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

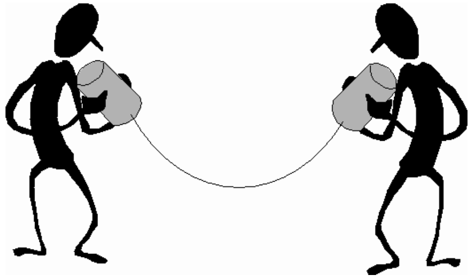
The Interoperability Challenge

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

Approaches to Interoperability

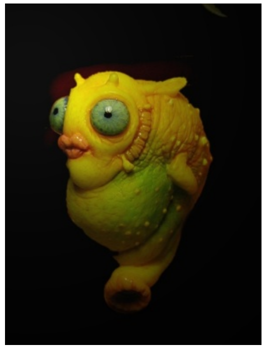
A chosen shared language

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



Auxiliary Languages (e.g. Esperanto)

One speaker talks the other's language



Babel fish

Approaches to Interoperability

No one-size-fits-all **standard**

A chosen shared language

Significant development effort for **bridging**

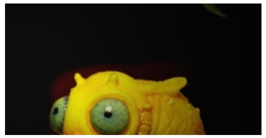
One 3rd party translator, e.g., English to French anslator

Interoperability up to common **ESB**

Auxiliary Languages (e.g. Esperanto)

Client-side only **plugins**
&
a priori knowledge

One speaker talks the other's language

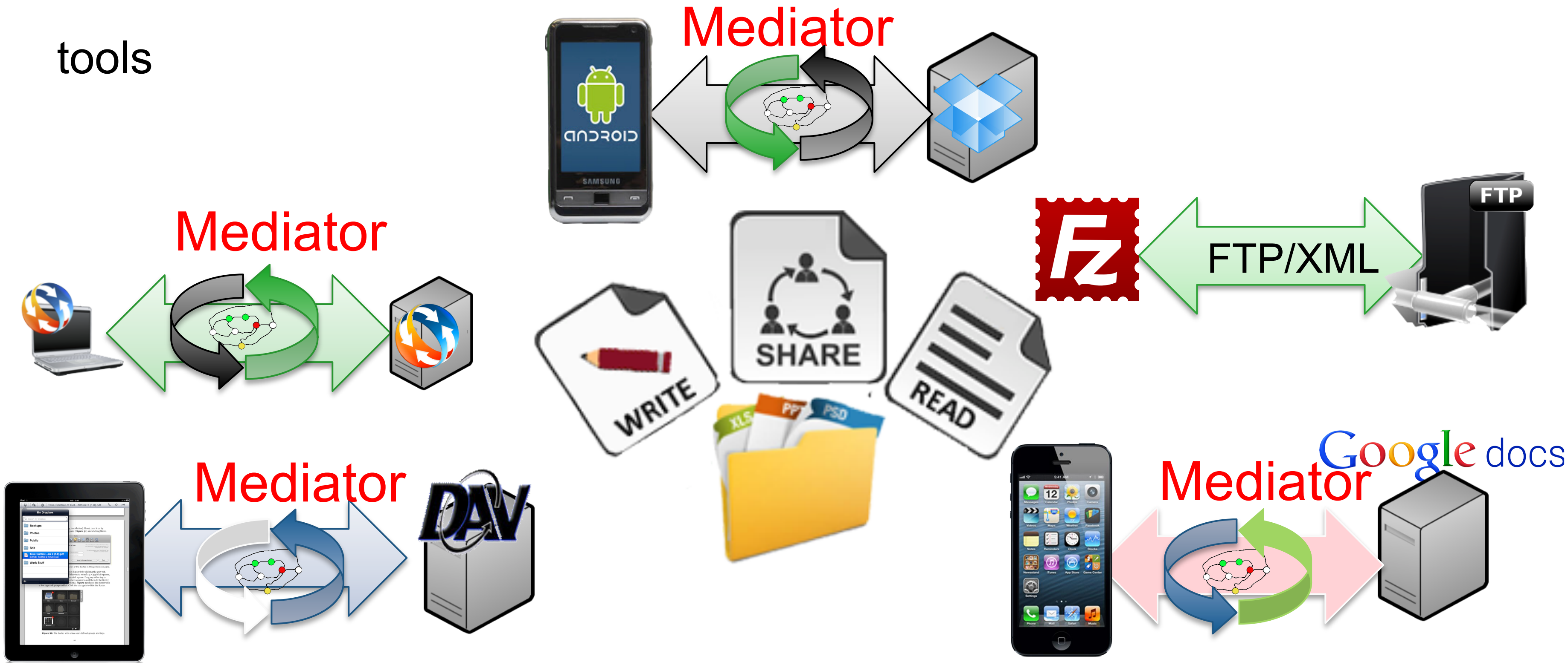


On-the-fly interoperability **through emergent middleware**

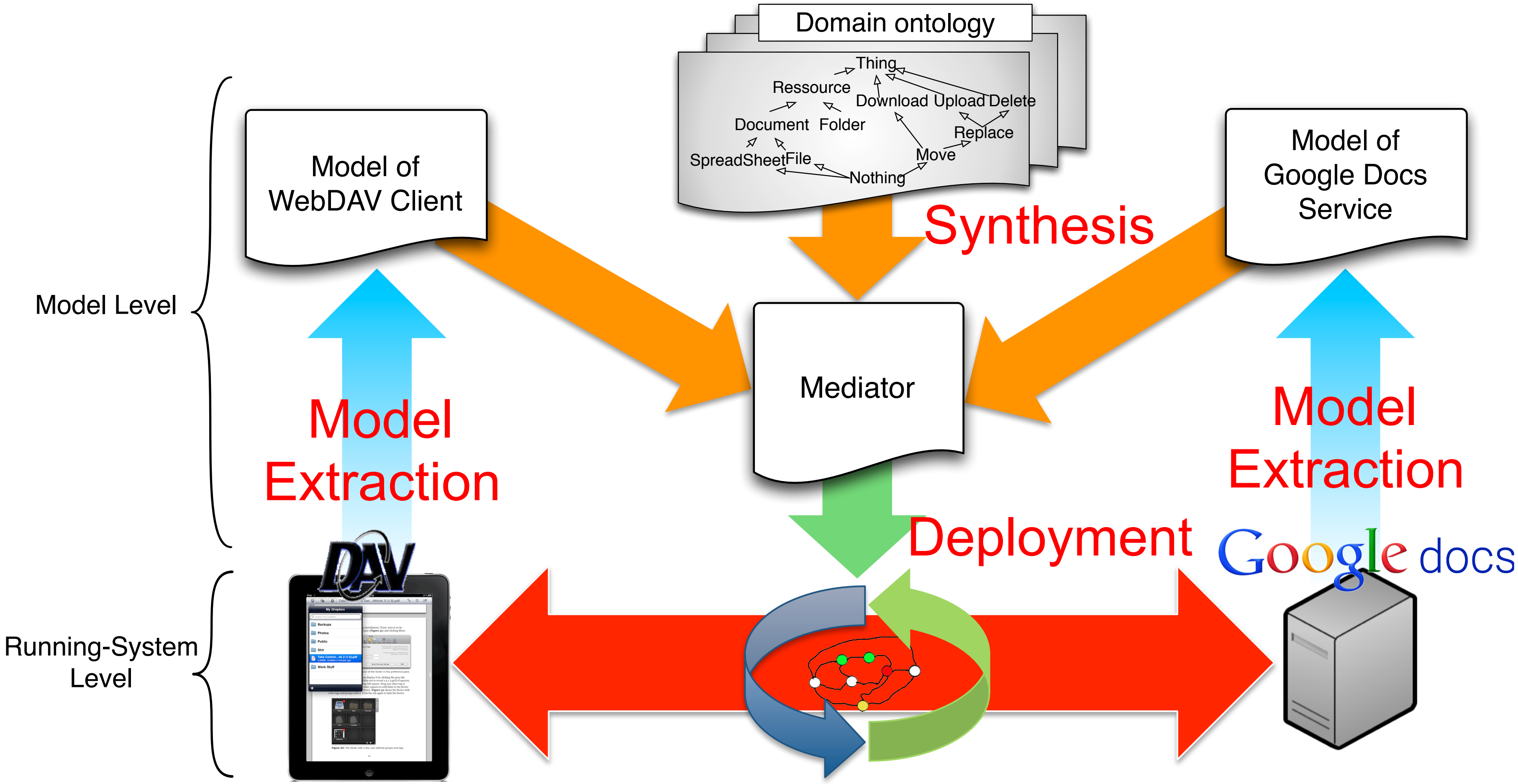
Babel fish

Achieving On-the-fly Interoperability

- Can we **observe**, **learn**, **synthesize** and **deploy** a binding dynamically
- **Emergent middleware** leveraging software engineering methods and tools

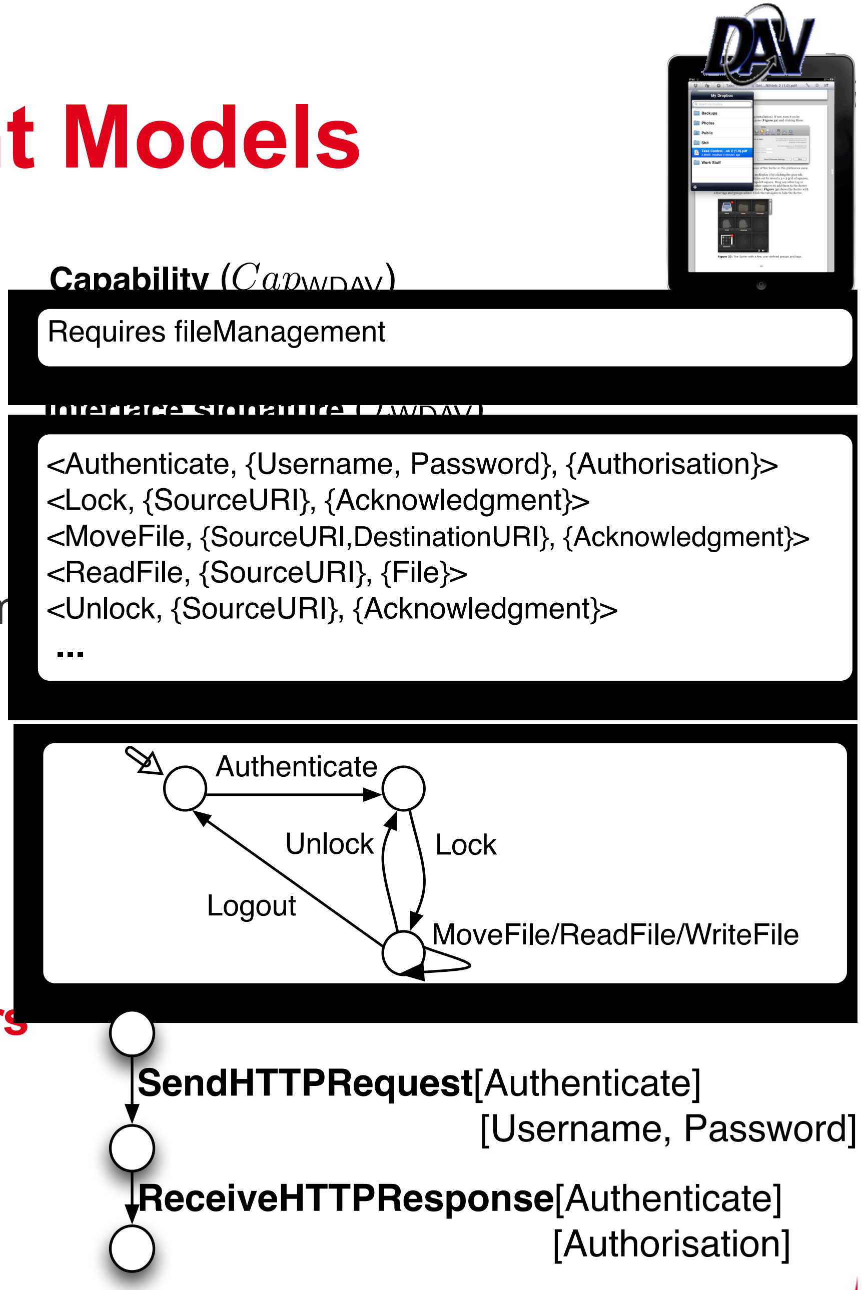


Model-based Emergent Middleware



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



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

Application layer

Interface Matching

→ many-to-many

Components' interfaces

Generating Correct-by-Construction Mediators

→ under ambiguity

Components' behaviours

Middleware layer

Concretisation

→ different interaction paradigms

Components' communication protocols

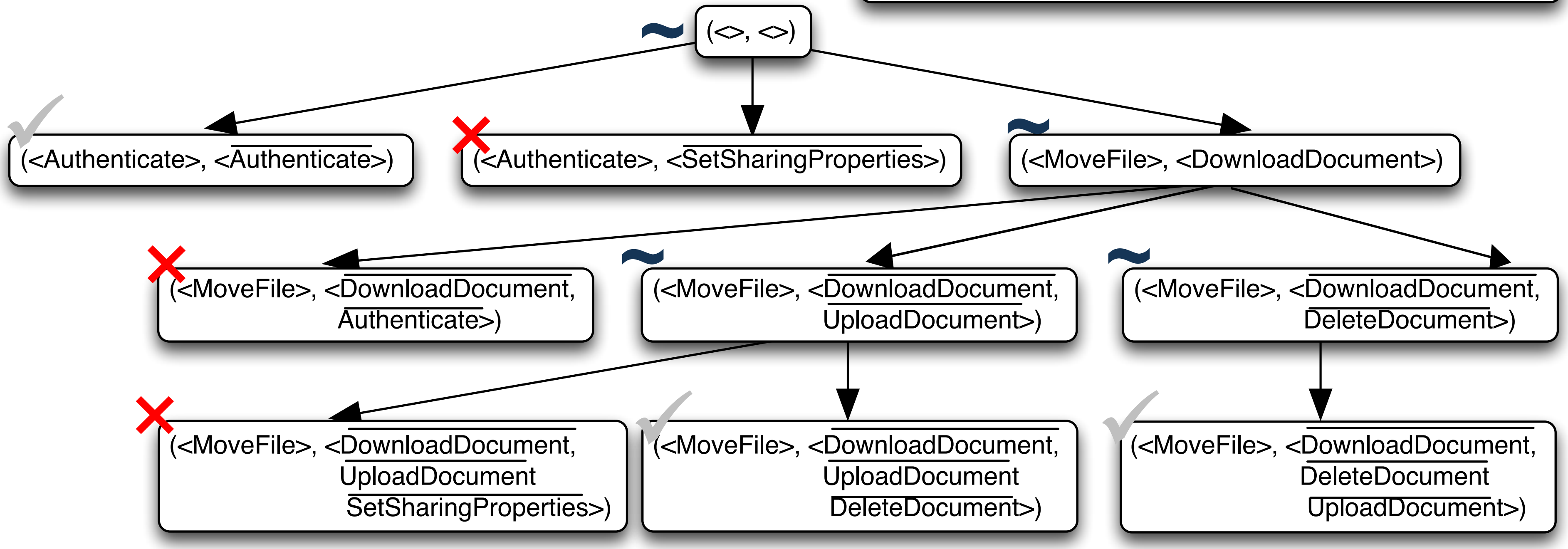
Interface Matching: An Example

DAV

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

Google docs

<Authenticate, {Username, Password}, {Authorisation}>
<SetSharingProperties, {SourceURI, SharingProperties}, {Acknowledgment}>
<UploadDocument, {Metadata, Content, DestinationURI}, {Acknowledgment}>
<DownloadDocument, {SourceURI}, {Document}>
<DeleteDocument, {SourceURI}, {Acknowledgment}>
...



Interface Matching: Computation

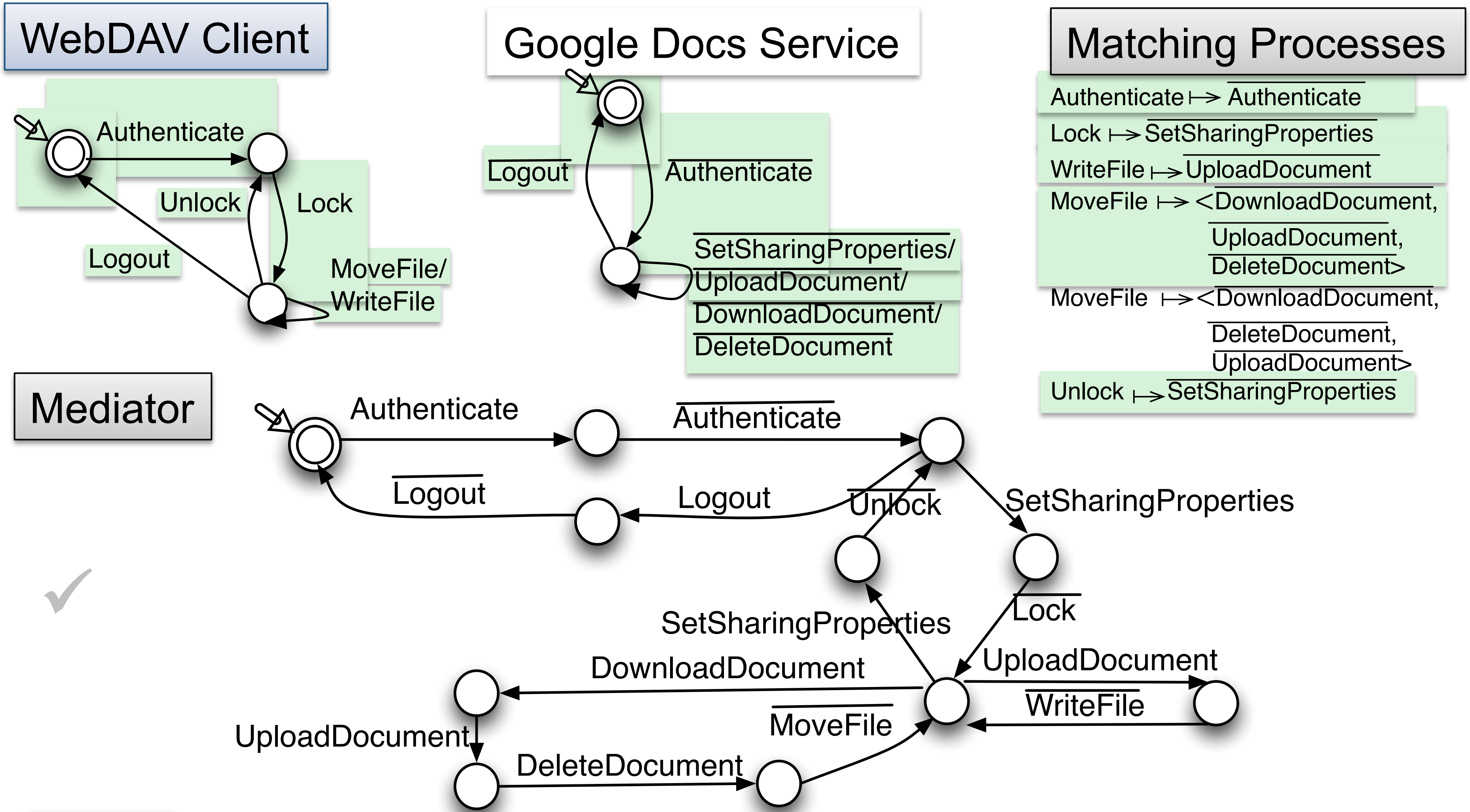
Matching interface $\mathcal{I}_1$ *to* interface $\mathcal{I}_2$ consists in finding all pairs of actions such that a sequence of actions required by the former can be safely performed using a sequence of actions provided by the latter. In addition, all pairs are minimal.

$$
\begin{aligned}
& Match\,(\mathcal{I}_1, \mathcal{I}_2) = \\
& \{\ (X_1, X_2)| \\
& \quad X_1 = \langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle\ , \alpha_{i=1..m} \in \mathcal{I}_1 \\
& \wedge X_2 = \left\langle \overline{\beta_1}, \overline{\beta_2}, \ldots, \overline{\beta_n} \right\rangle, \overline{\beta_{j=1..n}} \in \mathcal{I}_2 \\
& \wedge X_1 \mapsto X_2 \\
& \wedge \nexists\,(X_1', X_2')\,|\ \ X_1' = \langle \alpha_1, \alpha_2, \ldots, \alpha_{m'} \rangle,\ \alpha_{i=1..m'} \in \mathcal{I}_1 \\
& \qquad\qquad\qquad \wedge X_2' = \left\langle \overline{\beta_1}, \overline{\beta_2}, \ldots, \overline{\beta_{n'}} \right\rangle, \overline{\beta_{j=1..n'}} \in \mathcal{I}_2 \\
& \qquad\qquad\qquad \wedge (X_1' \mapsto X_2') \\
& \qquad\qquad\qquad \wedge\ (m' < m) \wedge (n' < n) \\
& \}
\end{aligned}
$$

But… NP-Complete

- Use Constraint programming with adequate ontology encoding

Mediator Synthesis: An Example



✓

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 $\quad \text{END} \leftrightarrow_{\text{END}} \text{END}$

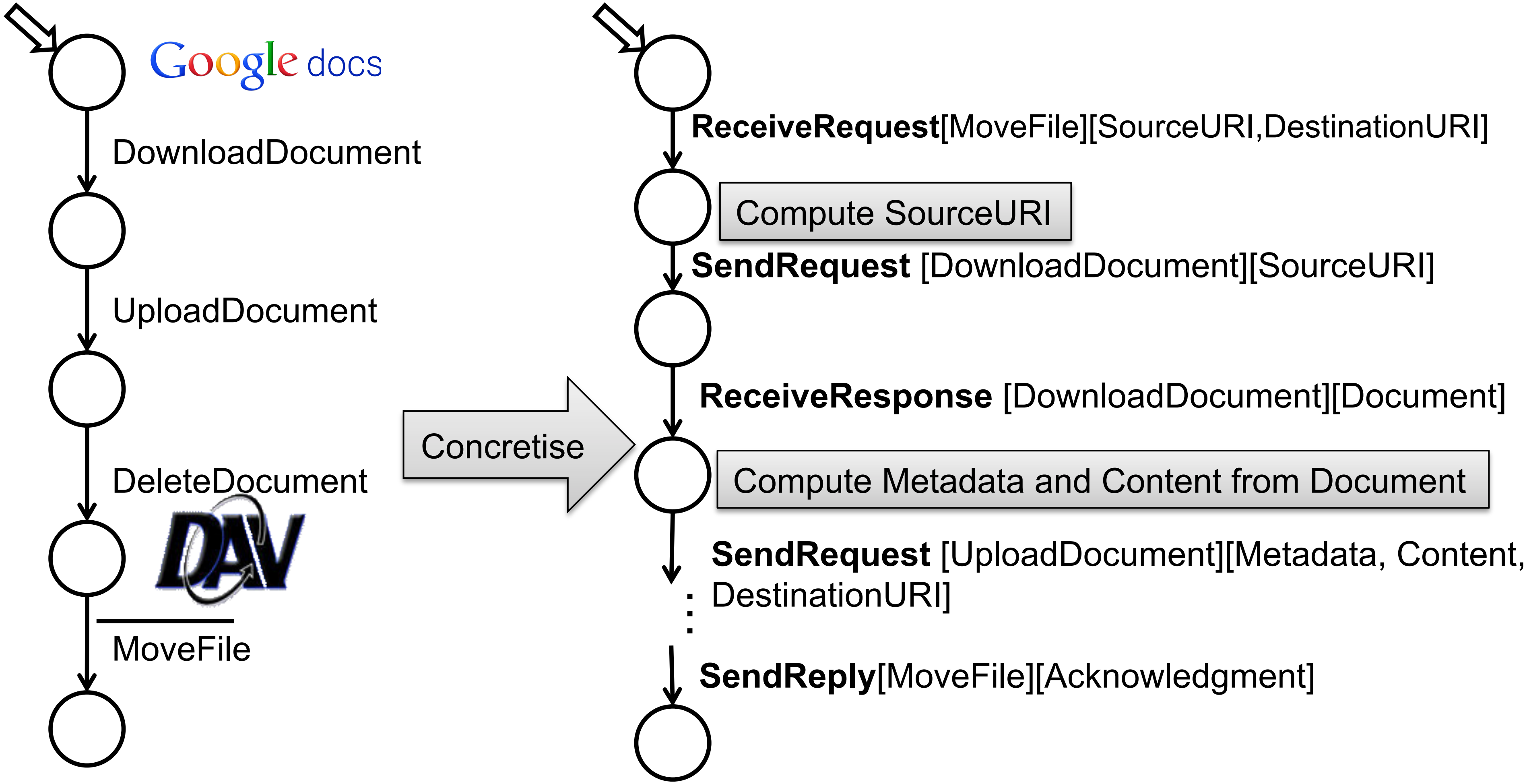
$$\begin{array}{ll}
\text{if} & P_1 \overset{X_1}{\Rightarrow} P_1' \text{ and } \exists\,(X_1, X_2) \in Match\,(\mathcal{I}_1, \mathcal{I}_2) \\
\text{such that} & P_2 \overset{X_2}{\Rightarrow} P_2' \text{ and } P_1' \leftrightarrow_{M'} P_2' \\
\text{then} & P_1 \leftrightarrow_M P_2 \text{ where } M = M_{m-n}(X_1, X_2); M'
\end{array}$$

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

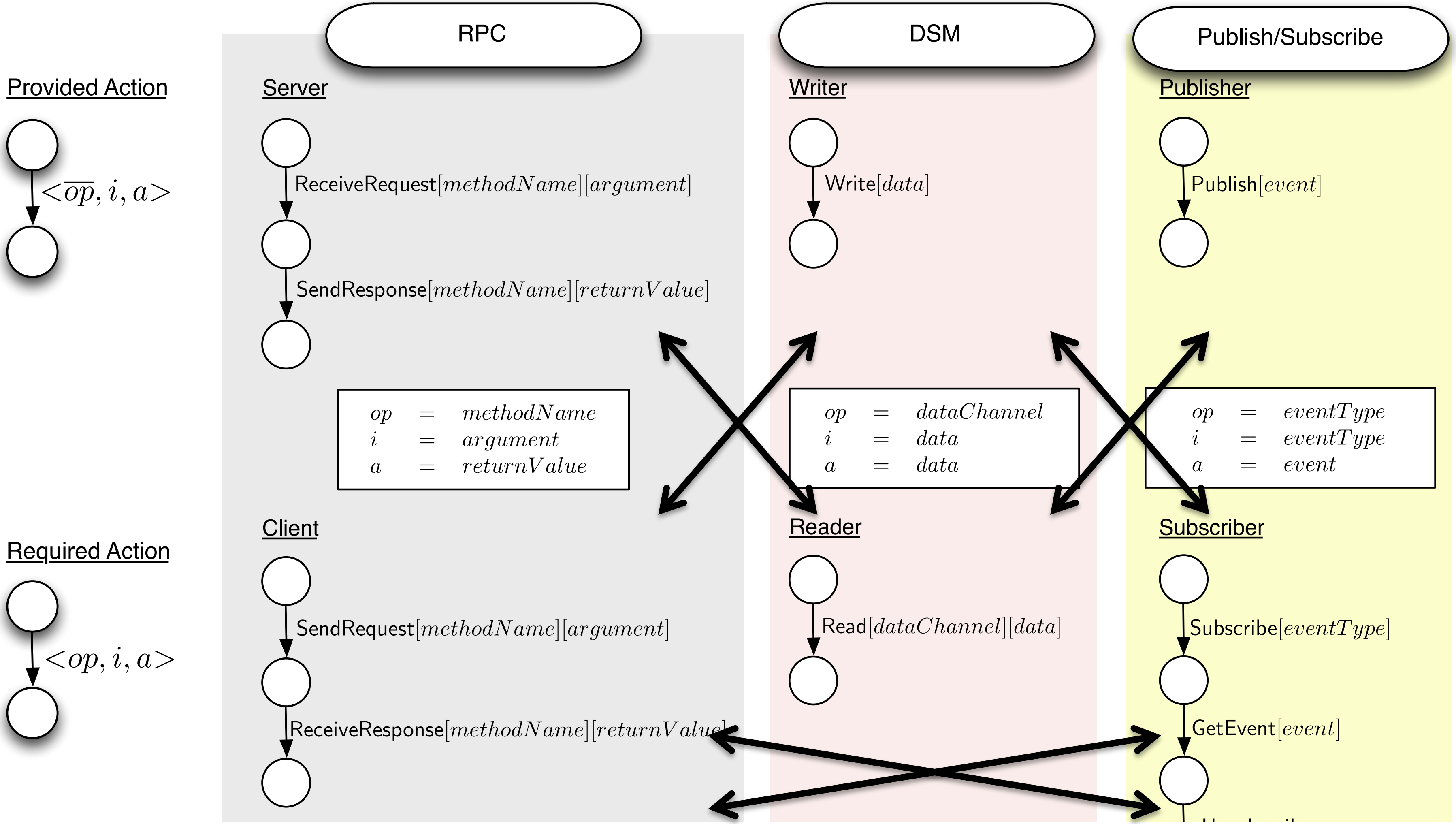
Coordinating middleware Protocols: An Example

Interface Matching
Generating Mediators
Concretisation

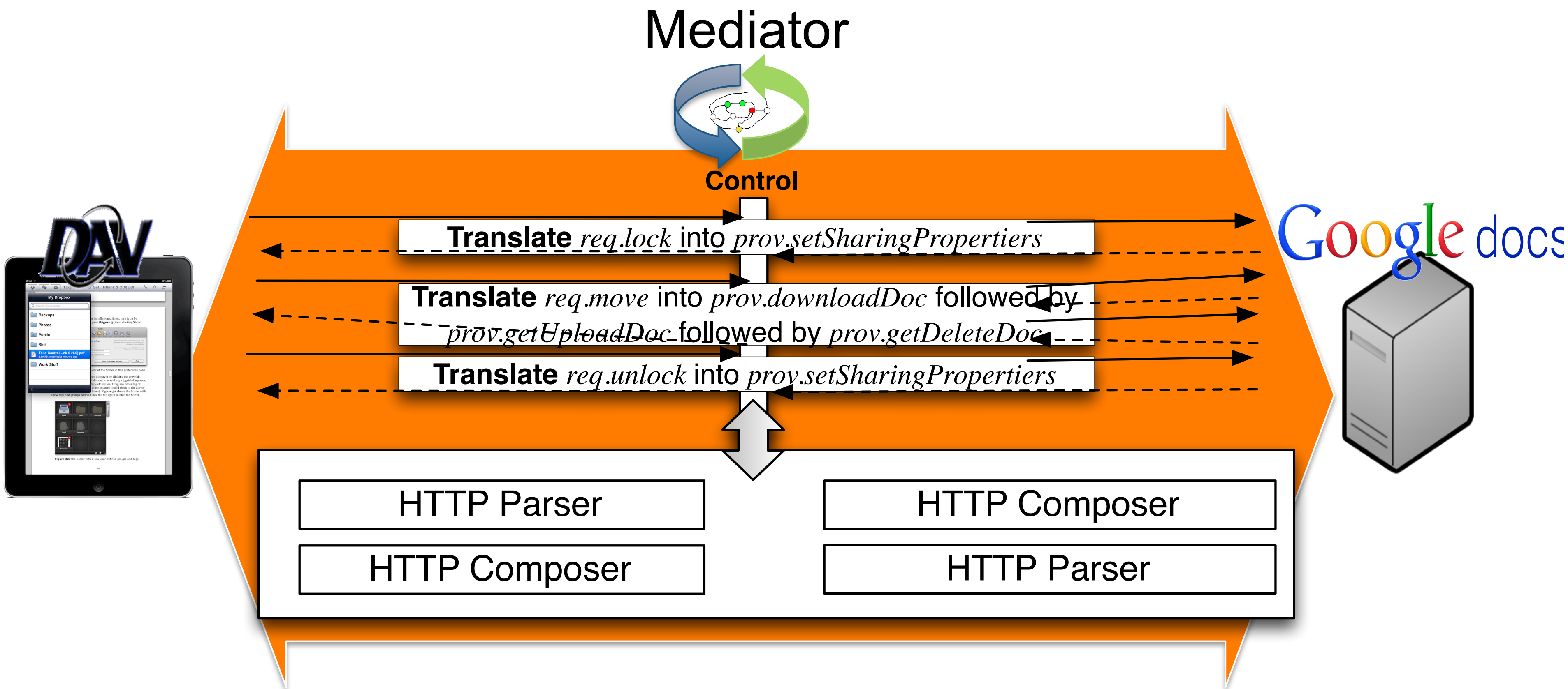


Interoperability across interaction paradigms

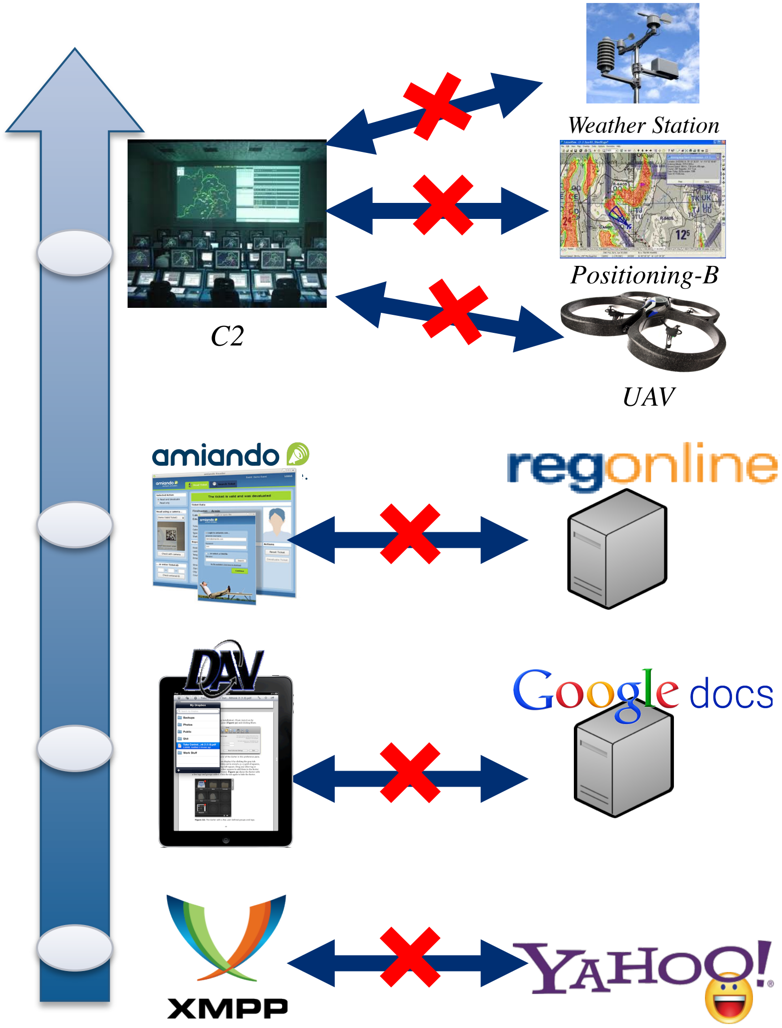
Interface Matching
Generating Mediators
Concretisation



Deploying the Mediator



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