

# **Make the Grid Pay** **(Economic Web Services)**

*Jeremy Cohen, William Lee,*  
*Anthony Mayer, Steven Newhouse*  
London e-Science Centre

What makes the grid different from:

- Distributed Computing?
  - Because it's high performance?
- Shared Computer Resources?
  - Because it's wide area?

*shared computing resources*

*across administrative domains:*

*public vs. private shared computing*

- Sharing compute resources where there is no pre-existing relationship:
  - Trust to a common greater authority
  - “from each service according to its capabilities, to each client according to its requirements”?
  - ...or we *pay*
  - Given the real world cost of compute resources...

# What's missing from the current state of the art?

“Private” Grid Applications  
Planned / Existing Relationship

“Public” Grid Applications  
Spontaneous / Real time

Higher Level  
Services

Data APIs  
Semantics  
Job Description

Economic  
Services

Contracts  
Pricing  
Monetary  
Transactions

Service  
Infrastructure

Publish-Subscribe Invocation Discovery Security  
OGSI Web Services (WS-I, WS-RF) Jini JXTA

- UK Core e-Science Programme project
- Interfaces & protocols to trade Grid Services
- Funded by Department of Trade & Industry
- Collaborators
  - London e-Science Centre
  - e-Science North West
  - Southampton e-Science Centre
  - UK Grid Support Centre
  - Astrophysics at LJM



## GESA

*Chargeable  
Grid  
Service*

*Grid  
Payment  
Service*

*Resource Usage  
Service*

## WS Agreement

Economic  
Web  
Services

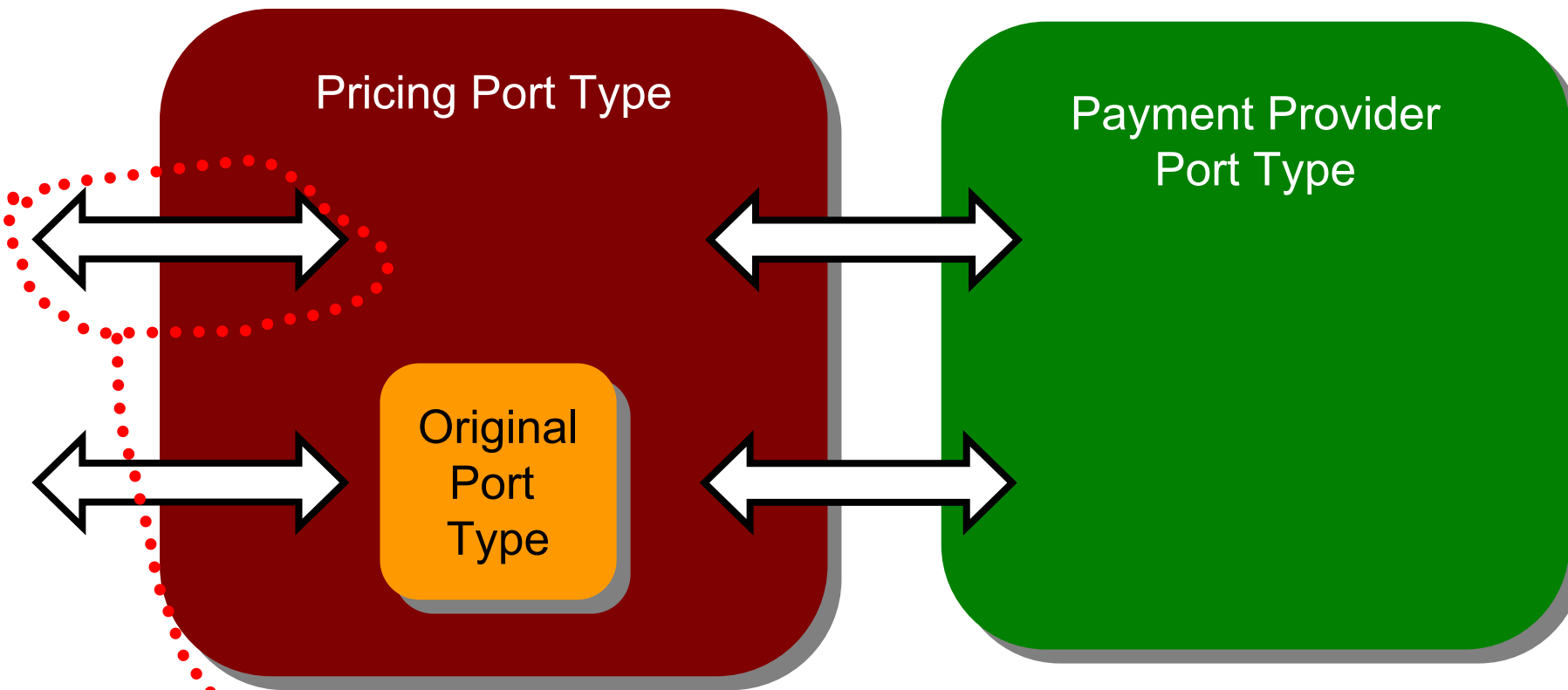
- Introduction    Infrastructure / Semantic Grid
- Negotiation (Agreeing a Price)    WS
- Settling a Contract    Agreements
  - Ends negotiation process with a *monetary commitment*
- Executing a trade
  - Invocation of service
  - Monetary transaction

Already present  
In underlying!

# Making a Service Chargeable: Decorator Pattern

*Chargeable Web Service*

*Payment Provider Service*

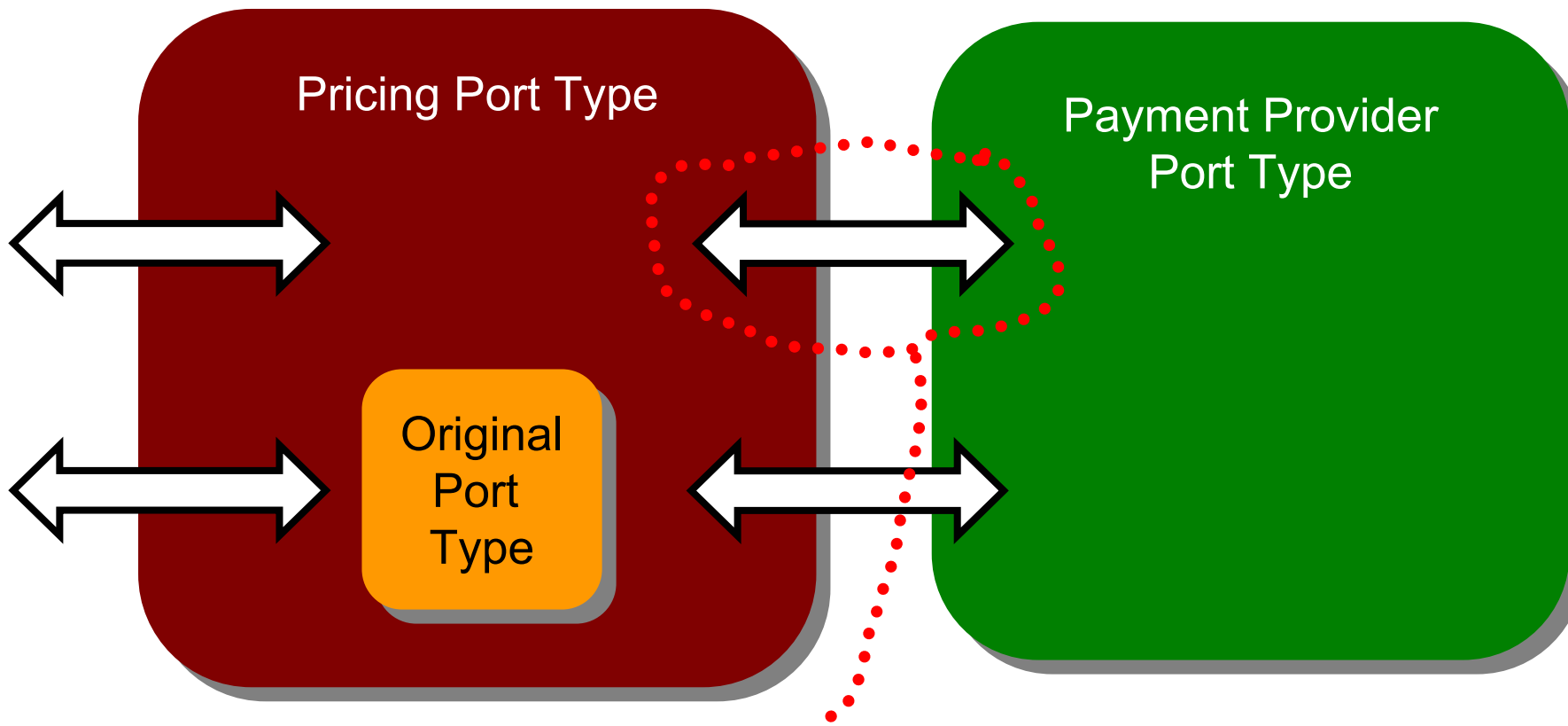


*... Negotiation and Agreement...*

# Making a Service Chargeable: Decorator Pattern

*Chargeable Web Service*

*Payment Provider Service*

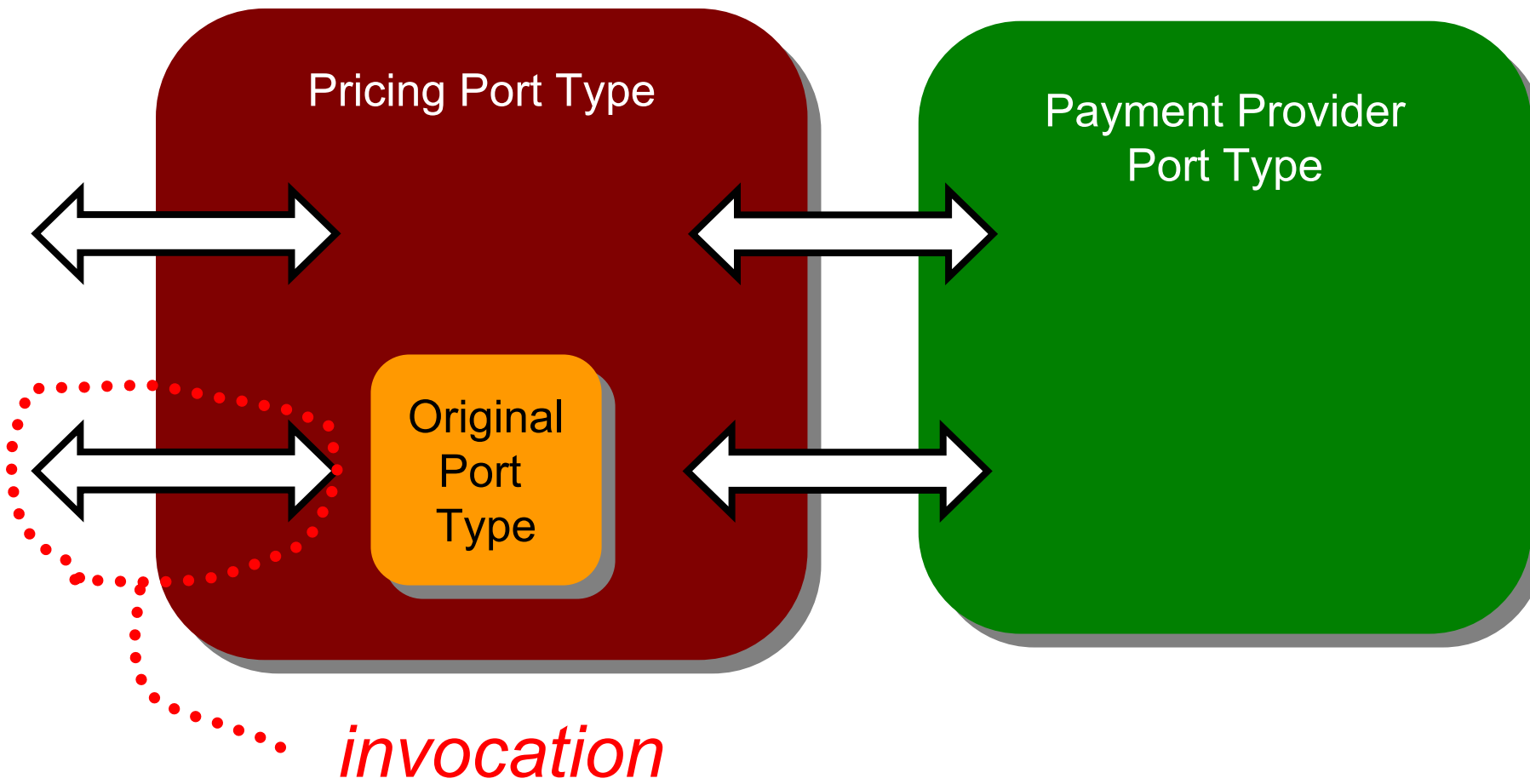


*...monetary commitment*

# Making a Service Chargeable: Decorator Pattern

*Chargeable Web Service*

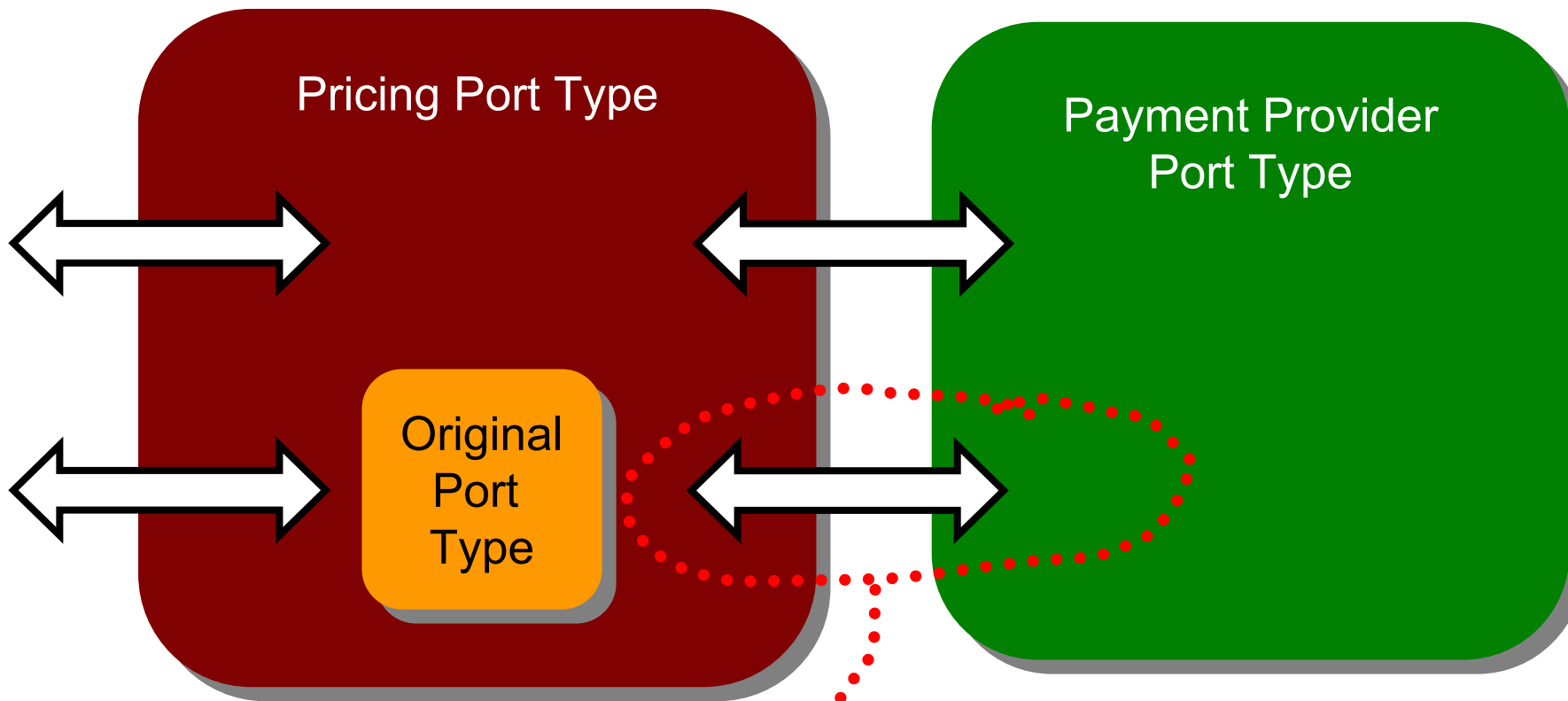
*Payment Provider Service*



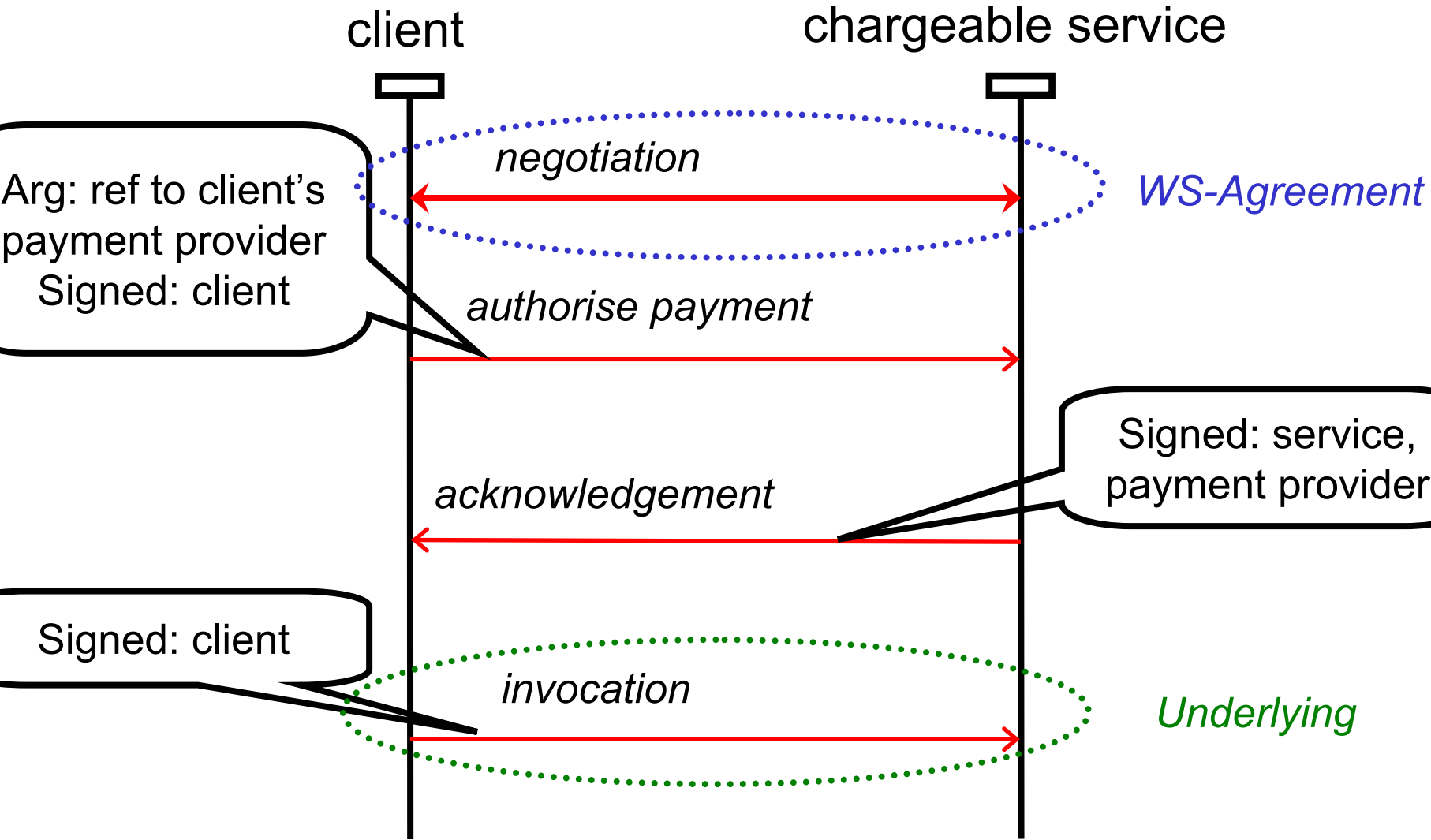
# Making a Service Chargeable: Decorator Pattern

*Chargeable Web Service*

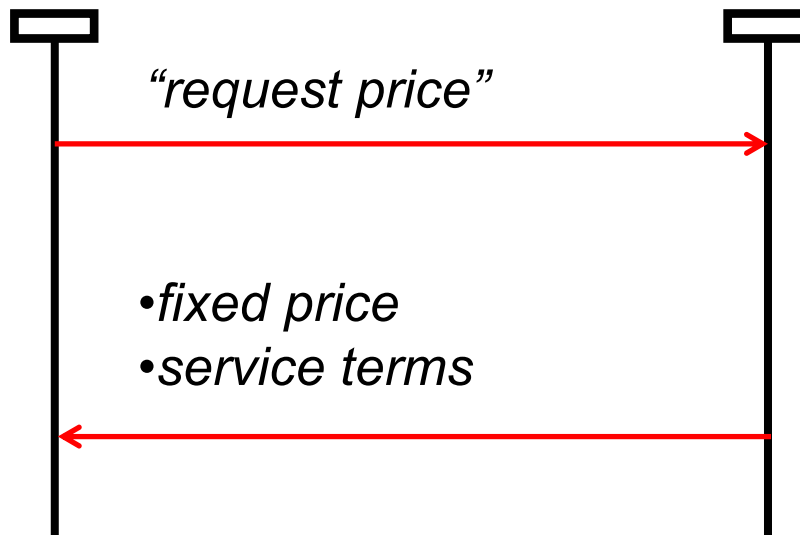
*Payment Provider Service*



*reconciliation*



- Evolving standard – currently using a trivial subset of WS-A from GESA
- Fixed Price terms which can be accepted or rejected



Service terms include:

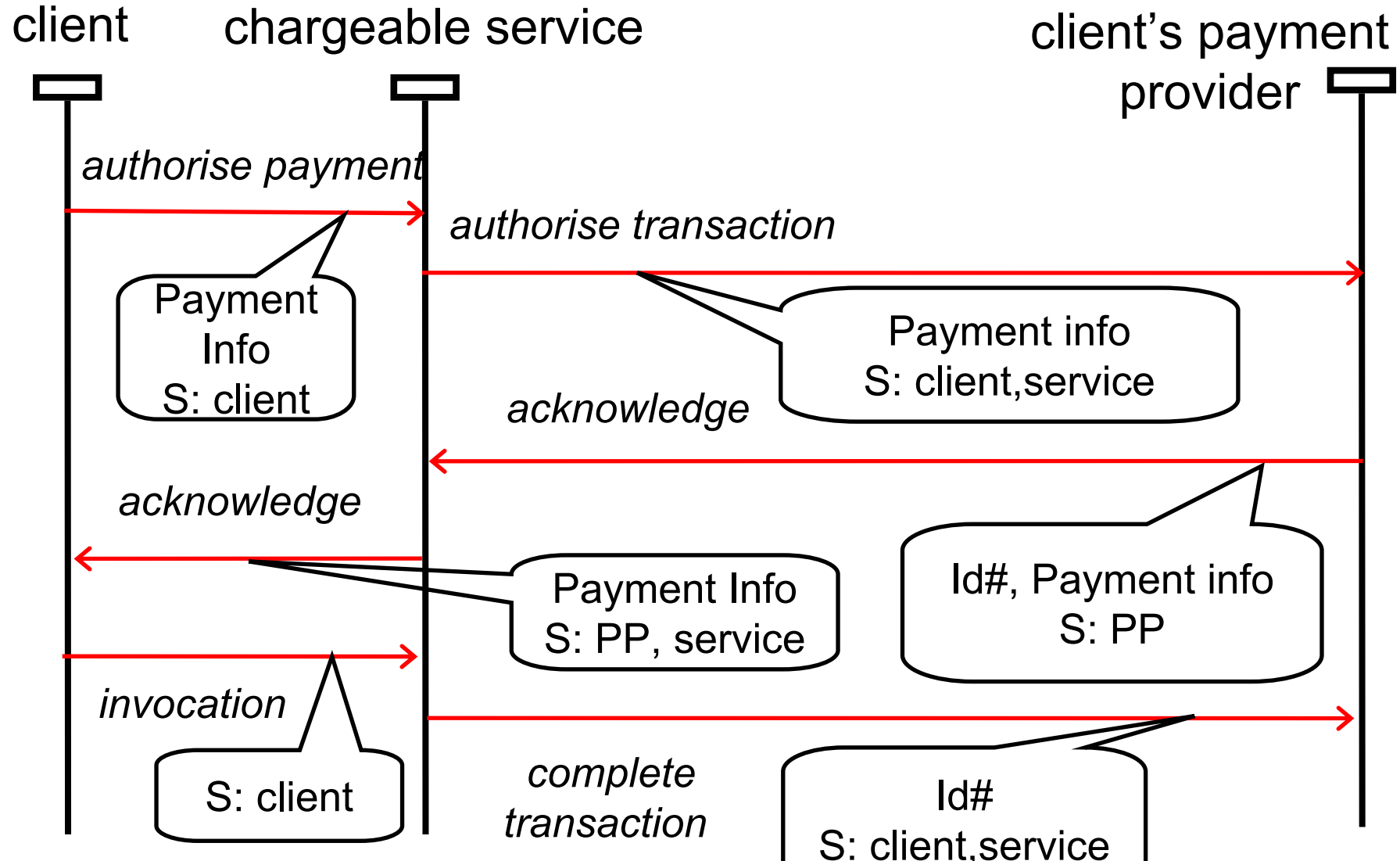
- Number of Transactions
- Per-Invocation Pricing

- Port Type (and implementation) are inherited from underlying
- Invocation of underlying is a sessional transaction
  - Need to tie the invocation into agreed contractual arrangements
  - Doesn't mean we break the stateless architectural requirements!
    - Session information embedded in the message – messaging completely defines the logic
    - WS-Context, for example

- `authorisePayment`
  - input: agreement document with additional terms:
    - `AccountId`
    - `Payment Provider EndPoint Address`
    - `JobID`
    - `maxTransactions`
    - `expiry`
    - `ammount`
    - ...
  - output: signed acknowledgement
  - fault:
    - *incorrect pricing*

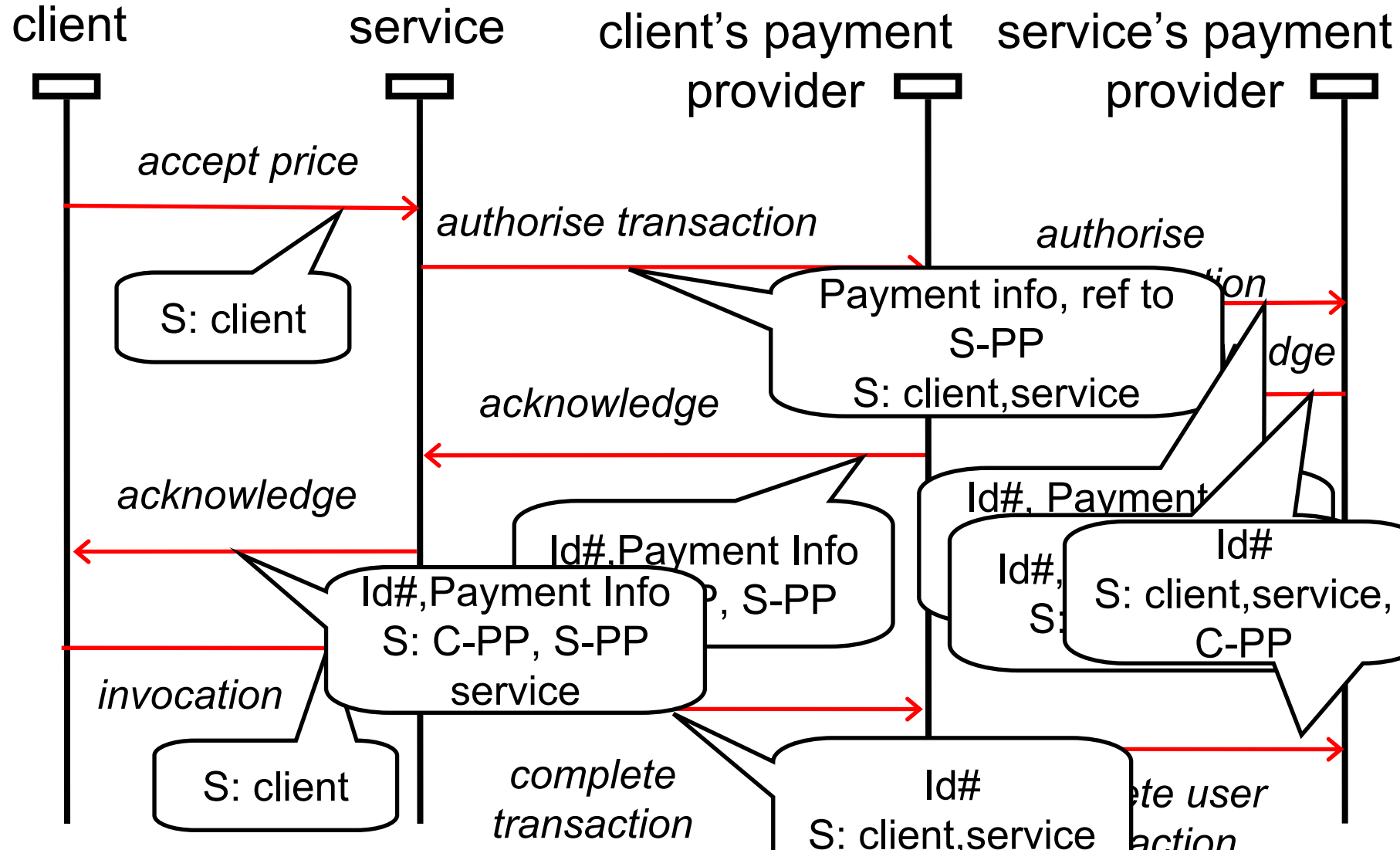
- `agreePrice`
  - input: WS-A proposed terms
  - output: WS-A response terms
  - faults: any WS-A fault
- `remainingCredit`
  - input: sessional context
  - output: document specifying credit
  - fault: *incorrect context*
- `remainingOperations`
  - input: WS-A terms specifying consumable resources
  - output: document specifying operations
  - fault: any WS-A fault

- Abstraction abstraction abstraction
  - Realisation with multiple payment systems
- Autonomous delegation
  - “Securitisation”
- Commodity security
- Resist replay attack



- Charging without permission
  - Invocation of payment requires client signed authorisation
- Replay
  - Charge can be made at most once; invocation includes id#
  - Still allows for micropayments-per-invocation

- Service may not have relationship with client's payment provider
- Peer-to-peer symmetry: service is also client (and has own payment provider)
- Reconciliation *between* payment providers



# Payment Provider Port Type Interface

London e-Science Centre

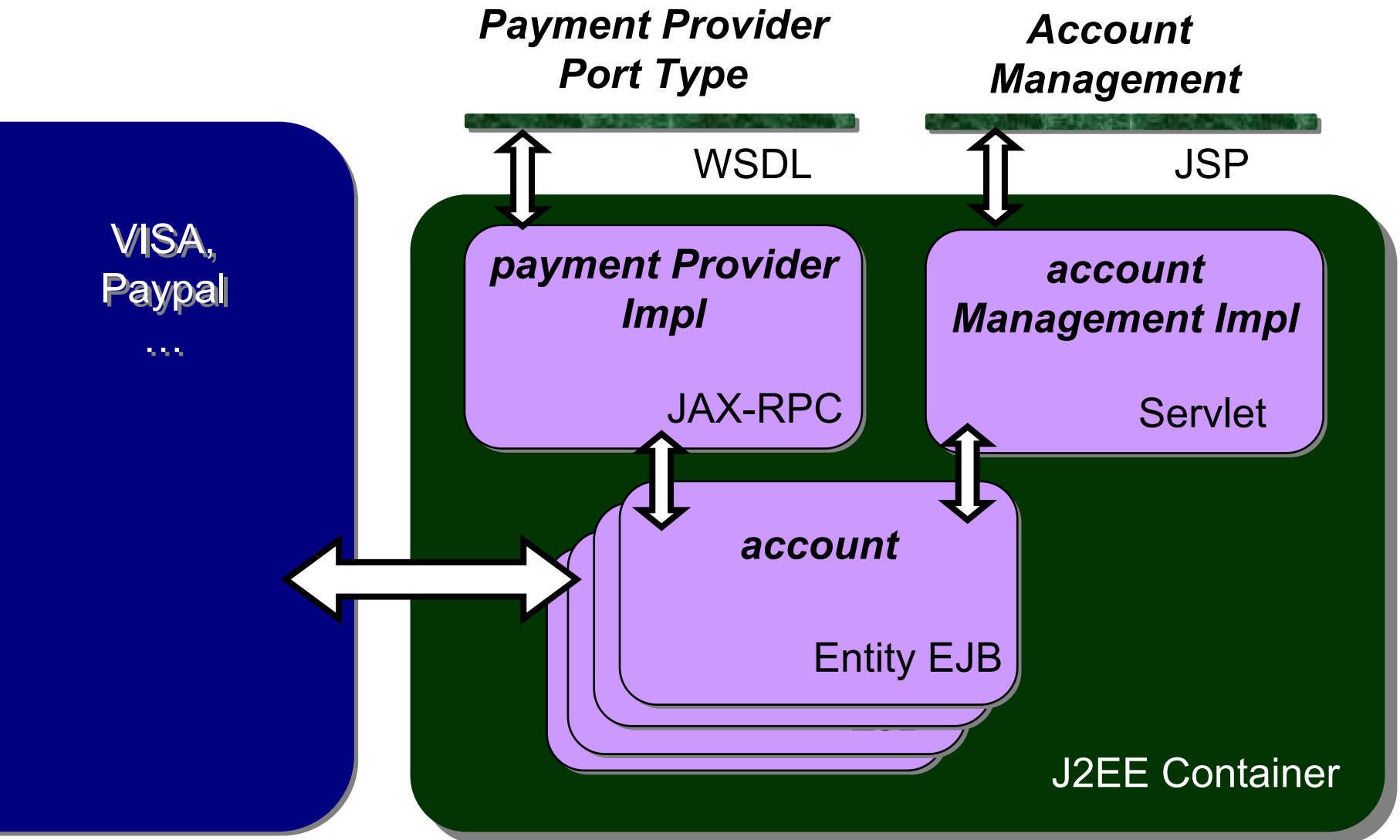
[www.lesc.imperial.ac.uk](http://www.lesc.imperial.ac.uk)

- `paymentSystem`
    - input: none
    - output: informational document
    - faults: none
  - `authoriseTransaction`
    - input:
      - `AccountId`
      - `ammount`
      - `maxTransactions`
      - `expiry`
    - output: signed acknowledgement with *transaction id#*
    - fault:
      - *FromAccountIdIdentifierDoesNotExist*
      - *ToAccountIdentitfierDoesNotExist*
      - *SignatureFailed*
      - *InsufficientFunds*
      -
- } Terms from pricing port's `authorisePayment` message

# Payment Provider Port Type Interface cont.

- `completeTransaction`
  - input: signed *transaction id#*
  - output: none
  - fault:
    - various *SignatureFailed* faults
    - *InsufficientFunds*
    - *TransactionAlreadyComplete*
    - *TransactionDoesNotExist*
    - *TransactionHasExpired*
    - ...
- `verifyUserTransaction` and other monitoring messages

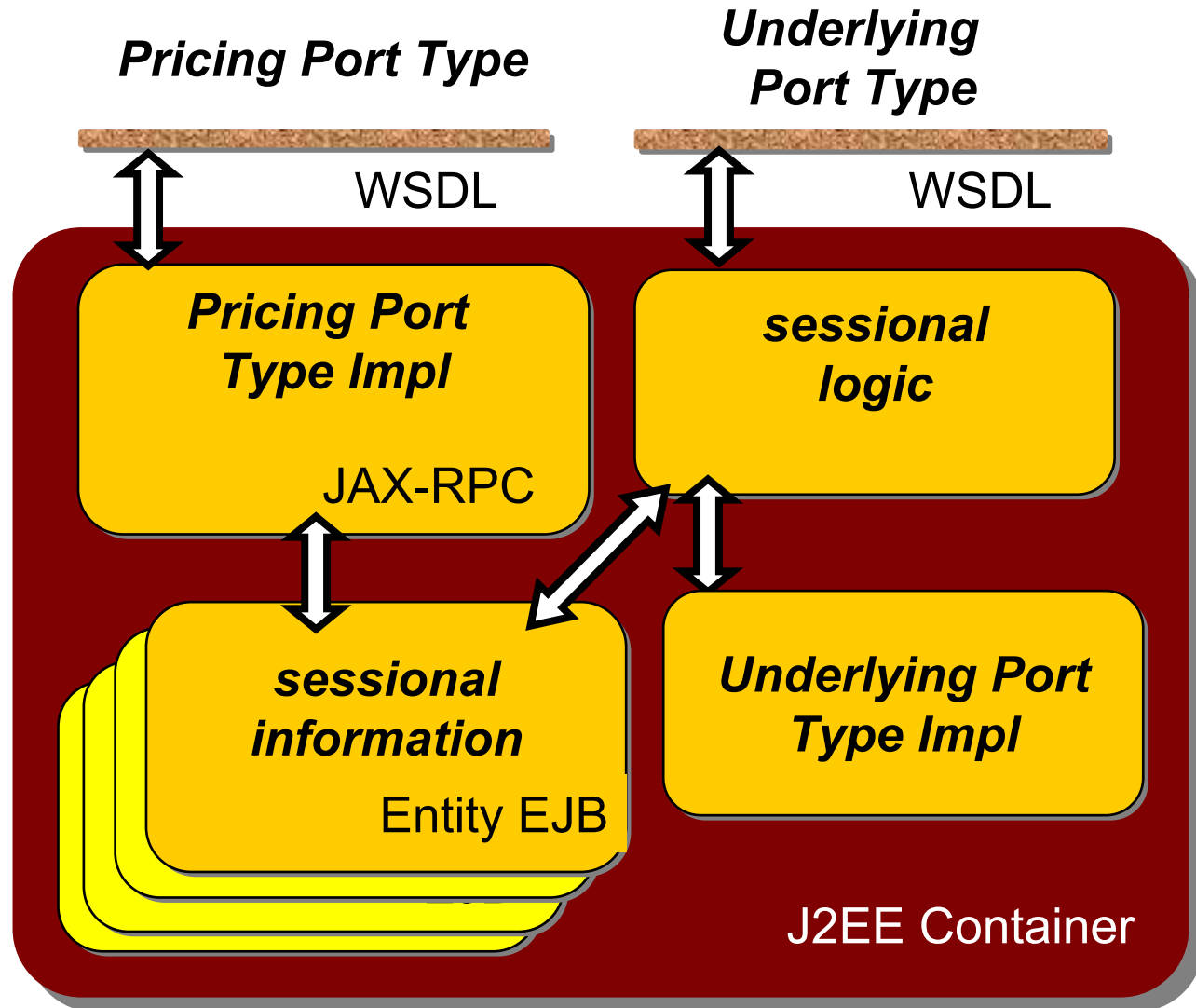
# Implementation Status: Payment Provider



# Implementation Status (*in progress*): Chargeable Web Service

Evaluating session mechanisms:

- embedding signed document into SOAP headers
- use WS-Context or similar



- Evaluation of Monetary Payment Systems
- Complex Negotiation and Pricing
  - Allows us to introduce economic theory
  - Tradable contracts
- Composition of Chargeable Services
  - Integrated Fault Management
  - Workflow Optimisation
- Integration of Brokering Systems
  - e-Science North West Centre

1. Economic Service features enable a *public* shared resource grid – not just a scheduling mechanism
2. Introduction & Negotiation can reuse existing standards and work within the GGF and wider community (SOA, WS-A)
3. Settlement & Execution /Reconciliation requires sessional web service features – but otherwise can use off-the-shelf WS-Interoperability standards (WSDL, SOAP)