Emergent Middleware Facing the Interoperability Challenge

Valérie Issarny, Inria
Joint work with colleagues of ARLES team
and EC CONNECT project
Special thanks to Amel Bennaceur

Inria@SiliconValley
Outline

- The interoperability challenge
- Emergent middleware for on-the-fly interoperability
- Some initial experiments
- What’s next
The Interoperability Challenge

- Same functionality, various applications
  - Heterogeneous interfaces & behaviours
The Interoperability Challenge

• Same functionality, various applications, diverse middleware solutions

• Heterogeneous interfaces & behaviours across the protocol stack
The Interoperability Challenge

- Same functionality, various applications, diverse middleware solutions
- Increasingly connected world
Approaches to Interoperability

- A chosen shared language
- One speaker talks the other’s language
- One 3rd party translator, e.g., English to French translator
- Auxiliary Languages (e.g. Esperanto)
- Babel fish
Approaches to Interoperability

No one-size-fits-all standard
A chosen shared language
Client-side only plugins & a priori knowledge
One speaker talks the other’s language

Significant development effort for bridging
One 3rd party translator, e.g., English to French translator

Interoperability up to common ESB
Auxiliary Languages (e.g. Esperanto)

On-the-fly interoperability through emergent middleware
Babel fish
Achieving On-the-fly Interoperability

• Can we observe, learn, synthesize and deploy a binding dynamically

• Emergent middleware leveraging software engineering methods and tools
Model-based Emergent Middleware

Model of WebDAV Client

Model of Google Docs Service

Domain ontology

Synthesis

Model Extraction

Deployment

Model Extraction

Google docs

Mediator

Thing

Ressource

Document

Folder

SpreadSheetFile

Nothing

Download

Upload

Delete

Move

Replace

Model Extraction

Model Level

Running-System Level

Synthesis

Deployment

Model Extraction

Model Level

Running-System Level

Inria
Component Models

- Background from Semantic Web Services
- Ontology-based functional semantics
  - Capability
    - The high-level functionality of a system
  - Interface
    - A set of observable actions
- LTS-based behavioural semantics
  - The way the observable actions are coordinated
  - At both application and middleware layers
    - Application → Business logic
    - Middleware → Communication & coordination protocol

- Capability (CapWDAV)
  - Requires fileManagement
  - Interface signature (IWDAV)
    - <Authenticate, {Username, Password}, {Authorisation}>
    - <Lock, {SourceURI}, {Acknowledgment}>
    - <MoveFile, {SourceURI,DestinationURI}, {Acknowledgment}>
    - <ReadFile, {SourceURI}, {File}>
    - <Unlock, {SourceURI}, {Acknowledgment}>
    - ...

- Component Models
  - SendHTTPRequest [Authenticate] [Username, Password]
  - ReceiveHTTPResponse [Authenticate] [Authorisation]
  - Authenticate → Lock
  - Lock → MoveFile/ReadFile/WriteFile
  - Unlock
  - Logout
Model Extraction

Limited information in actual interfaces

- **Statistical learning** for inferring capability
- **Automata learning** for inferring behaviour
  - Passive vs Active?
  - **Active learning** based on L* algorithm
    - Start with the most general behaviour that allows any sequence of the operations of the interface to be executed
    - Test and refine when an interaction error, aka a counterexample, is discovered
  - Passive learning to refine the model
On-the-fly Mediator Synthesis

Overcoming the Heterogeneity of

Components’ interfaces

Components’ behaviours

Components’ communication protocols

Application layer

Middleware layer

Interface Matching

Generating Correct-by-Construction Mediators

Concretisation

→ many-to-many

→ under ambiguity

→ different interaction paradigms
Interface Matching: An Example

Google docs

<Authenticate, {Username, Password}, {Authorisation}>
<List, {SourceURI}, {Acknowledgment}>
<MoveFile, {SourceURI, DestinationURI}, {Acknowledgment}>
<ReadFile, {SourceURI}, {File}>
<Unlock, {SourceURI}, {Acknowledgment}>
...

(<Authenticate>, <Authenticate>)
(<Authenticate>, <SetSharingProperties>)
(<MoveFile>, <DownloadDocument>)

(<MoveFile>, <DownloadDocument, Authenticate>)
(<MoveFile>, <DownloadDocument, UploadDocument>)
(<MoveFile>, <DownloadDocument, DeleteDocument>)

(<MoveFile>, <DownloadDocument, UploadDocument SetSharingProperties>)
(<MoveFile>, <DownloadDocument, UploadDocument DeleteDocument>)
(<MoveFile>, <DownloadDocument, DeleteDocument UploadDocument>)
Interface Matching: Computation

Matching interface $\mathcal{I}_1$ to interface $\mathcal{I}_2$ consists in finding all pairs of actions such that a sequence of actions required by the former can be safely performed using a sequence of actions provided by the latter. In addition, all pairs are minimal.

$$Match(\mathcal{I}_1, \mathcal{I}_2) =$$
$$\{ (X_1, X_2) |$$
$$X_1 = \langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle, \alpha_{i=1..m} \in \mathcal{I}_1$$
$$\land X_2 = \langle \beta_1, \beta_2, \ldots, \beta_n \rangle, \beta_{j=1..n} \in \mathcal{I}_2$$
$$\land X_1 \mapsto X_2$$
$$\land \exists (X'_1, X'_2) | X'_1 = \langle \alpha_1, \alpha_2, \ldots, \alpha_{m'} \rangle, \alpha_{i=1..m'} \in \mathcal{I}_1$$
$$\land X'_2 = \langle \beta_1, \beta_2, \ldots, \beta_{n'} \rangle, \beta_{j=1..n'} \in \mathcal{I}_2$$
$$\land (X'_1 \mapsto X'_2)$$
$$\land (m' < m) \land (n' < n)\}$$

But... NP-Complete

Use Constraint programming with adequate ontology encoding
Mediator Synthesis: An Example

WebDAV Client
- Authenticate
- Lock
- Logout
- MoveFile/WriteFile

Google Docs Service
- Authenticate
- Logout
- Unlock
- SetSharingProperties
- UploadDocument
- DownloadDocument
- DeleteDocument
- MoveFile

Matching Processes
- Authenticate ↔ Authenticate
- Lock ↔ SetSharingProperties
- WriteFile ↔ UploadDocument
- Unlock ↔ SetSharingProperties
- Logout
- Unlock
- Logout
- Unlock
- Unlock
- Unlock
- Unlock
- Unlock

Concretisation
Generating Correct-by-Construction Mediators

- The mediator composes the mapping processes in order to allow both components, whose behaviours are $P_1$ and $P_2$, to coordinate and reach their final states.

The basic case

Translation

\[
\text{if } P_1 \xrightarrow{X_1} P_1' \text{ and } \exists (X_1, X_2) \in \text{Match} (\mathcal{I}_1, \mathcal{I}_2) \\
\text{such that } P_2 \xrightarrow{X_2} P_2' \text{ and } P_1' \leftrightarrow_{M'} P_2' \\
\text{then } P_1 \leftrightarrow_M P_2 \text{ where } M = M_{m-n}(X_1, X_2); M'
\]
From Abstract to Concrete Mediator

- Refine the synthesised mediator
  - Translating application data
    - Combining ontology relations with schema matching techniques
  - Coordinating middleware protocols
- Deploying the mediator
Coordinating middleware Protocols: An Example

Google docs

- DownloadDocument
- UploadDocument
- DeleteDocument
- MoveFile

Concretise

- ReceiveRequest [MoveFile][SourceURI, DestinationURI]
- Compute SourceURI
- SendRequest [DownloadDocument][SourceURI]
- ReceiveResponse [DownloadDocument][Document]
- Compute Metadata and Content from Document
- SendRequest [UploadDocument][Metadata, Content, DestinationURI]
- SendReply [MoveFile][Acknowledgment]
Interoperability across interaction paradigms

**Provided Action**

<op, i, a>

**Server**

- ReceiveRequest [methodName][argument]
- SendResponse [methodName][returnValue]

- op = methodName
- i = argument
- a = returnValue

**Client**

- SendRequest [methodName][argument]
- ReceiveResponse [methodName][returnValue]

**Required Action**

<op, i, a>

**RPC**

**Writer**

- Write [data]

**Reader**

- Read [dataChannel][data]

**Publisher**

- Publish [event]

**Subscriber**

- GetEvent [event]

**DSM**

- op = dataChannel
- i = data
- a = data

**Publish/Subscribe**

- op = eventType
- i = eventType
- a = event

**Interface Matching**

- Generating Mediators
- Concretisation
Deploying the Mediator

- Translate `req.lock` into `prov.setSharingProperties`
- Translate `req.move` into `prov.downloadDoc` followed by `prov.getUploadDoc` followed by `prov.getDeleteDoc`
- Translate `req.unlock` into `prov.setSharingProperties`
Applicability - Case Studies

**Emergency Management**
one-to-many interface matching
cross interaction patterns
mediation at runtime

**Event Management**
one-to-many interface matching
cross middleware solutions

**File Management**
one-to-many interface matching

**Instant Messaging**
one-to-one interface matching
Conclusion - Contributions

• Generating interface matching automatically
  • Dealing with one-to-many and many-to-many correspondence
• Synthesising correct-by-construction mediators
  • Dealing with ambiguity of interface matching
• Dealing with differences at both application and middleware layers

Dynamic Mediator Synthesis: From Theory to Practice
Conclusion – What’s next

• Increasingly complex distributed systems
  • Interoperability remains a central concern

• Emergent middleware as a promising solution
  • Central role of ontology and learning
  • Cross-layer messaging

➔ System properties that become highly dynamic