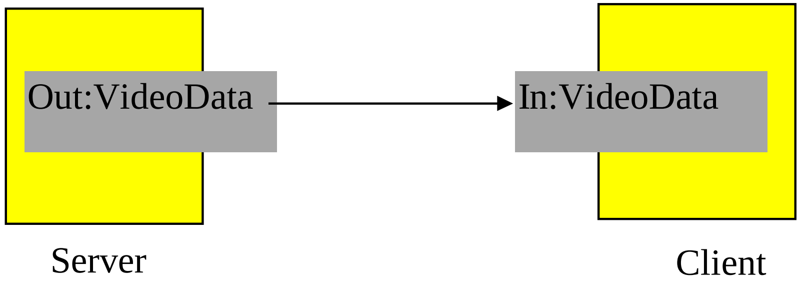
Configuration Semantics for FlexiNet Applications

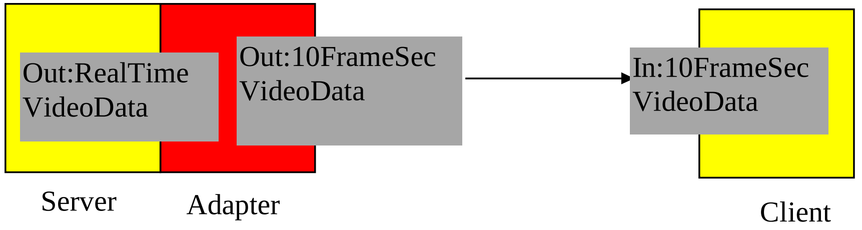
Will Harwood

The problem

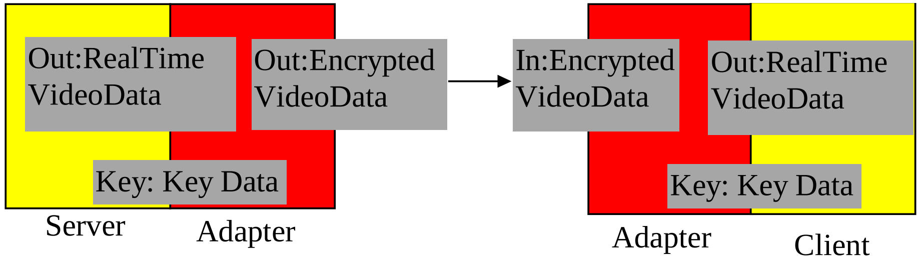
- Need to agree the protocol by which clients and servers talk to one another



Adapters

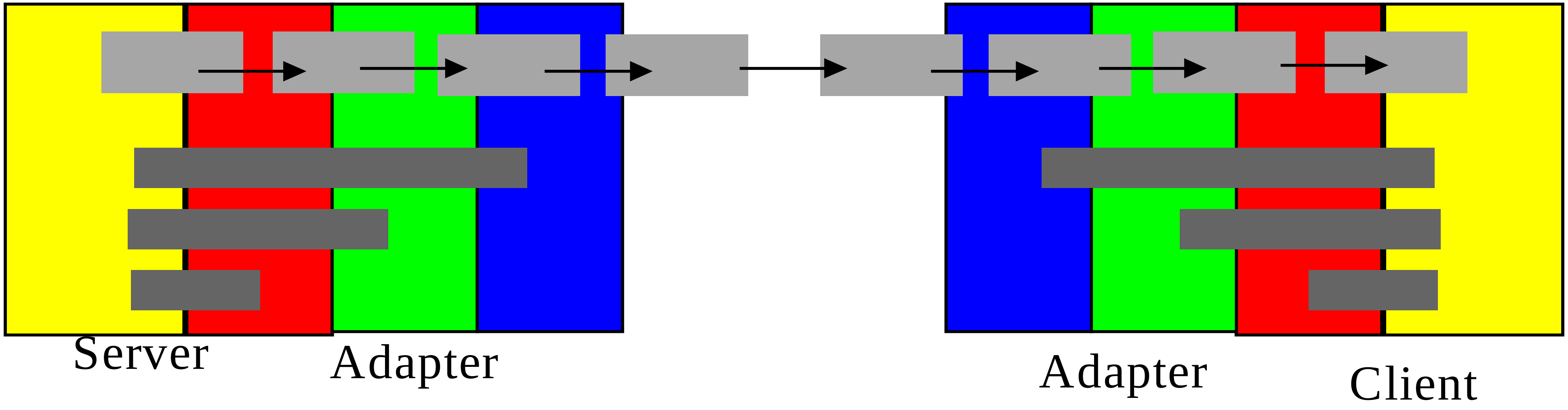


- But there may be mismatches between client and server and so we need to interpose an adapter
- The choice of which adapter and where it is placed depends on costs to client and server



- Adapters may require additional information
- Adapters may have external constraints e.g. some adapters may be illegal in some circumstances

Stacking Adapters Together



A;B;C

Goal

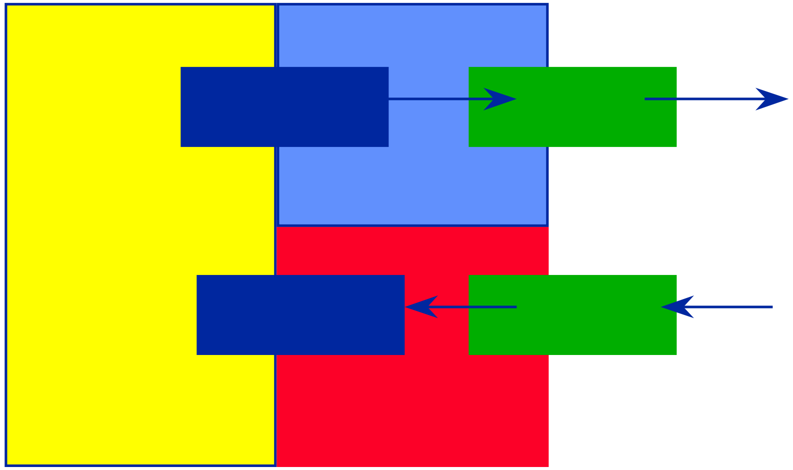
- Wish to find a sequence of adapters on each side of a client server interaction that satisfies the requirements of both client and server
- Which means we need to be able to specify the behaviour of adapters and the client and server requirements for an interaction

What do you need to specify?

- The transformation an adapter carries out
- The context in which an adapter may be used
  - local context
  - global context
- The cost of the deployment of an adapter
  - Static Cost
  - Variable Cost

Transformations: Simple Case

- An adapter is a pair of functions that transforms data types



Transformation: pair of function types

Functions don’t work

- Problem
  - Adapters may have state
  - Adapters may have more than one input and output
  - Adapters may not even be “functional”
- Solution
  - Use a notion of typed processes and describe adapter transformations as process types

Process Types

- Choice of granularity
- Processes have typed input and output channels
  - name ! type - $\alpha\ !\tau$ An output channel
  - name ? type - $\alpha\ ?\tau$ An input channel
- A process type is the signature of its input and output channels
  - {channel ,…,channel} - { $\alpha\ !\tau$, $\beta\ ?\sigma$ } put:A signature
- A process supports interfaces
  - name: signature - default : { $\alpha\ !\tau$, $\beta\ ?\sigma$ } An interface

Adapter Types

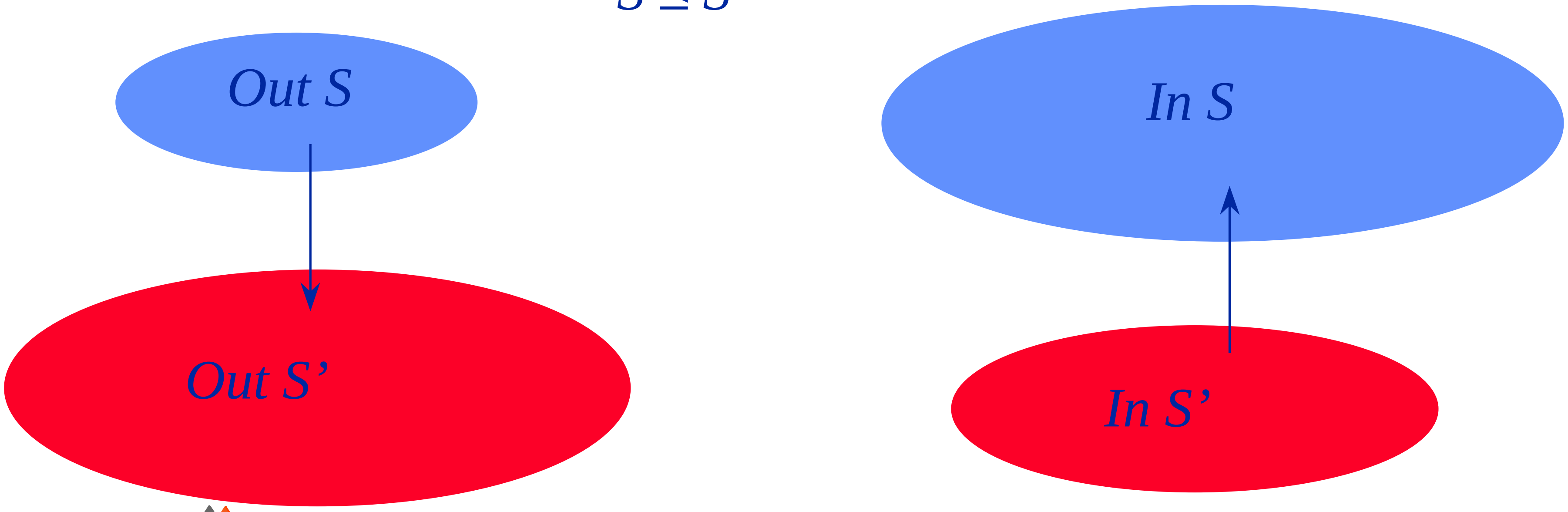
- Adapters have an “input” side and an “output” side
  - sig ↔ sig : set of interfaces
  - A adapter fits onto and interface and changes its type
- To simplify adapter descriptions the input signature is interpreted in a complementary fashion
  - i.e. { α !τ, β ?σ } as an input signature means { α ?τ, β !σ } as a process signature
- Permit conformant matching of signatures

SubTypes of signatures

$$Out(S) = Out(S') \wedge In(S) = In(S')$$

$$(\forall x \in Out(S).type(x) \subseteq type(x')) \wedge \forall x \in In(S).type(x) \supseteq type(x')$$

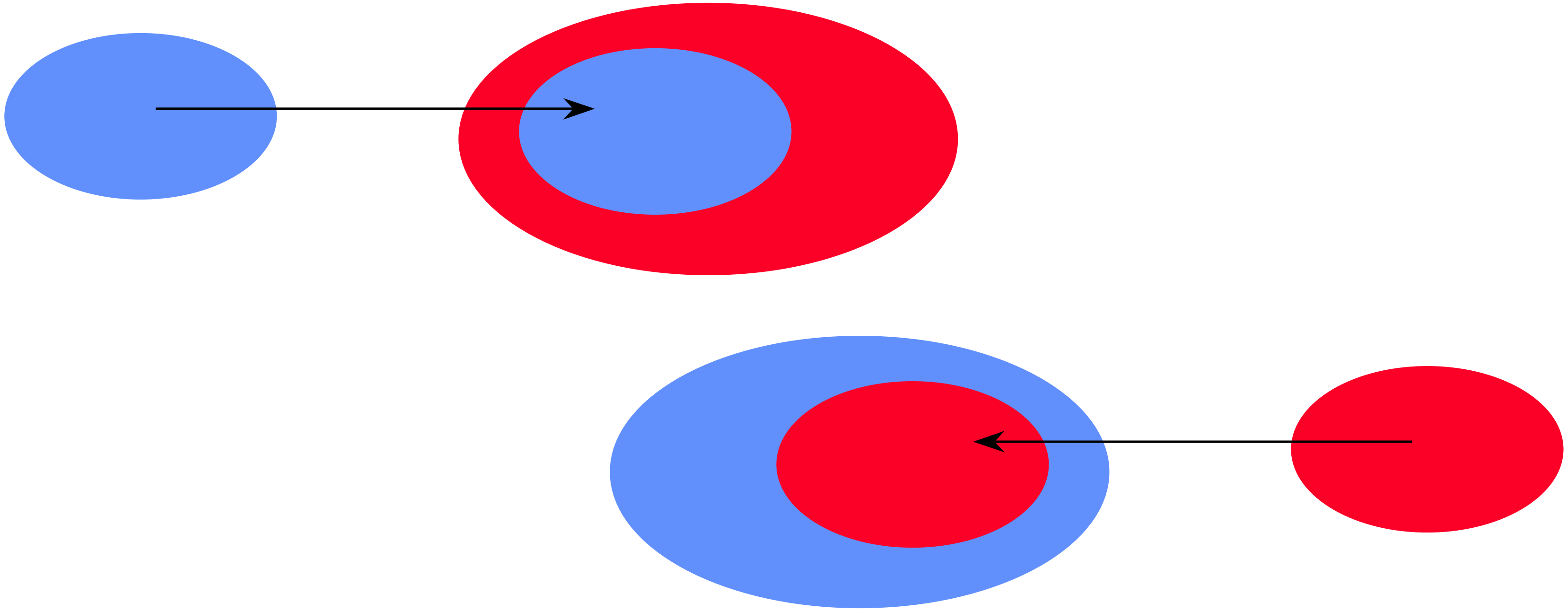
$S \leq S'$



Type Conformance of Signatures

$$Out(S) = In(S') \wedge In(S) = Out(S')$$

$$(\forall x \in Out(S).type(x) \subseteq type(x'*)) \wedge \forall x \in In(S).type(x'*) \subseteq type(x)$$



Type Rules

$$\frac{\mathbf{A} \triangleright A \leftrightarrow B{:}U \qquad \mathbf{B} \triangleright B' \leftrightarrow C{:}V}{\mathbf{A}\,;\mathbf{B} \triangleright A \leftrightarrow C{:}U \cup V}$$

where B and B’ are conformant

Allow Subtype Polymorphism

$$\prod_{\alpha \prec T} A(\alpha) \leftrightarrow B(\alpha) : U(\alpha)$$

$$\prod_{\alpha \prec ByteStream} data!\alpha \leftrightarrow data!Encrypt(\alpha) : Key?keyType$$

Contexts

- Global Context - Basic propositions about the use of adapters
  - "Legal in France"
  - "ANSA sponsors only"
  - For simplicity need some default reasoning
- Local Context - Facts about what adapters have already been used in a chain
  - "compressed"

Global Context Rules

$\mathbf{A} \triangleright_{\sigma} \varphi$ *means that **A** satisfies φ under assumptions σ*

- σ is a consistent set of assumptions of the form (**X**, φ, **true**) or (**X**, φ, **false**)
- either is true for the adapter **A** or (**X**, φ, **true**) ∈ σ

$$\frac{\mathbf{A} \triangleright_{\sigma} \varphi \qquad \mathbf{B} \triangleright_{\sigma'} \varphi}{\mathbf{A;B} \triangleright_{\sigma \cup \sigma'} \varphi}$$

- provided σ ∪ σ′ is consistent

Local Context Constraints

- $\mathbf{A} \triangleright (\alpha_A, \beta_A)$ means that α is the complete set of **A’**s requirements to the left and β is a complete set of atomic historical propositions that **A** obeys.

$$\frac{\mathbf{A} \triangleright (\alpha_A, \beta_A) \quad \mathbf{B} \triangleright (\alpha_B, \beta_B) \quad \beta_A \text{‘}_{\alpha \in \alpha_B} \alpha}{\mathbf{A;B} \triangleright (\alpha_A, \beta_A \cup \beta_B)}$$

- where $\beta_A \text{‘}_{\alpha \in \alpha_B} \alpha$ means that β_A entails every α in the set α_B by the rules of propositional logic.

Costs

- Cost is a vector e.g. cpu cost, memory cost, bandwidth
- Static Costs
  - fixed cost vector of an adapter
- Variable Cost
  - function from the number of calls on an adapter to cost vector
  - plus a scaling function from number of input calls to number of out calls
  - [f,g]

Static Cost Rule

$\mathbf{A} \rhd \mathbf{x}$ *means that **A** satisfies cost vector x*

$$\frac{\mathbf{A} \rhd \mathbf{x} \qquad \mathbf{B} \rhd \mathbf{y}}{\mathbf{A} ; \mathbf{B} \rhd \mathbf{x} + \mathbf{y}}$$

Variable Costs Rule

$\mathbf{A} \triangleright [f, g]$ *means that **A** satisfies the cost function f and call multiplier g*

$$\frac{\mathbf{A} \triangleright [f_A, g_A] \quad \mathbf{B} \triangleright [f_B, g_B]}{\mathbf{A};\mathbf{B} \triangleright [f_B \circ f_A, g_A \hat{+} g_B \circ f_A]}$$

where $\quad a \hat{+} b = \lambda \; x.a(x) + b(x)$

Negotiation goals

- A requirement = $(n,a) \Rightarrow (\tau,\mathbf{v},s)$
  - n = expected number of calls
  - a = set of assumptions already satisfied
  - = adapter type required
  - **v** = maximum cost vector (for simplicity)
  - s = set of global context constraints
- The goal is to find an adapter chain **A** that satisfies the requirement

Satisfying a Goal

- **i.e. find an A** with
  - with type τ' where is a τ' subtype of τ.
  - with cost $c + (f\ n) \leq \mathbf{v}$ where **A** satisfies static cost **c** and variable cost $[f, g]$
  - where **A** satisfies the set of propositions s
  - where **A** satisfies (a, w) where w is any consistent set of atomic adapter propositions

The Goal Rule

$$\frac{\begin{array}{c} \qquad\qquad\qquad \mathbf{A} \triangleright_{\mathrm{cost}} \mathbf{c} \\ \tau' \leq \tau \qquad\qquad \mathbf{A} \triangleright_{\mathrm{variable}} [f, g] \\ \mathbf{A} \triangleright_{type}\ \tau' \quad \mathbf{c} + f(n) + b(m) \leq \mathbf{v} \quad \mathbf{A} \triangleright_{global} \wedge s \quad \mathbf{A} \triangleright_{local} (a, w) \end{array}}{\mathbf{A} \triangleright (n, a) \Rightarrow (\tau, \mathbf{v}, s)}$$

Where to go Next

Approaches to Negotiation

- Three situations:
  - a small number of sensible adapters for any given interaction
    - SSL handshake protocol approach - small number of interaction styles, simple parameterisation of requests
  - many possible combinations of basic adapters that can be used to achieve an overall adapter type but there is a simple global notion of "best" for all participants
    - Precompute solutions and treat as parameterised
  - a large number of possible adapters and no simple global notion of "best"
    - compute solutions on the fly

Need Examples

- Rule based semantics gives us a framework to e.g. build rule based negotiator
- but need to build a body of protocol examples to explore which alternatives to support
  - Already added mobility to FlexiNet
  - Implement more protocols on Flexinet
    - IIOP
    - TCP with SSL
  - Seek other examples