\cup_{j<i} \{O_j\}) \cup \{I_a\}$$
$$\wedge\ O_a \sqsubseteq (\cup_{j<i} \{O_i\}) \cup \{I_a\}$$
$$\}$$

and:

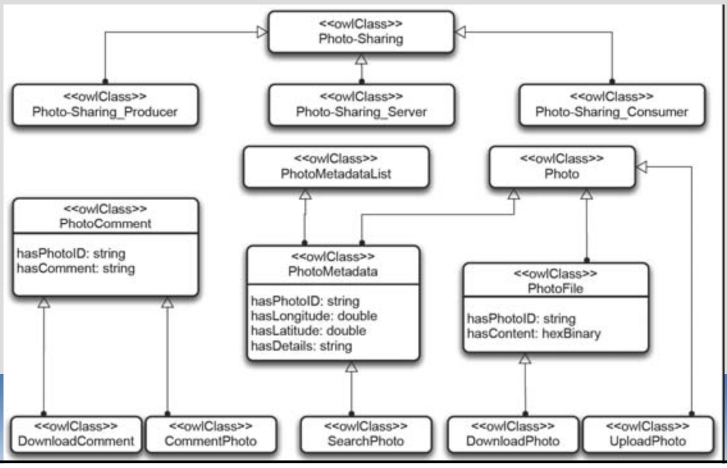
$\forall seq_1 \in map(<a, I_a, O_a>, I), \nexists\ seq_2 \in map(<a, I_a, O_a>, I) \mid seq_2 < seq_1$

Interface Mapping between Photo Sharing Systems

$Map(Int'_{\text{photo sharing consumer}}, Int_{\text{photo sharing}}) = \{$

$< \underline{SearchPhotos, < photo}Metadata >, < photoMetadataList >>$

$\rightarrow \{<< PhotoMetadata, \phi, < photoMetadata >>>\},$



4. Behavioral Matching

- Must ensure that networked systems are able to synchronize
  - According to the matching of respective actions
  - Possibly mediated according to supported mediation patterns, i.e., computed interface mapping

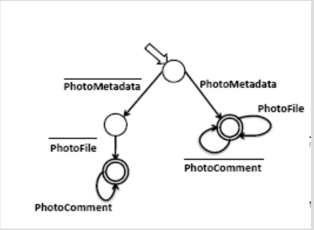
→ Mediated matching that amounts to a base model checking problem

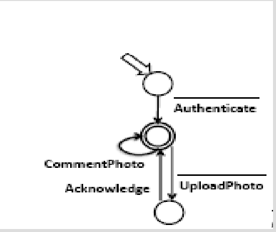
$$P_1 \mid\mid M_1 \leq P2 \mid\mid M_2$$

with ≤ denoting trace refinement and A1 req A2

Reasoning about Mediated Matching

Inclusion of LTS traces as the basis

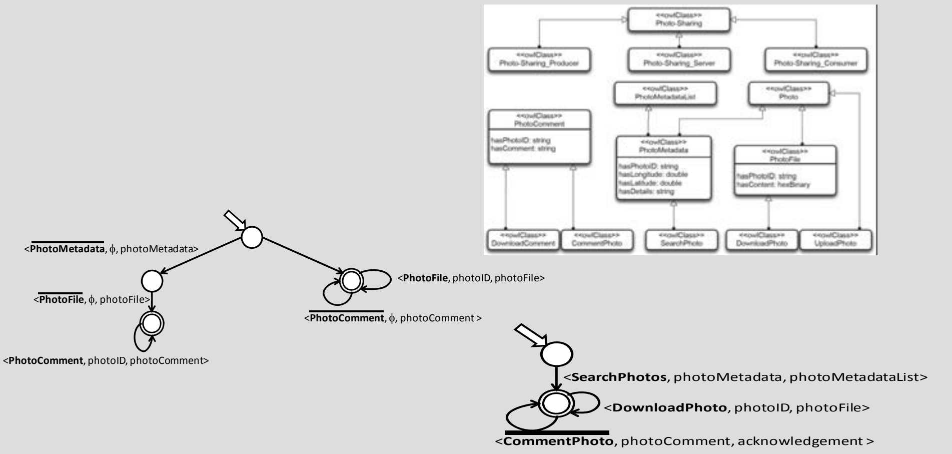




Behavioral matchmaking under:
mapping of semantic-based actions

Leveraging the rich SOTA on protocol conversion/mediation since the 80s

Interoperable Systems at Abstract Level



<<owlClass>> Photo-Sharing
<<owlClass>> Photo-Sharing_Producer
<<owlClass>> Photo-Sharing_Server
<<owlClass>> Photo-Sharing_Consumer
<<owlClass>> PhotoMetadataList
<<owlClass>> Photo
<<owlClass>> PhotoComment
hasPhotoID: string
hasComment: string
<<owlClass>> PhotoMetadata
hasPhotoID: string
hasLongitude: double
hasLatitude: double
hasDetails: string
<<owlClass>> PhotoFile
hasPhotoID: string
hasContent: hexBinary
<<owlClass>> DownloadComment
<<owlClass>> CommentPhoto
<<owlClass>> SearchPhoto
<<owlClass>> DownloadPhoto
<<owlClass>> UploadPhoto
<$\overline{\text{PhotoMetadata}}$, ϕ, photoMetadata>
<$\overline{\text{PhotoFile}}$, ϕ, photoFile>
<PhotoComment, photoID, photoComment>
<PhotoFile, photoID, photoFile>
<$\overline{\text{PhotoComment}}$, ϕ, photoComment >
<SearchPhotos, photoMetadata, photoMetadataList>
<DownloadPhoto, photoID, photoFile>
<$\overline{\text{CommentPhoto}}$, photoComment, acknowledgement >

5. Mediator Synthesis

Adaptation processes:

M_a = ($||_i$ Processes that merge/split A1 actions)

$M_{ai} = b'_1 \rightarrow \ldots \rightarrow b'_n \rightarrow a_i \rightarrow M_{ai}$

$M_{b'}$ = ($||_i$ Processes that consume extra output actions of A2)

$M_{b'i} = b'_1 \rightarrow M_{b'i}$

$M_{a'}$ = ($||_{i'}$ Processes that merge/split A2 actions)

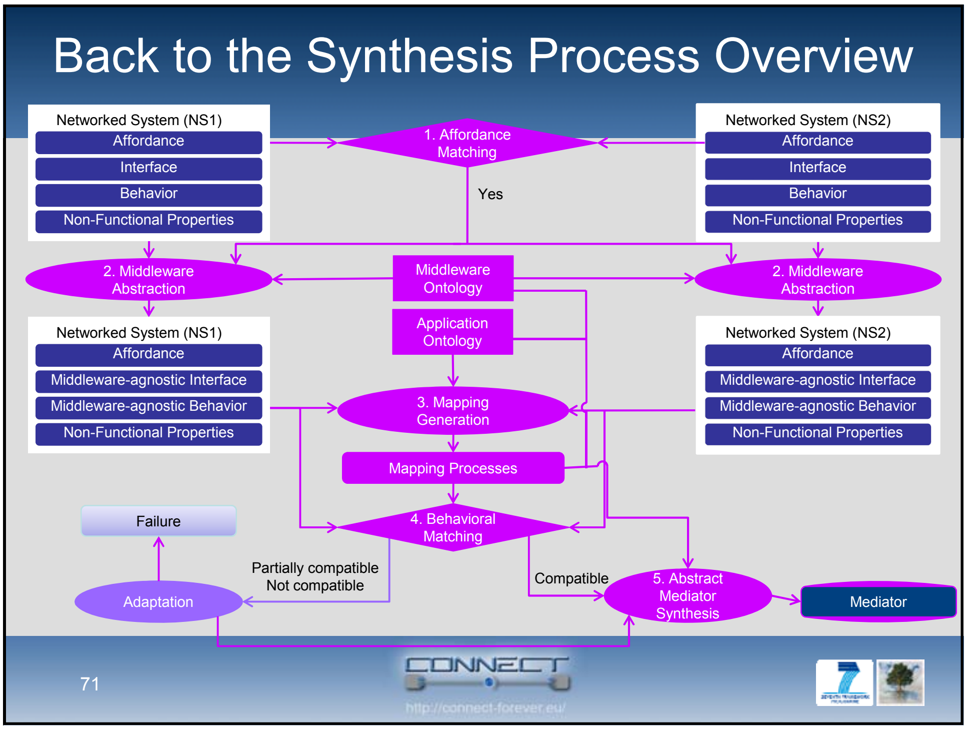
M_b = ($||_i$ Processes that consume extra output actions of A1)

Behavioral matching under mediation:

$P1 \,||\, M_a \,||\, M_{b'} \leq P2 \,||\, M_{a'} \,||\, M_b$ *where A1* ⊑ A2 and A1 *req A2*

Emergent connector:

$$M_a \,||\, M_{b'} \,||\, M_{a'} \,||\, M_b$$



Back to the Synthesis Process Overview

Networked System (NS1)
Affordance
Interface
Behavior
Non-Functional Properties

1. Affordance Matching

Yes

Networked System (NS2)
Affordance
Interface
Behavior
Non-Functional Properties

2. Middleware Abstraction

Middleware Ontology

2. Middleware Abstraction

Networked System (NS1)
Affordance
Middleware-agnostic Interface
Middleware-agnostic Behavior
Non-Functional Properties

Application Ontology

3. Mapping Generation

Networked System (NS2)
Affordance
Middleware-agnostic Interface
Middleware-agnostic Behavior
Non-Functional Properties

Mapping Processes

Failure

4. Behavioral Matching

Partially compatible
Not compatible

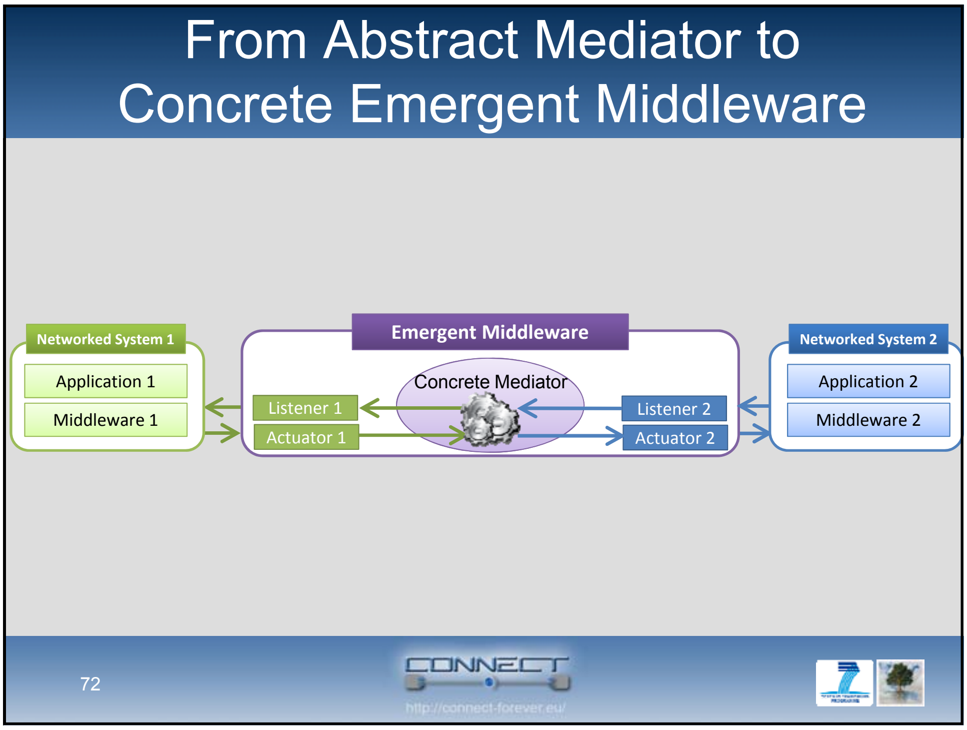
Compatible

Adaptation

5. Abstract Mediator Synthesis

Mediator

CONNECT
http://connect-forever.eu/



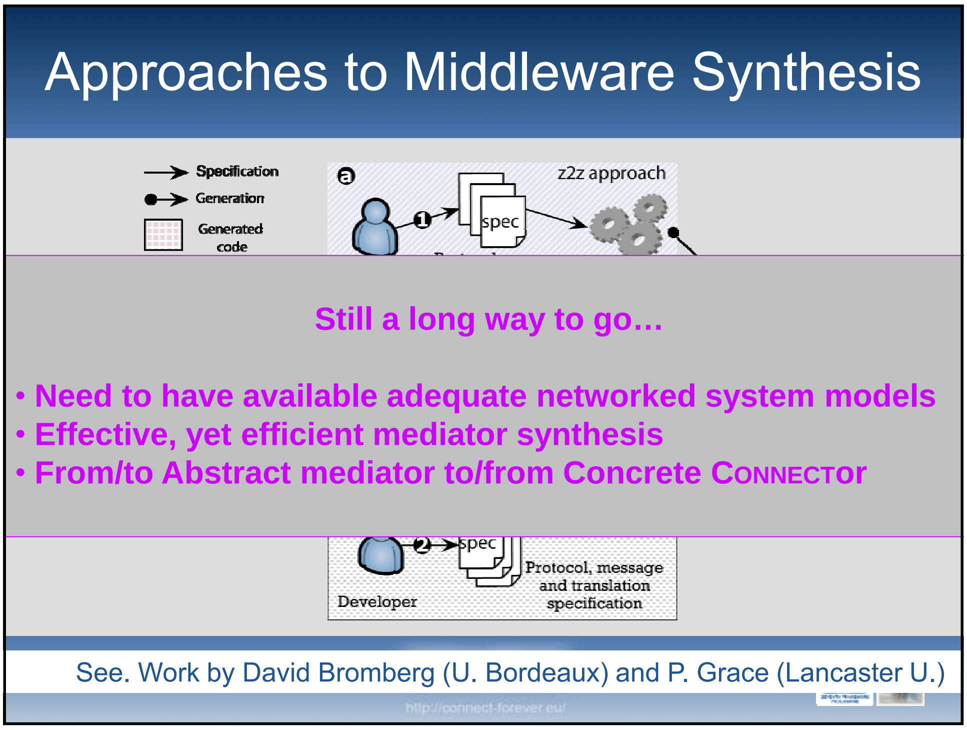
From Abstract Mediator to Concrete Emergent Middleware

Networked System 1
Application 1
Middleware 1

Emergent Middleware
Concrete Mediator
Listener 1
Actuator 1
Listener 2
Actuator 2

Networked System 2
Application 2
Middleware 2

CONNECT
http://connect-forever.eu/



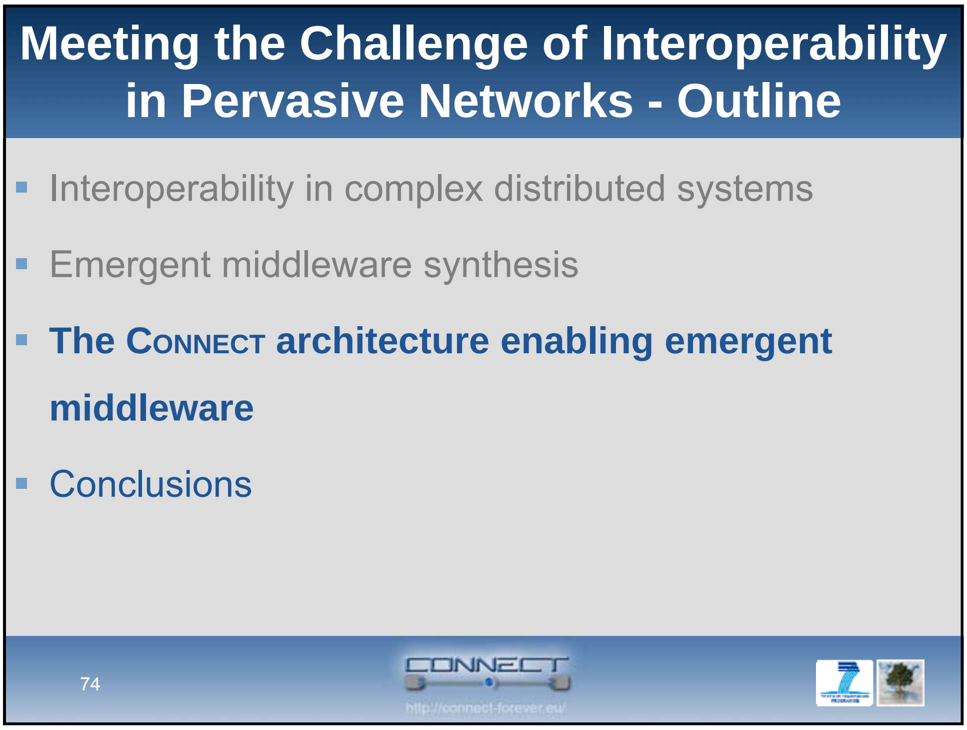
Approaches to Middleware Synthesis

Specification
Generation
Generated code
a
z2z approach
1
spec
2
spec
Protocol, message and translation specification
Developer

Still a long way to go…

- **Need to have available adequate networked system models**
- **Effective, yet efficient mediator synthesis**
- **From/to Abstract mediator to/from Concrete CONNECTOR**

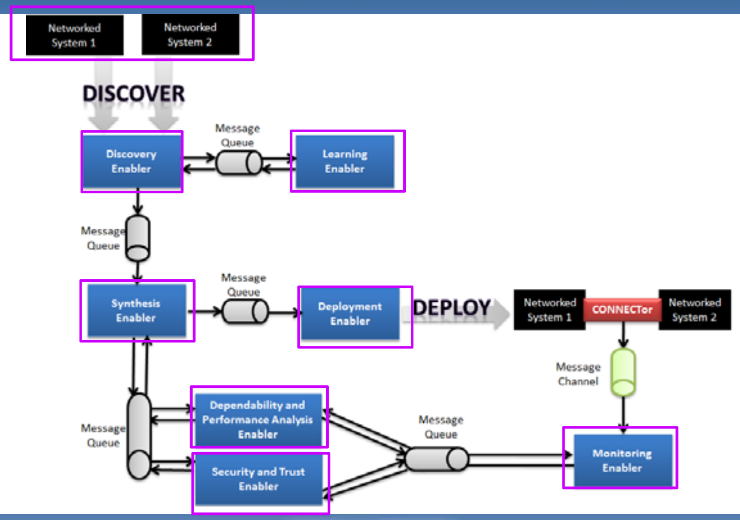
See. Work by David Bromberg (U. Bordeaux) and P. Grace (Lancaster U.)



Meeting the Challenge of Interoperability in Pervasive Networks - Outline

- Interoperability in complex distributed systems
- Emergent middleware synthesis
- **The CONNECT architecture enabling emergent middleware**
- Conclusions

The CONNECT Architecture of Enablers



Networked System 1

Networked System 2

DISCOVER

Discovery Enabler

Message Queue

Learning Enabler

Message Queue

Synthesis Enabler

Message Queue

Deployment Enabler

DEPLOY

Networked System 1

CONNECTor

Networked System 2

Message Channel

Message Queue

Dependability and Performance Analysis Enabler

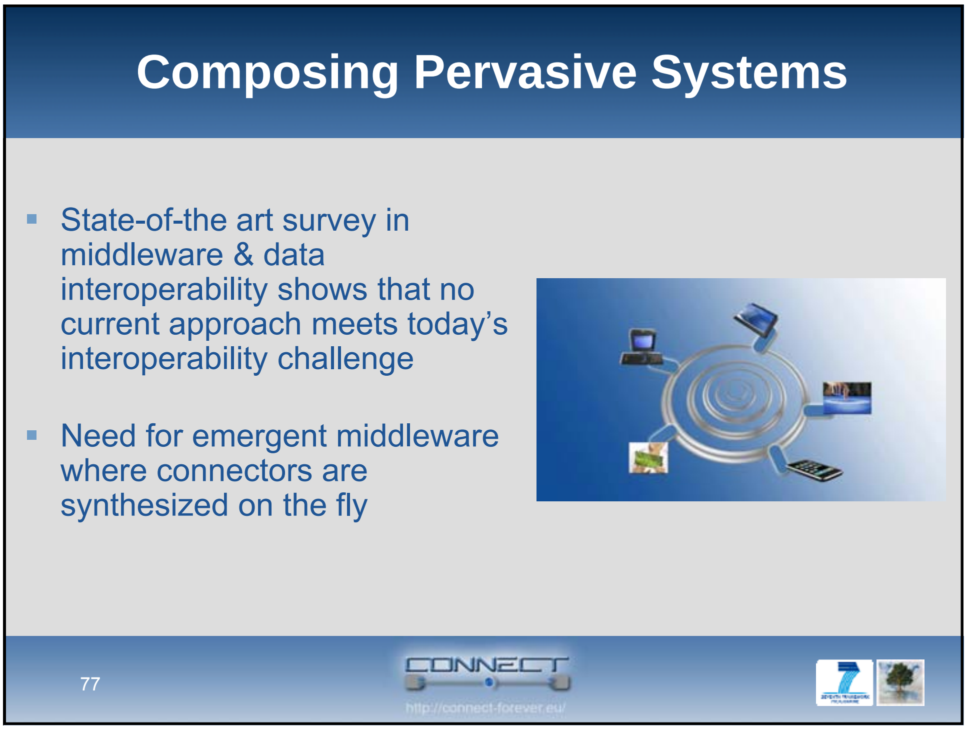
Security and Trust Enabler

Message Queue

Monitoring Enabler

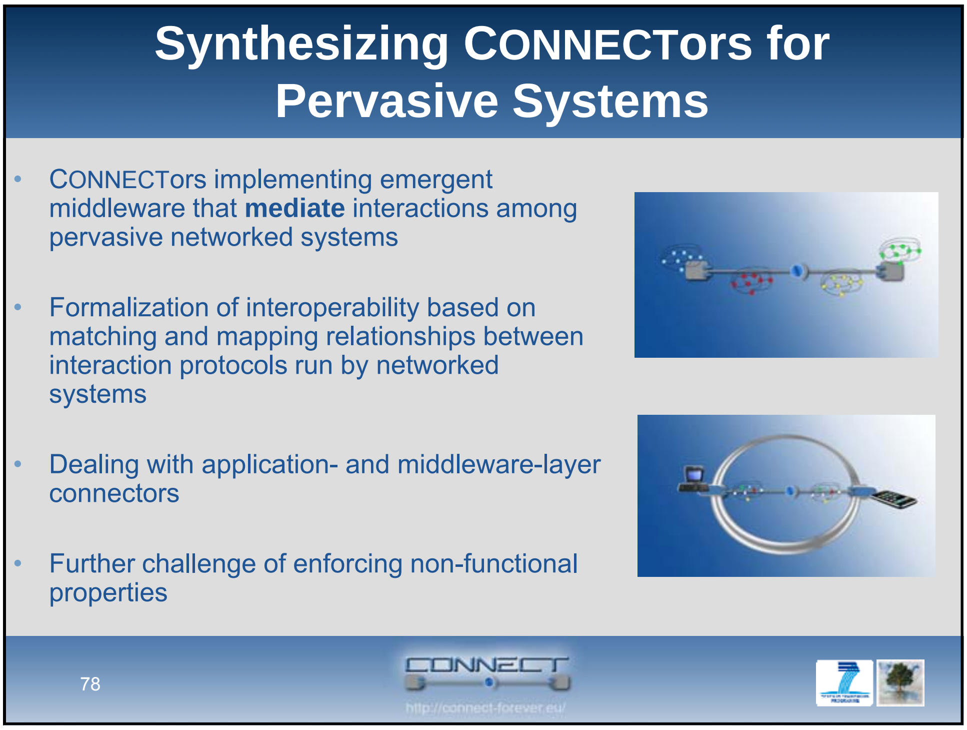
Meeting the Challenge of Interoperability in Pervasive Networks - Outline

- Interoperability in complex distributed systems
- Emergent middleware synthesis
- The CONNECT architecture enabling emergent middleware
- **Conclusions**



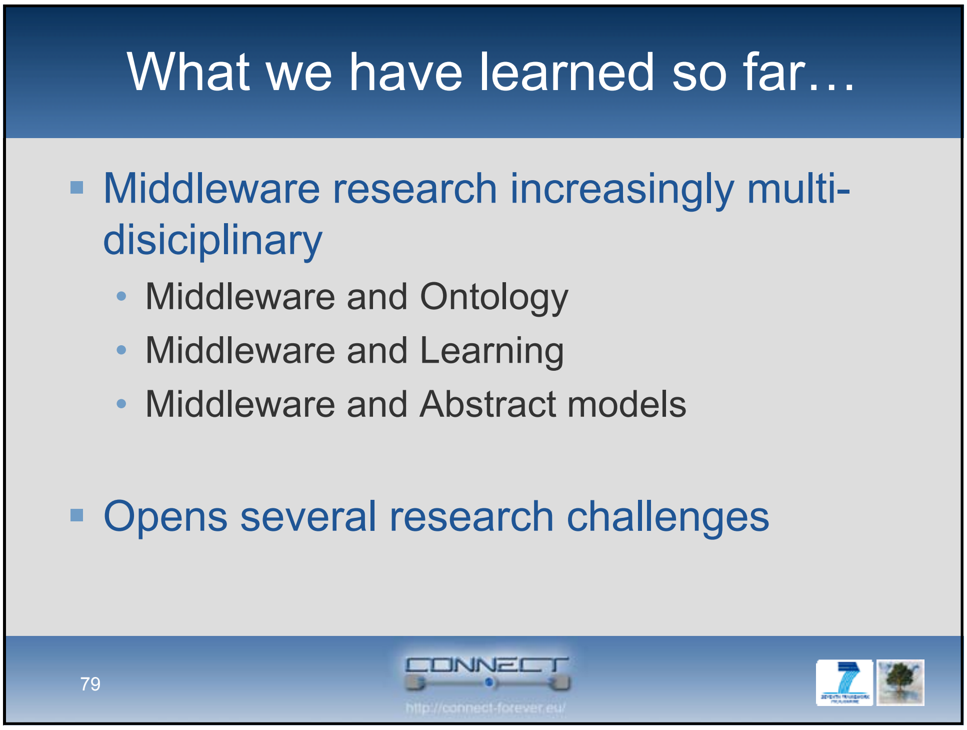
Composing Pervasive Systems

- State-of-the art survey in middleware & data interoperability shows that no current approach meets today's interoperability challenge
- Need for emergent middleware where connectors are synthesized on the fly



Synthesizing CONNECTors for Pervasive Systems

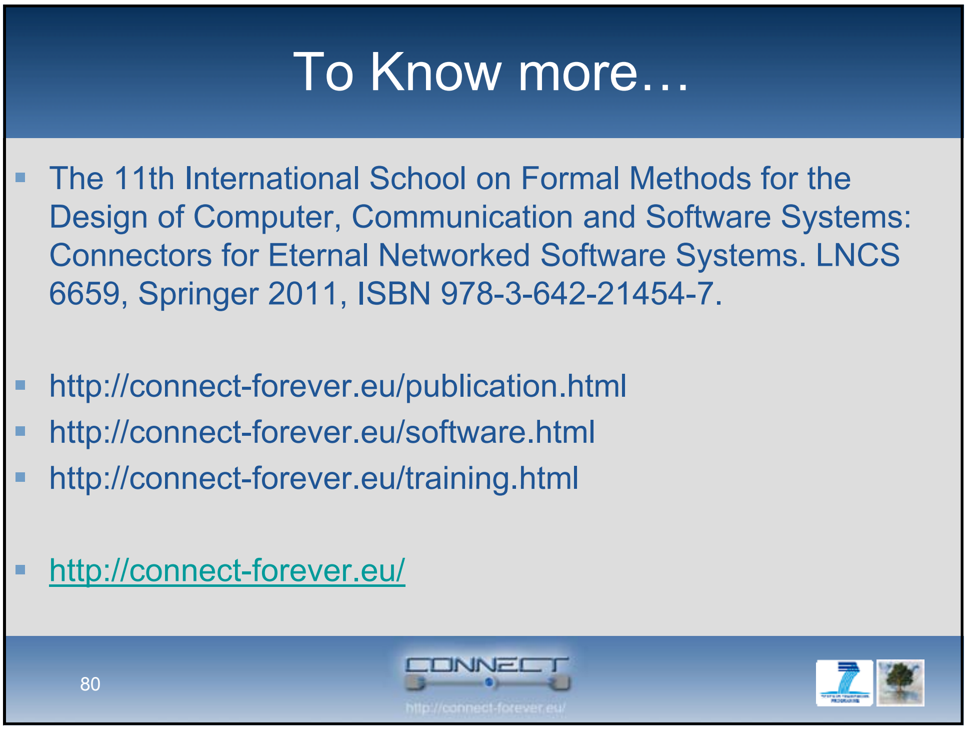
- CONNECTors implementing emergent middleware that **mediate** interactions among pervasive networked systems
- Formalization of interoperability based on matching and mapping relationships between interaction protocols run by networked systems
- Dealing with application- and middleware-layer connectors
- Further challenge of enforcing non-functional properties



What we have learned so far…

- Middleware research increasingly multi-disiciplinary
  - Middleware and Ontology
  - Middleware and Learning
  - Middleware and Abstract models
- Opens several research challenges

http://connect-forever.eu/



To Know more…

- The 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Connectors for Eternal Networked Software Systems. LNCS 6659, Springer 2011, ISBN 978-3-642-21454-7.
- http://connect-forever.eu/publication.html
- http://connect-forever.eu/software.html
- http://connect-forever.eu/training.html
- http://connect-forever.eu/

CONNECT

Emergent Connectors for
Eternal Software Intensive Networked Systems

Sweden
Germany
China
U.K.
France
Italy
INRIA
THALES
docomo

Thank you

Questions?

INRIA
docomo
DOCOMO Euro-Labs
LANCASTER UNIVERSITY
THALES
Università dell'Aquila
tu technische universität dortmund
UNIVERSITY OF OXFORD
UPPSALA UNIVERSITET