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

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.
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

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 Focus
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

1. Discovery Protocol Interoperability
2. Middleware Protocol Interoperability

CORBA Service (IIOP)  Web Service (SOAP)

3. Data Interoperability

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

<photo>
  <resolution> 72x72 </resolution>
  <unit> DPI </unit>
  <format> jpg </format>
</photo>

<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

- 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…

Bridging

2. One 3rd party translator

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

- 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

- 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

- 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?

Meeting the Challenge of Interoperability in Pervasive Networks - Outline

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

Classifying Connection Mismatches

- Syntactic
- Semantic
- Data
- Behavioral
- Application-level
  - Business process/logic
  - Operation granularity
- Middleware-level
  - Coordination model (C/S, P/S, ...)
  - Coordination model instantiation
Application Mismatch Example

Middleware Mismatch Example
Mediation Connector *aka* Emergent 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”

- Photo sharing using SOAP
- Photo sharing using LIME

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

**Talking the Same Language: The Key Role of Ontology**

- Photo sharing using SOAP
  - $\mathcal{P}$
  - $O_{\text{Photo}}$
  - $O_{\text{SOAP}}$
- Photo sharing using LIME
  - $\mathcal{Q}$
  - $O_{\text{Photo}}$
  - $O_{\text{LIME}}$

**Connection matchmaking?**

**Talking the Same Language**

- **Aligned Ontology**
  - $\mathcal{P}$
  - $O_{\text{Photo}}$
  - $O_{\text{SOAP}}$
  - $O_{\text{MW}}$
  - $O_{\text{Photo}}$
  - $O_{\text{LIME}}$

- Translated Protocols
  - $\mathcal{P}'$
  - $O_{\text{Photo}}$
  - $O_{\text{MW}}$

- Semantic & Behavioral matchmaking?
  - $\mathcal{Q}'$
  - $O_{\text{Photo}}$
  - $O_{\text{MW}}$

**Interoperate?**
Networked System Model for On-the-fly Connection

- Interface definition leveraging Semantic Web Service technologies
  - Affordance *aka* Capability
    - \(<\text{Type}, \text{Concept}, \text{Inputs}, \text{Outputs}\rangle\)
  - Interface signature
    - Action defined as \(<\text{Md}, \text{Application}, I, O\rangle\)
  - Affordance behavior
  - Non-functional properties

Reasoning about Networked Systems Models

- Ontologies to formalize the semantics of affordances and actions
The Photo Sharing Ontology

Classified according to subsumption relationships

C/S Photo Sharing Interface

```
interface photoSharingProducer = {
    <SOAP-ENVELOPE, <authenticationToken>, <photo>, <acknowledgment>;
    <SOAP-ENVELOPE, <photoID>, <photoFile>;
    <SOAP-ENVELOPE, <photoMetaList>, <photoID>;
    <SOAP-ENVELOPE, <photoComment>, <photoCommentID>;
    <SOAP-ENVELOPE, <photoFile>;
}
interface photoSharingConsumer = {
    <SOAP-ENVELOPE, <login>, <photoID>;
    <SOAP-ENVELOPE, <photoID>, <photoFile>;
    <SOAP-ENVELOPE, <photoMetaList>, <photoID>;
    <SOAP-ENVELOPE, <photoComment>, <photoCommentID>;
    <SOAP-ENVELOPE, <photoFile>;
}
interface photoSharingServer = {
    <SOAP-ENVELOPE, <authenticationToken>, <login>;
    <SOAP-ENVELOPE, <photoID>, <photoMetaList>;
    <SOAP-ENVELOPE, <photoID>, <photoFile>;
    <SOAP-ENVELOPE, <photoComment>, <photoCommentID>;
    <SOAP-ENVELOPE, <photoFile>;
```
P2P Photo Sharing Interface

\[
\text{Interface}_{\text{photo\_sharing}} = \{
\begin{array}{l}
\text{Out, PhotoMetadata, }, \emptyset, \langle \text{photoMetadata} \rangle, \\
\text{Out, PhotoFile}, \emptyset, \langle \text{photoFile} \rangle, \\
\text{Rdg, PhotoMetadata}, \langle \text{photoMetadata} \rangle, \langle \text{photoMetadataList} \rangle, \\
\text{Rd, PhotoFile, }, \langle \text{photoID} \rangle, \langle \text{photoFile} \rangle, \\
\text{Rd, PhotoComment}, \langle \text{photoID} \rangle, \langle \text{photoComment} \rangle, \\
\text{Out, PhotoComment, }, \emptyset, \langle \text{photoComment} \rangle, \\
\text{In, PhotoComment}, \langle \text{photoID} \rangle, \langle \text{photoComment} \rangle, \\
\text{Rd, PhotoComment, }, \langle \text{photoID} \rangle, \langle \text{photoComment} \rangle
\end{array}
\}
\]
**FSP: Finite State Processes**

END
set S
[i : S]

**Primitve Processes (P)**

| a → P | Action prefix |
| a → 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-RPCCall, SOAP-RPCReceiveCall, SOAP-RPCReply, SOAP-RPCReceiveReply} → GlueSOAP

||ConnectorSOAP = ClientSOAP || GlueSOAP || ServerSOAP

See Work by D. Garlan et al. at CMU

http://connect.cs.cmu.edu
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).

ClientSOAP (X = op) =
    (req.[X] → P1 | terminate → END),

ServerSOAP (X = op) =
    (prov.[X] → P2 | terminate → END),

GlueSOAP (X = op) =
    (SOAP-RPCCall[X] → P0 | terminate → END),
P0 = (SOAP-RPCReceiveCall[X] → SOAP-RPCReply[X]
C/S Photo Sharing over SOAP
- Photo Sharing System -

\[\text{SOAP \_PhotoSharing} = \]
\[\text{(PhotoSharingProducer} \]
\[\text{\| PhotoSharingConsumer} \]
\[\text{\| PhotoSharingServer} \]
\[\text{\| (forall [op:SOAP\_PhotoSharing\_Actions] ServerSOAP (op))} \]
\[\text{\| (forall [op:SOAP\_PhotoSharing\_Actions] ClientSOAP (op))} \]
\[\text{\| (forall [op:SOAP\_PhotoSharing\_Actions] GlueSOAP (op))}.\]

P2P Photo Sharing over LIME
- Application -

set Lime\_PhotoSharing\_Actions = \{photoMetadata, photoFile, photoComment\}

\[\text{PhotoSharingPeer} = (req.photoMetadata \rightarrow \text{Consumer} \]
\[\text{\| prov.photoMetadata \rightarrow \text{Producer}),} \]
Producer = (prov.photoFile \rightarrow \text{PhotoSharingPeer}),
Consumer = (req.photoFile \rightarrow \text{Consumer} \]
\[\text{\| req.photoComment \rightarrow \text{Consumer} \]
\[\text{\| prov.photoComment \rightarrow \text{Consumer} \]
\[\text{\| req.photoFile \rightarrow \text{PhotoSharingPeer} \]
\[\text{\| req.photoComment \rightarrow \text{PhotoSharingPeer} \]
\[\text{\| prov.photoComment \rightarrow \text{PhotoSharingPeer} \]
\[\text{\| terminate \rightarrow END}).} \]
P2P Photo Sharing over LIME
- LIME Middleware -

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

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

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

```
<table>
<thead>
<tr>
<th>Networked System (NS1)</th>
<th>Networked System (NS2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affordance</td>
<td>Affordance</td>
</tr>
<tr>
<td>Interface</td>
<td>Interface</td>
</tr>
<tr>
<td>Behavior</td>
<td>Behavior</td>
</tr>
<tr>
<td>Non-Functional Properties</td>
<td>Non-Functional Properties</td>
</tr>
</tbody>
</table>

1. Affordance Matching

2. Middleware Abstraction

3. Mapping Generation

4. Behavioral Matching

5. Abstract Mediator Synthesis

6. Mediation
```

- Compatible
- Partially compatible
- Not compatible

---

Adaptation

Failure
1. Semantic Matching of Affordances

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

- $\text{Aff}_1 = \langle \text{Req}, F_1, I_1, O_1 \rangle$, $\text{Aff}_2 = \langle \text{Prov}, F_2, I_2, O_2 \rangle$

- $\text{Aff}_1$ and $\text{Aff}_2$ semantically match iff:
  - $F_1 \subseteq F_2$
  - $I_2 \subseteq I_1$
  - $O_1 \subseteq O_2$

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

From SOAP to RPC C/S Photo Sharing

**Client (X='op1)** = (req.[X] → P1),


**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

I) Photo-Sharing Producer
- Call, Authenticate, login, φ
- ReceiveReply, Authenticate, φ, authenticationToken
  - Call, UploadPhoto, (authenticationToken, photo), φ
  - ReceiveReply, UploadPhoto, φ, acknowledgement
- < ReceiveReply, CommentPhoto, φ, acknowledgement >
- < ReceiveReply, SearchPhotos, φ, photoMetadataList >
- < ReceiveReply, DownloadPhoto, φ, photoFile >
- < ReceiveReply, CommentPhoto, φ, acknowledgement >
- < ReceiveReply, UploadPhoto, φ, acknowledgement >
- < ReceiveReply, Authenticate, φ, authenticationToken >
- < ReceiveReply, Authenticate, φ, authenticationToken >

II) Photo-Sharing Consumer
- Call, SearchPhotos, φ, photoMetadataList
- < Call, DownloadPhoto, φ, photoID >
- < Call, CommentPhoto, φ, photoComment >
- < ReceiveCall, SearchPhotos, φ, photoMetadataList >
- < Reply, SearchPhoto, φ, photoMetadataList >
- < Reply, DownloadPhoto, φ, photoFile >
- < Reply, Authenticate, φ, acknowledgement >
- < Reply, Authenticate, φ, acknowledgement >
- < Reply, Authenticate, φ, acknowledgement >
- < Reply, Authenticate, φ, acknowledgement >

III) Photo-Sharing Server
- Call, SearchPhotos, φ, photoMetadataList
- < Call, DownloadPhoto, φ, photoID >
- < Call, CommentPhoto, φ, photoComment >
- < ReceiveCall, SearchPhotos, φ, photoMetadataList >
- < Reply, SearchPhoto, φ, photoMetadataList >
- < Reply, DownloadPhoto, φ, photoFile >
- < Reply, Authenticate, φ, acknowledgement >
- < Reply, Authenticate, φ, acknowledgement >
- < Reply, Authenticate, φ, acknowledgement >
- < Reply, Authenticate, φ, acknowledgement >

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

SearchPhotos, photoMetadata, photoMetadataList

DownloadPhoto, photoID, photoFile

Authenticate, login, authenticationToken

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),

SM_glue(X = 'data) = (write[X] → P3 | terminate→END),
P3 = (read[X] → SM_glue).
Shared Memory P2P Photo Sharing
LTS Semantics

... to Middleware Agnostic
P2P Photo Sharing

\[\text{Reader}(X = 'data) = (\text{req} . [X] \to P1), \]
P1 = (input[X] \to \text{Reader} | \text{terminate} \to \text{END}).

\[\text{Writer}(X = 'data) = (\text{prov} . [X] \to P2), \]
P2 = (output[X] \to \text{Writer} | \text{terminate} \to \text{END}).

\[\text{SM}\_\text{glue}(X = 'data) = \]
\(\text{SM}\_\text{glue}(X = 'data) = \)
\((\text{output}[X] \to P | \text{terminate} \to \text{END}), \)
P = (input[X] \to \text{SM}\_\text{glue}).\]
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

\[ <a, I_a, O_a>, I \rangle \text{ splits into} \]
\[ \{<b_i, I_i, O_i> \in I \}_{i=1..n} \mid \]
\[ a \subseteq \bigcup_i \{b_i\} \]
\[ \land I_{i \leq n} \subseteq (\bigcup_{j<i} \{O_j\}) \cup \{I_a\} \]
\[ \land O_a \subseteq (\bigcup_{j<i} \{O_j\}) \cup \{I_a\} \]


Computing Interface Mapping

\[ \text{Map}(I_{A_1}, I_{A_2}) = \bigcup_{<a, I, O> \in I_{A_1}} \{<a, I, O> \rightarrow \text{map}(<a, I, O>, I_{A_2})\} \cup \bigcup_{<a', I', O'> \in I_{A_2}} \{<a', I', O'> \rightarrow \text{map}(<a', I', O'>, I_{A_1})\} \]

with:

\[ \text{map}(<a, I_a, O_a>, I) = \{<b_i, I_i, O_i> \in I \mid a \subseteq \bigcup_i \{b_i\} \]
\[\text{and:} \quad \forall \text{seq}_1 \in \text{map}(<a, I_a, O_a>, I), \exists \text{seq}_2 \in \text{map}(<a, I_a, O_a>, I) \mid \text{seq}_2 < \text{seq}_1 \]

Interface Mapping between Photo Sharing Systems

\[ \text{Map}(\text{Int}^*_{\text{photo sharing consumer}}, \text{Int}^*_{\text{photo sharing}}) = \{ \]
\[ \langle \text{SearchPhotos}, \text{photoMetadata}, \text{photoMetadataList} \rangle \rightarrow \{ \langle \text{PhotoMetadata}, \phi, \text{photoMetadataList} \rangle \}. \]
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

\[ P_1 \| M_1 \leq P_2 \| M_2 \]

with \( \leq \) 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

5. Mediator Synthesis

Adaptation processes:

\[ M_a = (|| \text{Processes that merge/split A1 actions}) \]
\[ M_{ai} = b_1 \rightarrow \ldots \rightarrow b_n \rightarrow a_i \rightarrow M_{ai} \]
\[ M'_b = (|| \text{Processes that consume extra output actions of A2}) \]
\[ M'_{b1} = b'_1 \rightarrow M_{b1} \]
\[ M'_a = (|| \text{Processes that merge/split A2 actions}) \]
\[ M'_b = (|| \text{Processes that consume extra output actions of A1}) \]

Behavioral matching under mediation:

\[ P1 \parallel M_a || M_b \leq P2 \parallel M'_a || M'_b \text{ where } A1 \sqsubseteq A2 \text{ and } A1 \text{ req } A2 \]

Emergent connector:

\[ M_a || M'_b || M'_a || M_b \]
Back to the Synthesis Process Overview

1. Affordance Matching
   - Networked System (NS1)
     - Affordance
     - Interface
     - Behavior
     - Not-Functional Properties
   - Networked System (NS2)
     - Affordance
     - Interface
     - Behavior
     - Not-Functional Properties
   - Yes

2. Middleware Abstraction
   - Networked System (NS1)
     - Middleware-agnostic Interface
     - Middleware-agnostic Behavior
     - Not-Functional Properties
   - Networked System (NS2)
     - Middleware-agnostic Interface
     - Middleware-agnostic Behavior
     - Not-Functional Properties

3. Mapping Generation
   - Mapping Processes
   - Middleware Ontology
   - Application Ontology

4. Behavioral Matching
   - Partially compatible
   - Not compatible
   - Compatible
   - Mediator

5. Abstract Mediator Synthesis
   - Compatible

From Abstract Mediator to Concrete Emergent Middleware

Networked System 1:
- Application 1
- Middleware 1

Networked System 2:
- Application 2
- Middleware 2

Emergent Middleware
- Concrete Mediator
  - Listener 1
  - Actuator 1
  - Listener 2
  - Actuator 2
Approaches to Middleware Synthesis

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

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-disciplinary
  - Middleware and Ontology
  - Middleware and Learning
  - Middleware and Abstract models

- Opens several research challenges

To Know more…


- http://connect-forever.eu/publication.html
- http://connect-forever.eu/software.html
- http://connect-forever.eu/training.html

- http://connect-forever.eu/
Thank you

Questions?