BSPL, the Blindingly Simple Protocol Language

Munindar P. Singh

North Carolina State University

April 2011
Interactions and Protocols

All actions are interactions

**Goal:** Specify distributed systems of autonomous, heterogeneous agents

- Focus on roles that agents play
- Identify rules of encounter
- Maintain independence from internal reasoning (policies)

**Approach:** Specify protocols as abstractions over interactions
## Traditional Approaches

<table>
<thead>
<tr>
<th></th>
<th><strong>Declarative</strong></th>
<th><strong>Procedural</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meaning</strong></td>
<td>Commitments and other norms</td>
<td>Hard coded within internal reasoning heuristics</td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>Temporal logic</td>
<td>State machines; Petri nets; process algebras</td>
</tr>
</tbody>
</table>

- Declarative approaches for meaning
  - Improve flexibility
  - Under-specify enactment: potential of interoperability failures
- Procedural or declarative approaches for operations
  - Operationally clear, but
    - Tend to emphasize control flow
    - Tend to over-specify operational constraints
    - Yield nontrivial interoperability and endpoint projections
Remark on Control versus Data Flow

- **Control flow**
  - Natural within a single computational thread
  - Exemplified by conditional branching
  - Presumes master-slave relationship across threads
  - Impossible between mutually autonomous parties because neither controls the other
  - May sound appropriate, but only because of long habit

- **Data flow**
  - Natural across computational threads
  - Explicitly tied to causality
Properties of Participants

- **Autonomy**
- **Myopia**
  - All choices must be local
- **Heterogeneity: local ≠ internal**
  - Local state
    - Public or observable
    - Typically, must be revealed for correctness
  - Internal state
    - Private
    - Must never be revealed to avoid false coupling
- **Shared nothing representation of local state**
  - Enact via messaging
Interaction Orientation
Interactions as first-class constructs

- **Protocol**
  - Abstract class (or interface) of interactions
  - Based on roles and parameters
- **Roles**
  - Local but not internal views of each agent
- **Parameters**
  - Distinguish different instances of the same protocol
- **Enact protocols via LoST: Local State Transfer**
Information Centrism

Characterize each interaction purely in terms of information

- Explicit causality
  - Flow of information coincides with flow of causality
  - No hidden control flows
  - No backchannel for coordination

- Keys
  - Uniqueness
  - Basis for completion

- Integrity
  - Must have bindings for some parameters
  - Analogous to NOT NULL constraints

- Immutability
  - Durability
  - Robustness: insensitivity to
    - Reordering by infrastructure
    - Retransmission: one delivery is all it needs
Motivation and Benefits

▶ Technical
  ▶ Statelessness
  ▶ Consistency
  ▶ Atomicity
  ▶ Natural composition

▶ Conceptual
  ▶ Make protocol designer responsible for specifying causality
  ▶ Avoid contortions of traditional approaches when causality is unclear
BSPL, the Blindingly Simple Protocol Language

Main ideas

- Only *two* syntactic notions
  -Declare a message schema: as an atomic protocol
  -Declare a composite protocol: as a bag of references (to existing protocols)

- Parameters are central
  -Provide a basis for expressing meaning in terms of bindings in protocol instances
  -Yield unambiguous specification of compositions through public parameters
  -Capture progression of a role’s knowledge
  -Capture the completeness of a protocol enactment
  -Capture uniqueness of enactments through keys

- Separate structure (parameters) from meaning (bindings)
  -Capture many important constraints purely structurally
Parameter Adornments in BSPL

Capture the essential causal structure of a protocol

- \(\text{in}\): Information that must be provided to instantiate a protocol
  - Bindings must exist locally in order to proceed
  - Bindings must be produced through some other protocol

- \(\text{out}\): Information that is generated by the protocol instances
  - Bindings can be fed into other protocols through their \(\text{in}\) parameters, thereby accomplishing composition
  - A standalone protocol must adorn all its public parameters \(\text{out}\)

- \(\text{nil}\): Information that is absent from the protocol instance
  - Bindings must not exist

Ignoring data types of parameters for simplicity: assume strings everywhere
Key Parameters in BSPL
Marked as «key»

- All the key parameters together form the key
- Each protocol must define at least one key parameter
- Each message or protocol reference must have at least one key parameter in common with the protocol in whose declaration it occurs
- The key of a protocol provides a basis for the uniqueness of its enactments
The **Hello** Protocol

\[
\text{Hello} \begin{array}{c}
\text{role} \quad \text{Self, Other} \\
\text{parameter} \quad \text{out \ greeting \ key}
\end{array}
\]

\[
\text{Self} \rightarrow \text{Other}: \text{hi[ out \ greeting \ key]}
\]

- At most one instance of *Hello* for each greeting
- At most one *hi* message for each greeting
- Enactable standalone: no parameter is \(\text{in}\)
- The key of *hi* is explicit; could be made implicit
The Pay Protocol

\[
\text{Pay } \{ \\
\quad \text{role Payer, Payee} \\
\quad \text{parameter in ID key, in amount} \\
\}
\]

- At most one \textit{payM} for each ID
- Not enactable standalone: \textit{why}?
- The key of \textit{payM} is implicit; could be made explicit
The Offer Protocol

Offer { 
  role Buyer, Seller 
  parameter in ID key, out item, out price 
  
  Buyer → Seller: rfq [in ID, out item] 
  Seller → Buyer: quote [in ID, in item, out price] 
}

- The key ID uniques instances of *Initiate Offer*, *rfq*, and *quote*
- Not enactable standalone: at least one parameter is «in»
- An instance of *rfq* must precede any instance of *quote* with the same ID: *why?*
- No message need occur: *why?*
- *quote* must occur for *Initiate Offer* to complete: *why?*
The *Initiate Order* Protocol

\[ \text{Initiate-Order} \begin{cases} \text{role} B, S \\
\text{parameter} \quad \text{out} \ ID, \ \text{out} \ key, \ \text{out} \ item, \ \text{out} \ price, \ \text{out} \ rID \\
B \mapsto S: \text{rfq} [\text{out} \ ID, \ \text{out} \ item] \\
S \mapsto B: \text{quote} [\text{in} \ ID, \ \text{in} \ item, \ \text{out} \ price] \\
B \mapsto S: \text{accept} [\text{in} \ ID, \ \text{in} \ item, \ \text{in} \ price, \ \text{out} \ rID] \\
B \mapsto S: \text{reject} [\text{in} \ ID, \ \text{in} \ item, \ \text{in} \ price, \ \text{out} \ rID] \end{cases} \]

- The key ID uniquifies instances of *Order* and each of its messages
- Enactable standalone
- An *rfq* must precede a *quote* with the same ID
- A *quote* must precede an *accept* with the same ID
- A *quote* must precede a *reject* with the same ID
- An *accept* and a *reject* with the same ID cannot both occur: why?
The *Purchase* Protocol

Purchase \{  
role B, S, Shipper  
parameter out ID \text{ key} ,  out item ,  out price ,  out outcome  

B $\mapsto$ S: rfq [out ID, out item]  
S $\mapsto$ B: quote [in ID, in item, out price]  
B $\mapsto$ S: accept [in ID, in item, in price, out address, out resp]  
B $\mapsto$ S: reject [in ID, in item, in price, out outcome, out resp]  

S $\mapsto$ Shipper: ship [in ID, in item, in address]  
Shipper $\mapsto$ B: deliver [in ID, in item, in address, out outcome]  
\}

- At most one item, price, and outcome binding per ID  
- Enactable standalone  
- *reject* conflicts with *accept* on response (a *private* parameter)  
- *reject* or *deliver* must occur for completion (to bind outcome)
Possible Enactment as a History Vector

Buyer

rfq

ID, item

quote

ID, price

accept

ID, address

ship

ID, item, address

deliver

ID, item, address, outcome

Seller

rfq

quote

accept

Shipper

ship

deliver

deliver

singh@ncsu.edu (NCSU)
Blindingly Simple Protocol Language
April 2011 17 / 34
LoST Schematically
Local State Transfer

Internal Reasoning
Business Meaning
Local State
Messages

Application-Specific Protocols
LoST
Communication Infrastructure

Internal Reasoning
Business Meaning
Local State
Messages
Knowledge and Viability
When is a message viable? What effect does it have on a role’s local knowledge?

- Knowledge increases monotonically at each role
- An $\textit{out}$ parameter creates and transmits knowledge
- An $\textit{in}$ parameter transmits knowledge
- Repetitions through multiple paths are harmless and superfluous
<table>
<thead>
<tr>
<th></th>
<th>Sends in</th>
<th>Sends out</th>
<th>Sends nil</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sends in</td>
<td>Unconstrained</td>
<td>Send out first</td>
<td>Send nil first</td>
</tr>
<tr>
<td>Sends out</td>
<td></td>
<td>Send at most one</td>
<td>Send nil first</td>
</tr>
<tr>
<td>Sends nil</td>
<td>Receive first copy before send</td>
<td>Receive may occur after send</td>
<td>Unconstrained Send before receive</td>
</tr>
<tr>
<td>Receives in</td>
<td>Receive first copy before send</td>
<td>Impossible</td>
<td>Send before receive</td>
</tr>
<tr>
<td>Receives out</td>
<td>Receive first copy before send</td>
<td>Unconstrained</td>
<td>Send before receive</td>
</tr>
<tr>
<td>Receives nil</td>
<td>Unconstrained</td>
<td>Unconstrained</td>
<td>Unconstrained</td>
</tr>
</tbody>
</table>
## Comparing LoST and ReST

<table>
<thead>
<tr>
<th></th>
<th>ReST</th>
<th>LoST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modality</strong></td>
<td>Two-party; client-server; synchronous</td>
<td>Multiparty interactions; peer-to-peer; asynchronous</td>
</tr>
<tr>
<td><strong>Computation</strong></td>
<td>Server computes definitive resource state</td>
<td>Each party computes its definitive local state and the parties</td>
</tr>
<tr>
<td></td>
<td></td>
<td>collaboratively and (potentially implicitly) compute the definitive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interaction state</td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>Server maintains no client state</td>
<td>Each party maintains its local state and, implicitly, the relevant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>components of the states of other parties from which there is a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chain of messages to this party</td>
</tr>
</tbody>
</table>
## Comparing LoST and ReST

<table>
<thead>
<tr>
<th></th>
<th>ReST</th>
<th>LoST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transfer</strong></td>
<td>State of a resource, suitably represented</td>
<td>Local state of an interaction via parameter bindings, suitably represented</td>
</tr>
<tr>
<td><strong>Idempotent</strong></td>
<td>For some verbs, especially GET</td>
<td>Always; repetitions are guaranteed harmless</td>
</tr>
<tr>
<td><strong>Caching</strong></td>
<td>Programmer can specify if cacheable</td>
<td>Always cacheable</td>
</tr>
<tr>
<td><strong>Uniform interface</strong></td>
<td>GET, POST, ...</td>
<td>⌜ in ⌝, ⌜ out ⌝, ⌜ nil ⌝</td>
</tr>
<tr>
<td><strong>Naming</strong></td>
<td>Of resources via URIs</td>
<td>Of interactions via (composite) keys, whose bindings could be URIs</td>
</tr>
</tbody>
</table>
Comparing BSPL and WS-CDL

- Similarity: both emphasize interaction
- Differences: WS-CDL is
  - Operational
    - Sequential message ordering by default
  - Agent-oriented
    - Includes agents’ internal reasoning within choreography (specify what service an agent executes upon receiving a message)
    - Relies on agents’ internal decision-making to achieve composition (take a value from Chor A and send it in Chor B)
  - No semantic notion of completeness
Well-Formedness Conditions

- A parameter that is adorned \(\text{in}\) in a declaration must be \(\text{in}\) throughout its body.
- A parameter that is adorned \(\text{out}\) in a declaration must be \(\text{out}\) in at least one reference.
  - When adorned \(\text{out}\) in zero references, not enactable.
  - When adorned \(\text{out}\) in exactly one reference, consistency is guaranteed.
  - When adorned \(\text{out}\) in two or more references, no more than one can execute.
- A private parameter must be \(\text{out}\) in at least one reference.
ACID Properties
With inspiration from database transactions though with modifications

- Atomicity: if a protocol completes, each reference within it that is initiated also completes
  - Ensured by placing one agent in charge of each conflict
- Consistency: at most one of a set of conflicting references occurs
  - Ensured by placing one agent in charge of each conflict
- Isolation: separate enactments do not interfere
  - Ensured by keys
- Durability: any enactment is permanent
  - Ensured by the immutability of bindings
References: Analogous to Macros or Procedures?

- **Macro**
  - Expanded into the body of a composite protocol: partially enactable
  - Maximize concurrency

- **Procedure**
  - All or none
  - Enable compositionality

- **BSPL treats references as both**
  - Enactment is maximally concurrent, at the level of individual messages
  - Atomicity avoids undesirable outcomes
Standing Order

As in insurance claims processing

Insurance−Claims { 
  \textit{role} Vendor, Subscriber  
  \textit{parameter} out policyNO, out reqForClaim \textit{key}, out claimResponse 

  Vendor $\mapsto$ Subscriber: createPolicy [out policyNO]  
  Subscriber $\mapsto$ Vendor: serviceReq [in policyNO, out reqForClaim]  
  Vendor $\mapsto$ Subscriber: claimService [in policyNO, in reqForClaim, out claimResponse] 
}

- Each claim corresponds to a unique policy and has a unique response
- One policy may have multiple claims
Flexible Sourcing of out Parameters
Buyer or Seller Offer

Buyer–or–Seller–Offer { role Buyer, Seller parameter in ID key, out item, out price, out confirmed

Buyer \mapsto \text{Seller}: \text{rfq}[\text{in} \text{ ID}, \text{out} \text{ item}, \text{nil} \text{ price}]
Buyer \mapsto \text{Seller}: \text{rfq}[\text{in} \text{ ID}, \text{out} \text{ item}, \text{out} \text{ price}]

\text{Seller} \mapsto \text{Buyer}: \text{quote}[\text{in} \text{ ID}, \text{in} \text{ item}, \text{out} \text{ price}, \text{out} \text{ confirmed}]
\text{Seller} \mapsto \text{Buyer}: \text{quote}[\text{in} \text{ ID}, \text{in} \text{ item}, \text{in} \text{ price}, \text{out} \text{ confirmed}]

\}

- The BUYER or the SELLER may determine the binding
- The BUYER has first dibs in this example
in-out Polymorphism

Flexible Offer

Flexible-Offer {  
role B, S  
parameter in ID key, out item, price, out qID

B \rightarrow S: \text{rfq}[\text{ID}, \text{out} \text{ item}, \text{nil} \text{ price}]  
B \rightarrow S: \text{rfq}[\text{ID}, \text{out} \text{ item}, \text{in} \text{ price}]  

S \rightarrow B: \text{quote}[\text{ID}, \text{in} \text{ item}, \text{out} \text{ price}, \text{out} \text{ qID}]  
S \rightarrow B: \text{quote}[\text{ID}, \text{in} \text{ item}, \text{in} \text{ price}, \text{out} \text{ qID}]  
}

- The price can be adorned with \textit{in} or \textit{out} in a reference to this protocol
The *Bilateral Price Discovery* protocol

\[
\begin{align*}
\text{BPD} & \{ \\
& \quad \underline{\text{role}} \ Taker, \ Maker \\
& \quad \underline{\text{parameter}} \ out \ reqID \ \underline{\text{key}}, \ out \ \underline{\text{query}}, \ out \ \underline{\text{result}} \\
& \quad \text{Taker} \leftrightarrow \text{Maker}: \text{priceRequest}[out \ reqID, \ out \ \text{query}] \\
& \quad \text{Maker} \leftrightarrow \text{Taker}: \text{priceResponse}[in \ reqID, \ in \ \text{query}, \ out \ \text{result}] \\
\} 
\end{align*}
\]
The Generalized Bilateral Price Discovery protocol

\[\begin{align*}
\text{GBPD} & \quad \{
\text{role} & \quad T, M \\
\text{parameter} & \quad \text{reqID}, \text{key}, \text{query}, \text{res}
\}
\end{align*}\]

\[\begin{align*}
T & \mapsto M: \text{priceRequest}[\text{out reqID}, \text{out query}] \\
T & \mapsto M: \text{priceRequest}[\text{in reqID}, \text{in query}] \\
M & \mapsto T: \text{priceResponse}[\text{in reqID}, \text{in query}, \text{out res}] \\
M & \mapsto T: \text{priceResponse}[\text{in reqID}, \text{in query}, \text{in res}]
\end{align*}\]
The *Multilateral Price Discovery* protocol

\[
\text{MPD} \{ \\
\quad \text{role} \ Taker, \ Exchange, \ Maker \\
\quad \text{parameter} \ out \ \text{reqID} \ key, \ out \ \text{query}, \ out \ \text{res} \\
\quad \text{GBP}(Taker, \ Exchange, \ out \ \text{reqID}, \ out \ \text{query}, \ in \ \text{res}) \\
\quad \text{GBP}(Exchange, \ Maker, \ in \ \text{reqID}, \ in \ \text{query}, \ out \ \text{res}) \\
\}
\]
Shopping Cart

Shopping Cart {
role B, S

parameter out ID key, out lineID key, out item, out qty, out price, out finalize

B ↦ S: create[out ID]
S ↦ B: quote[in ID, out lineID, in item, out price]
B ↦ S: add[in ID, in lineID, in item, out qty, in price]
B ↦ S: remove[in ID, in lineID]

S ↦ B: total[in ID, out sum]
B ↦ S: settle[in ID, in sum, out finalize]
B ↦ S: discard[in ID, out finalize]
}
Directions

- Implementation of LoST
- Methodology for specifying practical protocols
- Expansions of the language to handle role hierarchies
- Theoretical results
  - Decision procedures for judging consistent enactability
  - Treatment of recursive protocols