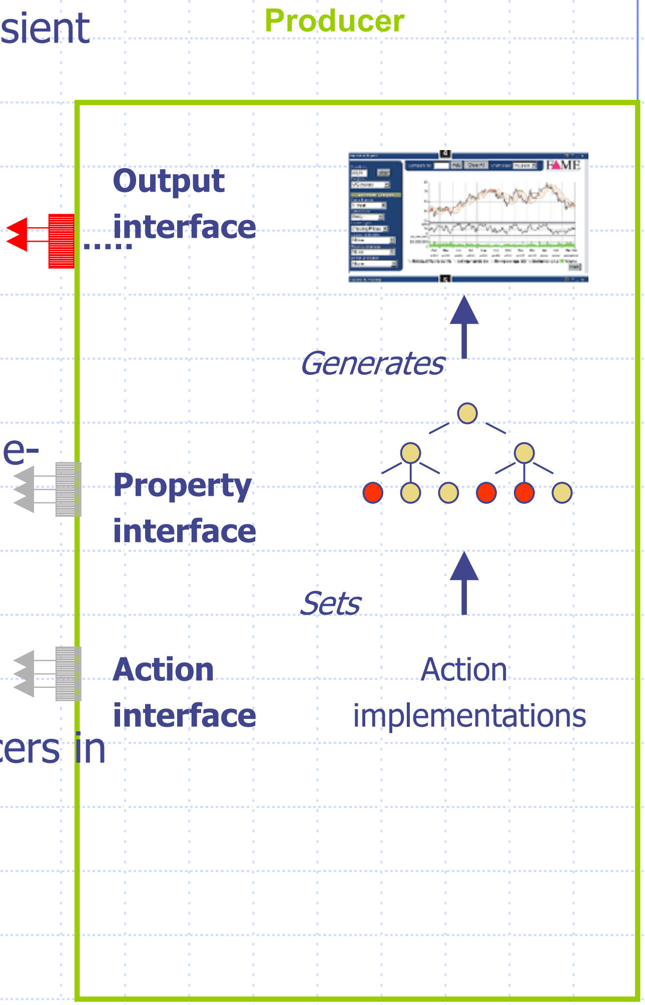
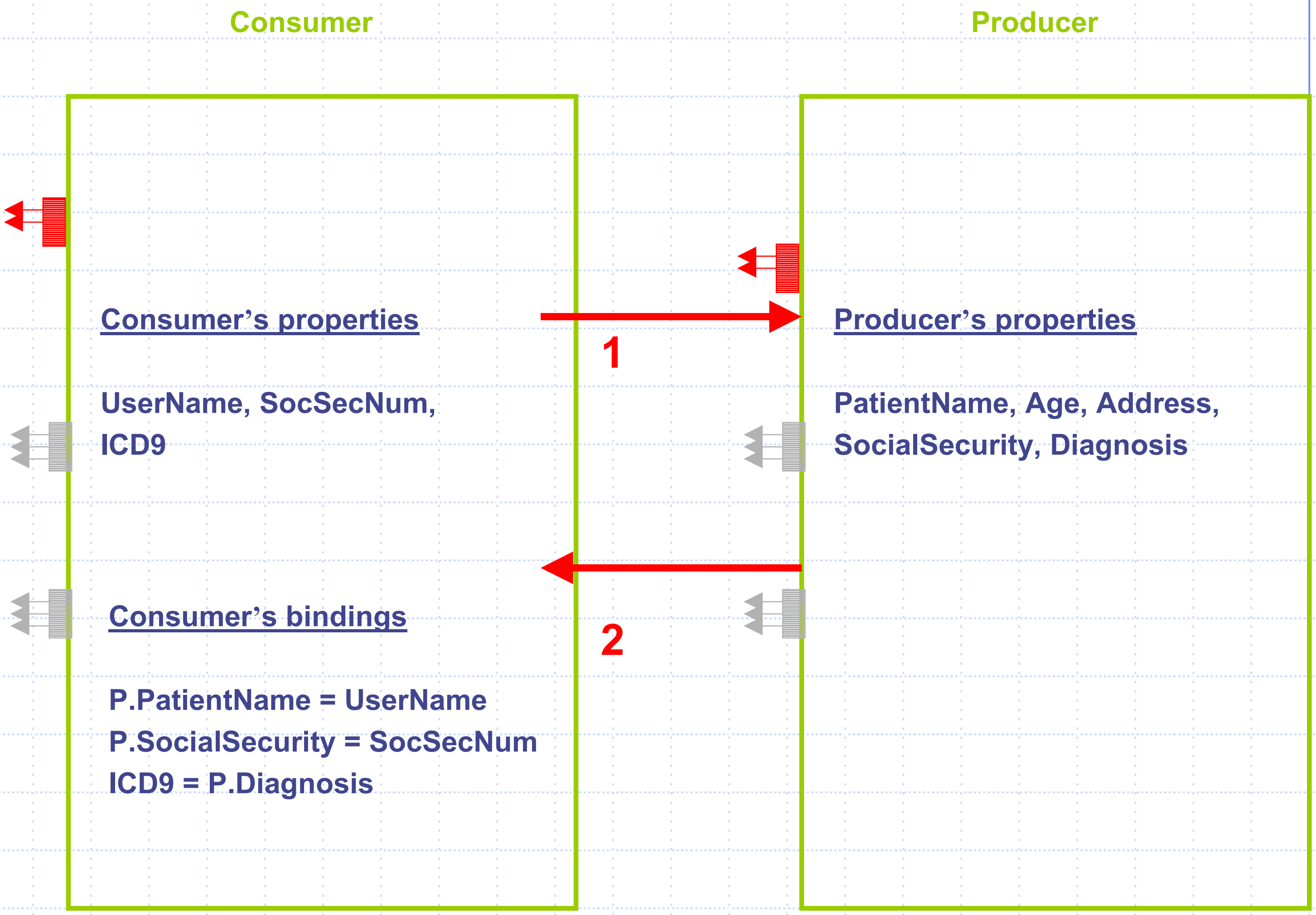
Property-oriented approach to Producer customization: Lifecycle of customizing and invoking a Producer

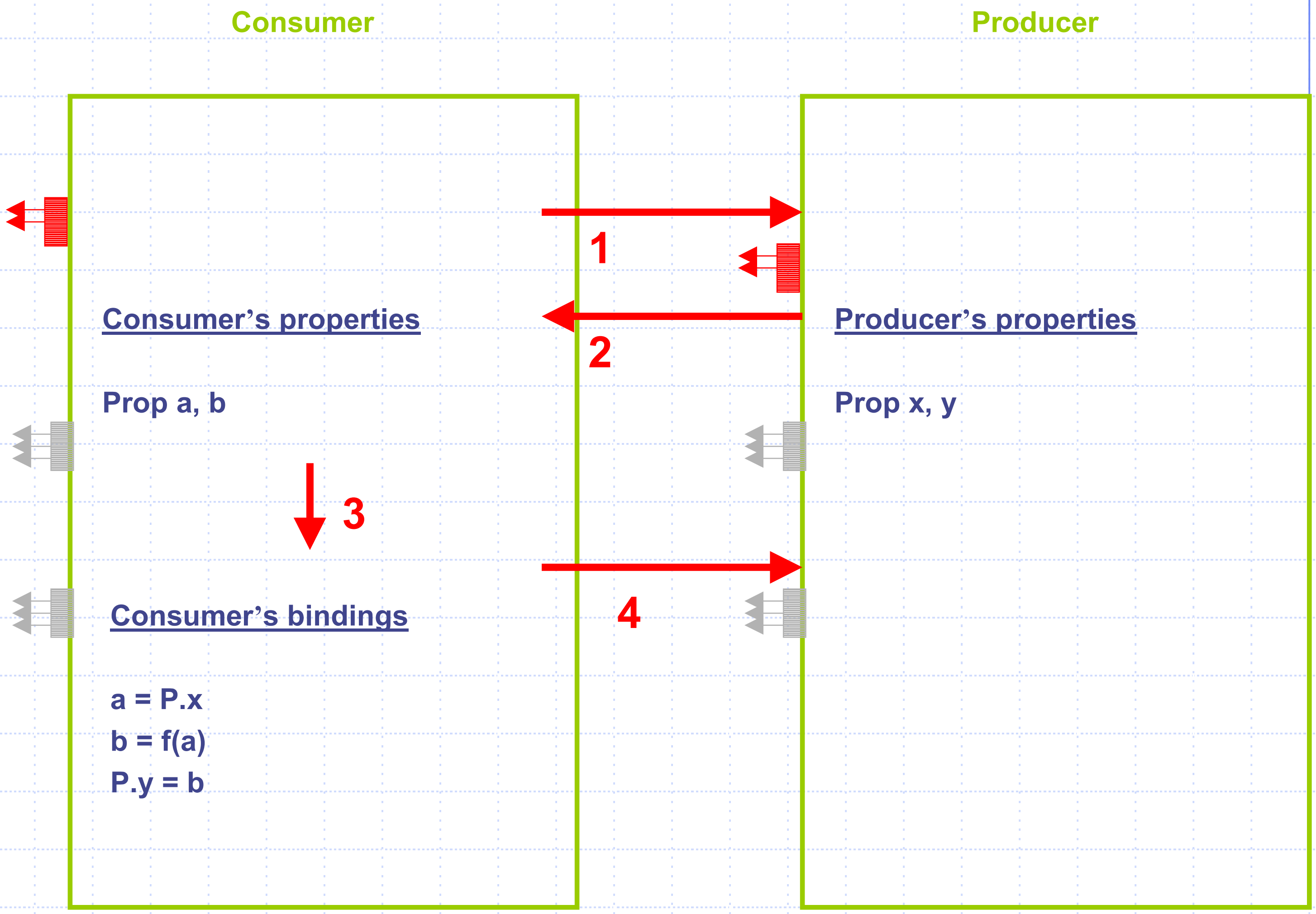
- Instantiation: from Template or Instance, create transient state with Consumer-applied property settings
- Request Output
- Invoke user Action (opaque arguments)
- Consumer receives Producer-determined property side-effects as return value from Action
- Consumer computes its own side effects
- Consumer applies property changes across all Producers in order determined by graph of property dependencies
- Request Output...



Example: Initializing data in Health Insurance Personal Profile Scenario



Example: Consumer propagates values between a Producer's properties



Example: The Memory Configurator and Availability scenarios

- Consumer writes bindings between its properties and Producer properties
- Bindings use a generalized XFORMS notation to define dependencies, and optionally calculations, between properties:
  - Consumer‘s manufacturer-part-num property is equal to the Producer‘s manufacturer part number column
  - Consumer computes its reseller-part-num from its manufacturer-part-num property
  - Producer‘s „user defined“ column is equal to the Consumer‘s reseller-part-num column

  - Cross-dependencies assure Consumer is passed Producer part numbers to generate reseller numbers before output is generated.
  - Producer has no a-priori knowledge of reseller numbers, or how to correlate them - totally generic column is used for this.

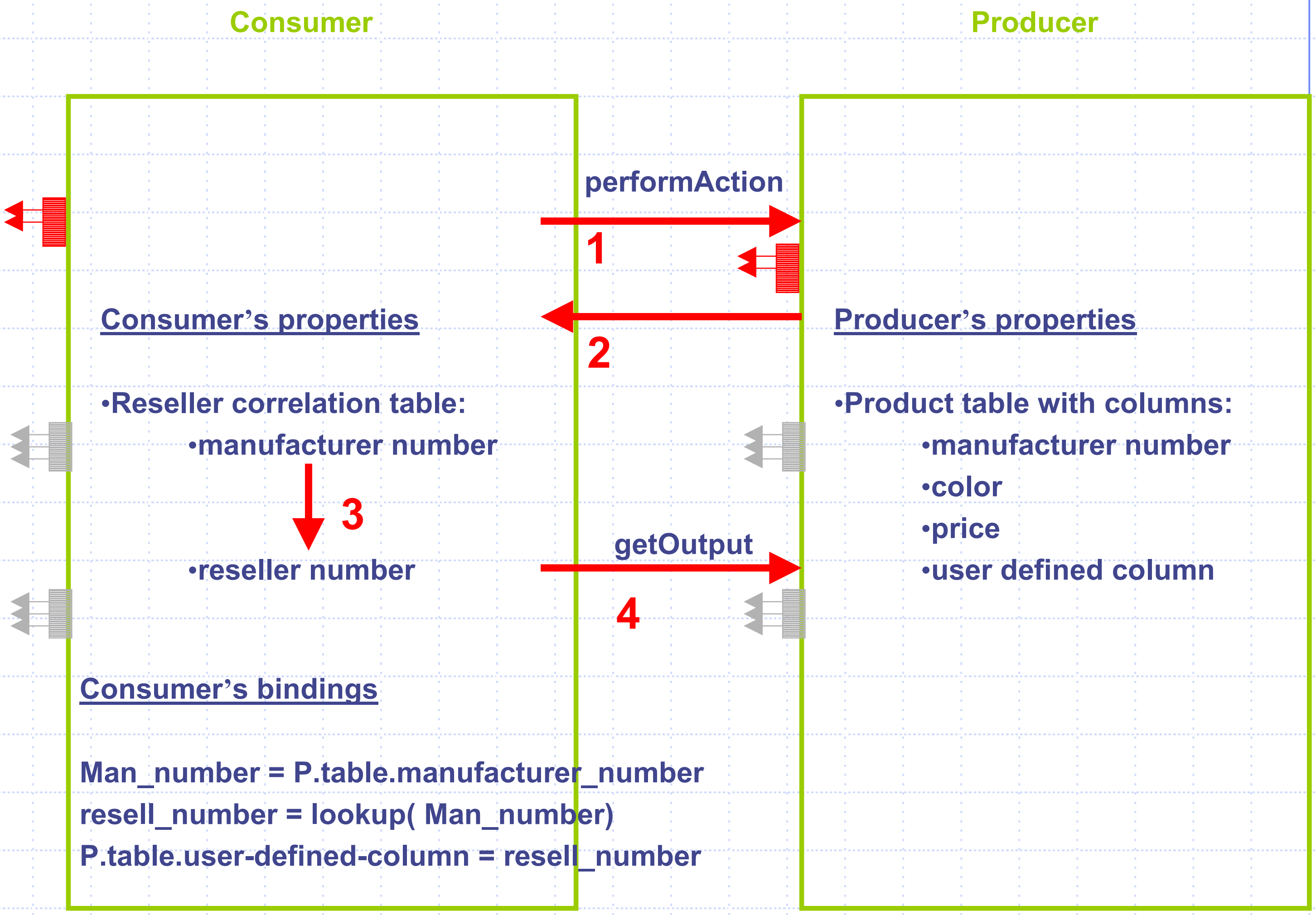
Producer

Producer’s properties

•Product table with columns:

•manufacturer number
•color
•price
•user defined column

Example: The Memory Configurator and Availability scenarios



Consumer's properties

•Reseller correlation table:
- •manufacturer number
- •reseller number

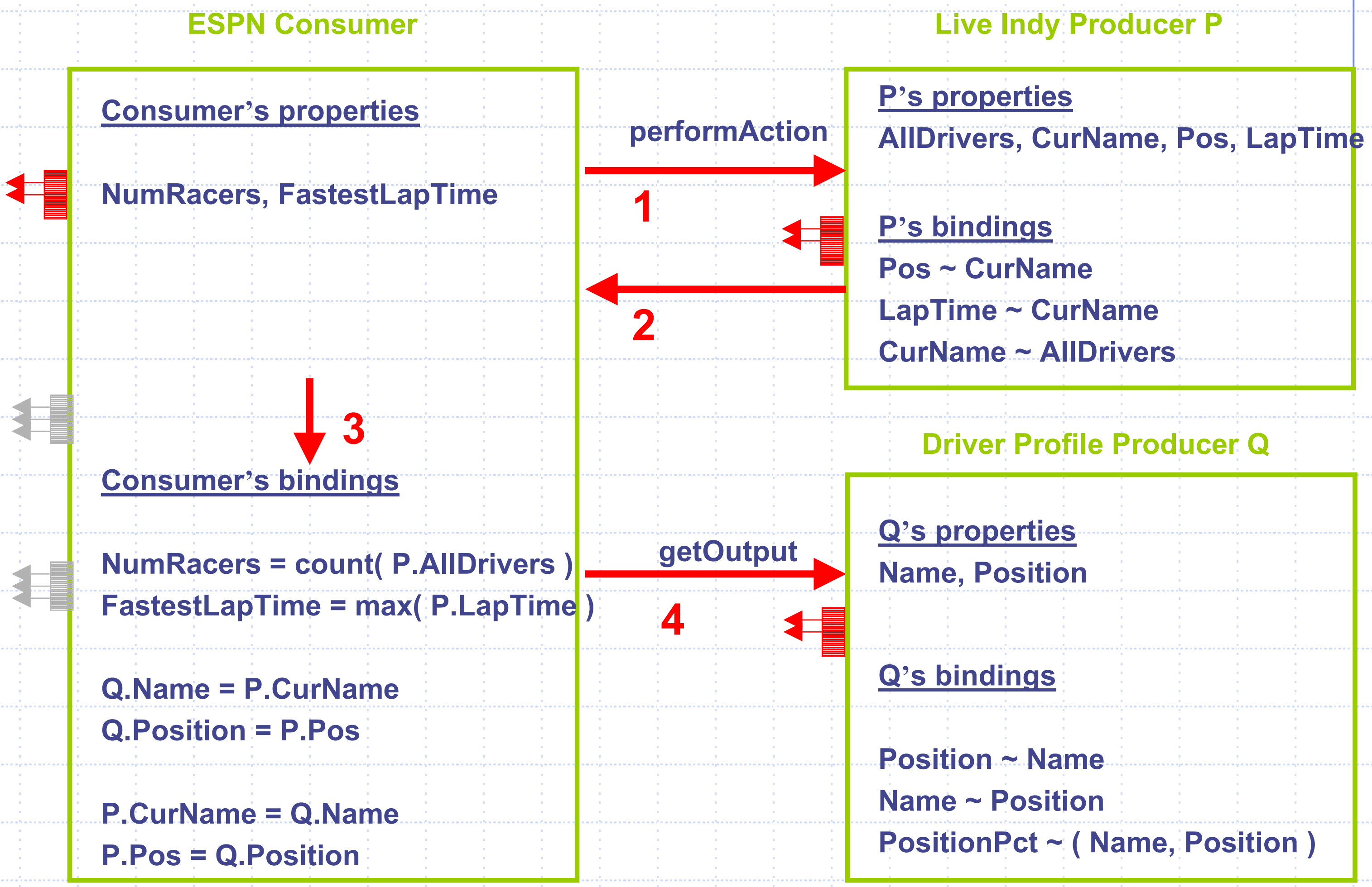
Producer's properties

•Product table with columns:
- •manufacturer number
- •color
- •price
- •user defined column

Consumer's bindings

```
Man_number = P.table.manufacturer_number
resell_number = lookup( Man_number)
P.table.user-defined-column = resell_number
```

Propagating values across Producers, a.k.a. Coordination Example: Indy car driver scenario



Backup

“Correlating” Producer and Consumer properties

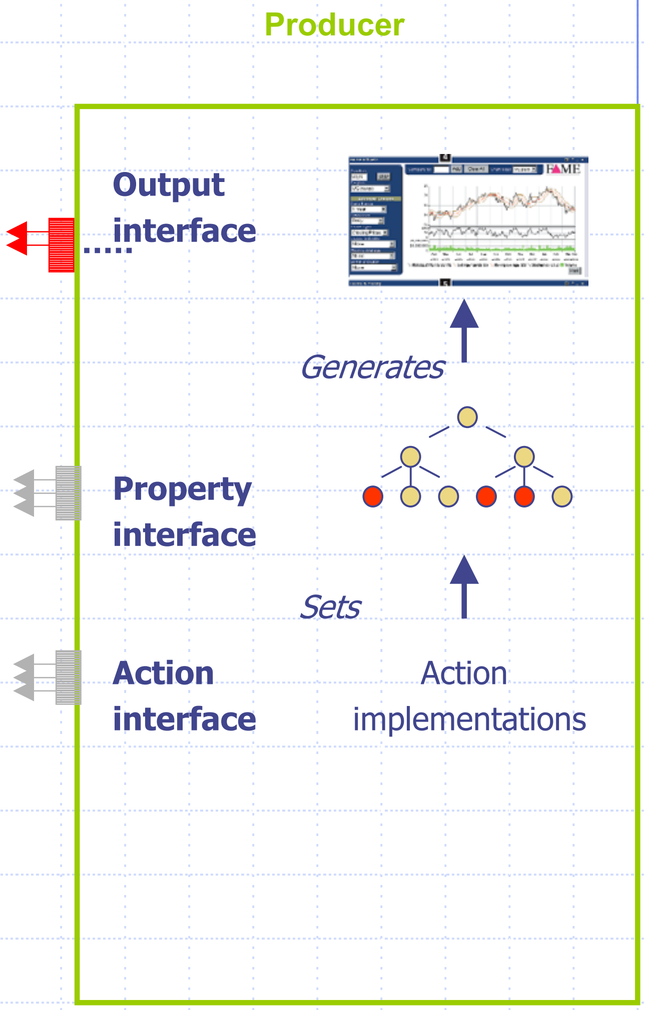
Scenarios: Knowledge of whether and how properties are related is split between Producer and Consumer

Examples:

- Producer determines manufacturer part numbers, Consumer correlates with reseller numbers
- Producer determines manufacturer part numbers, Consumer determines availability (column of booleans or first available ship dates)

Required infrastructure support:

- How to determine where there are Producer-Consumer interdependencies, and hence multiple passes to the Producer are required?
- Approach: Export property-to-property constraints from Producer to Consumer to support automated derivation of the update propagation sequence.



Business Scenarios and Use Cases
www.oasis-open.org/committees/wsia/scenarios/index.shtml

- Embedded
  - Consumer places the Producer's presentation web services inside a container as side-by-side independent applications.
- Customized
  - Model: Consumer sets Producer default values, restricts Producer types
  - View: Consumer alters look and feel styling, adds/deletes elements of Producer's view
  - Control: Consumer intercepts Producer actions
- Coordinated
  - Consumer integrates multiple Producers into a single user experience by wiring together their data and presentation states.

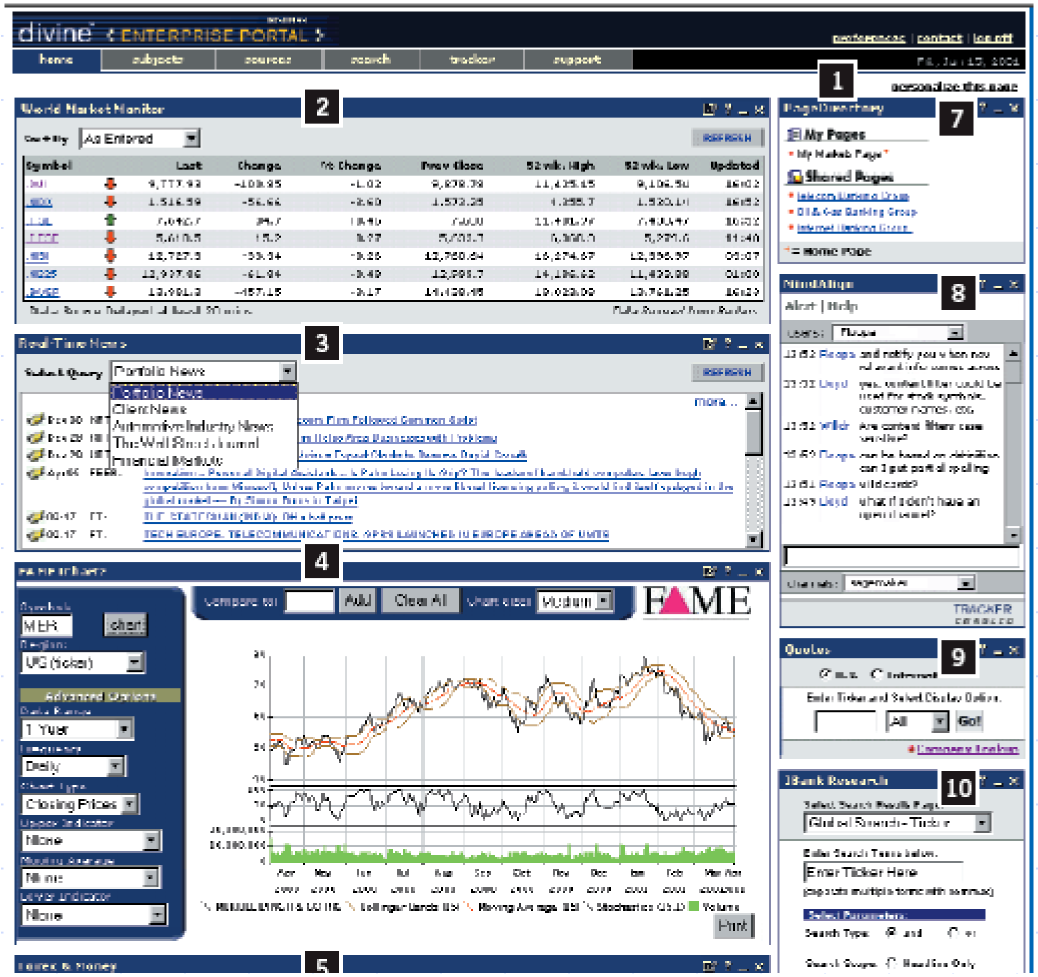
Business Scenarios and Use Cases
www.oasis-open.org/committees/wsia/scenarios/index.shtml

Customized

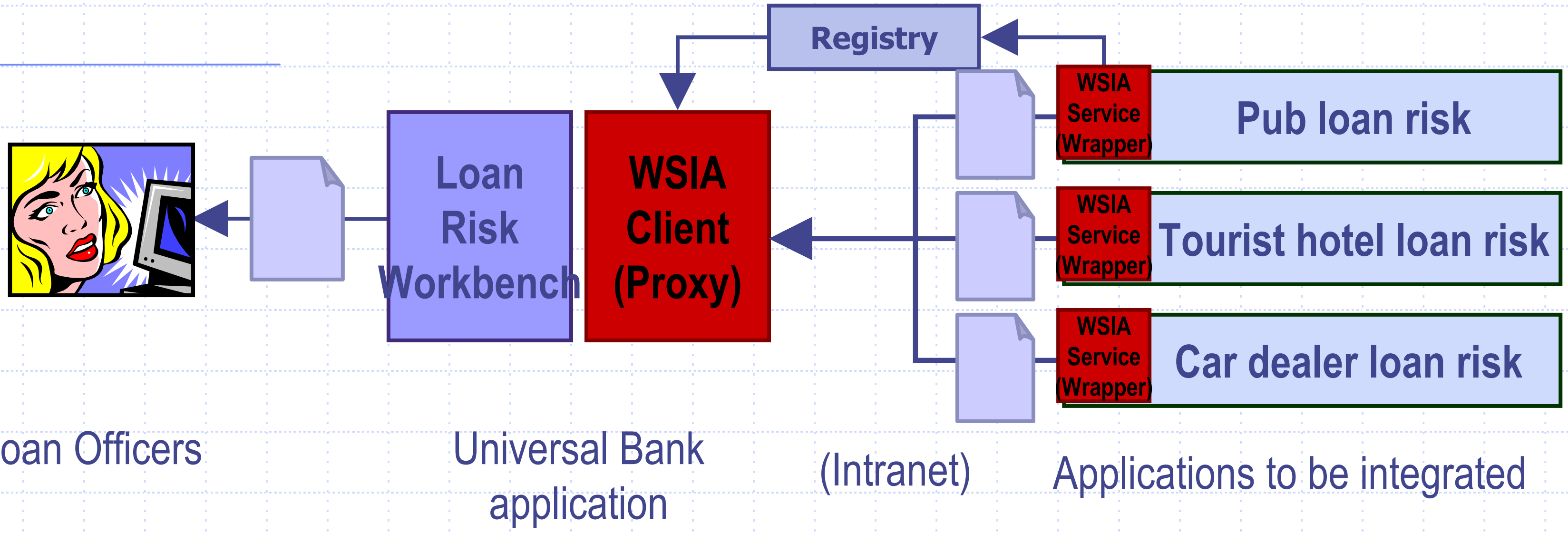
Model: Consumer sets Producer default values, restricts Producer types

View: Consumer alters look and feel styling, adds/deletes elements of Producer's view

Control: Consumer intercepts Producer actions

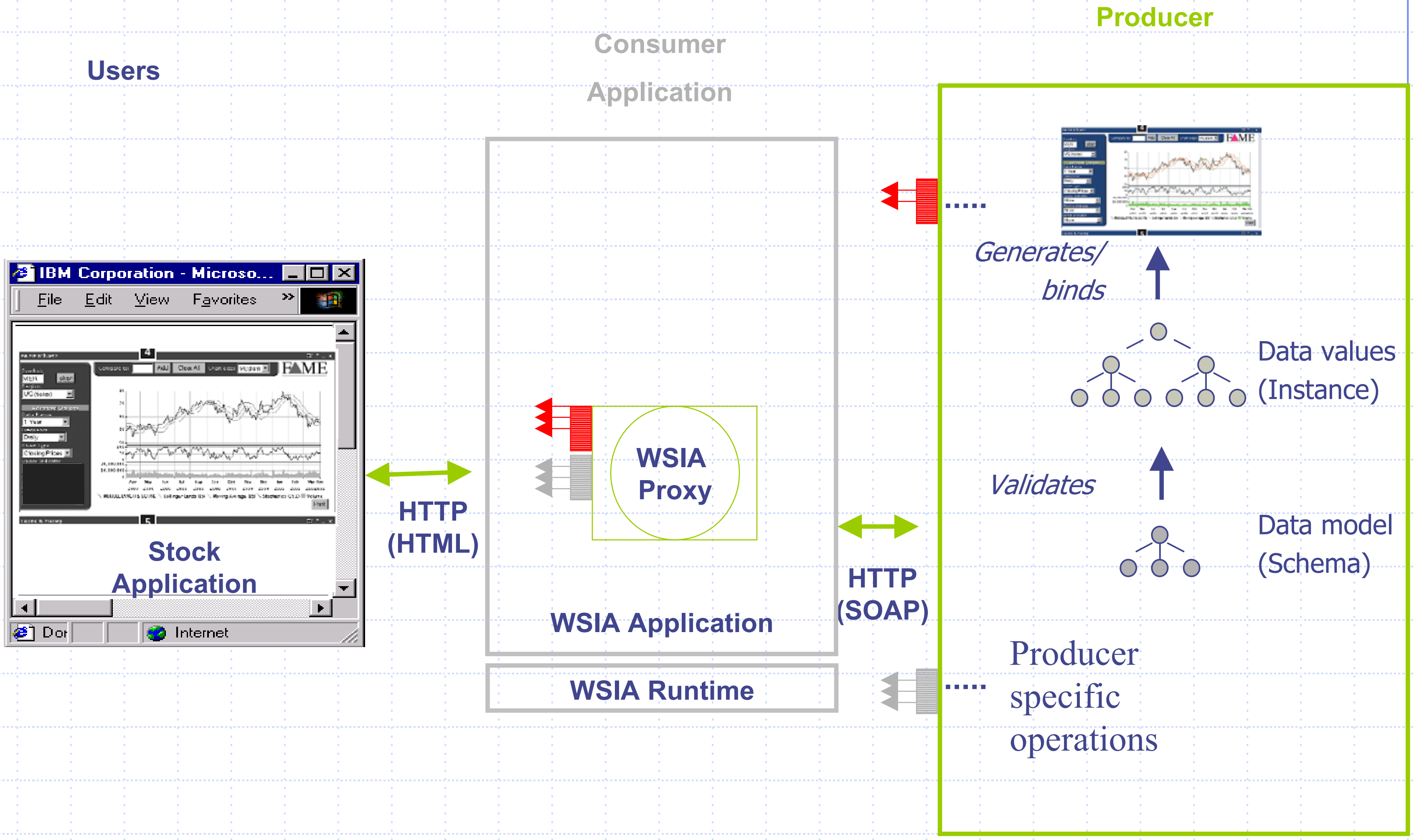


Scenario: Customization



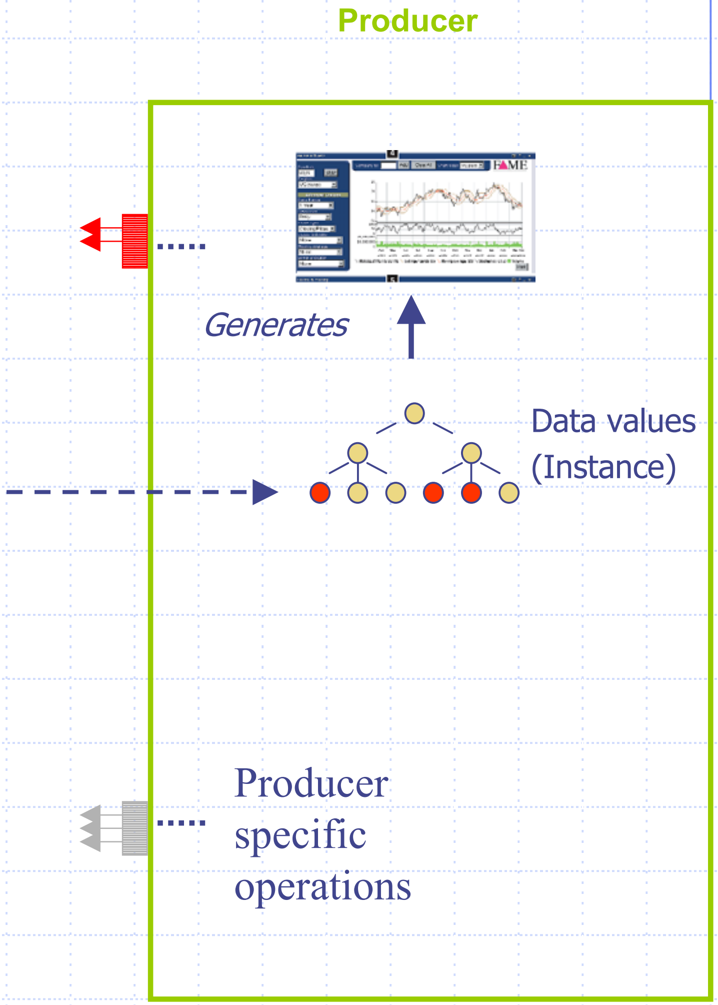
- Universal Bank wants to make the Loan Risk Applications of its subsidiaries available for use by its Loan Officers worldwide, without requiring central IT to become involved as the applications evolve
- The subsidiaries deploy their applications as WSIA services or wrap them externally.
- Universal Bank central IT develops the Loan Risk Workbench, which interacts with the Loan Officer, selects the right Loan Risk application, invokes it with the right parameters, and captures the return data

XFORMS-based approach to customization



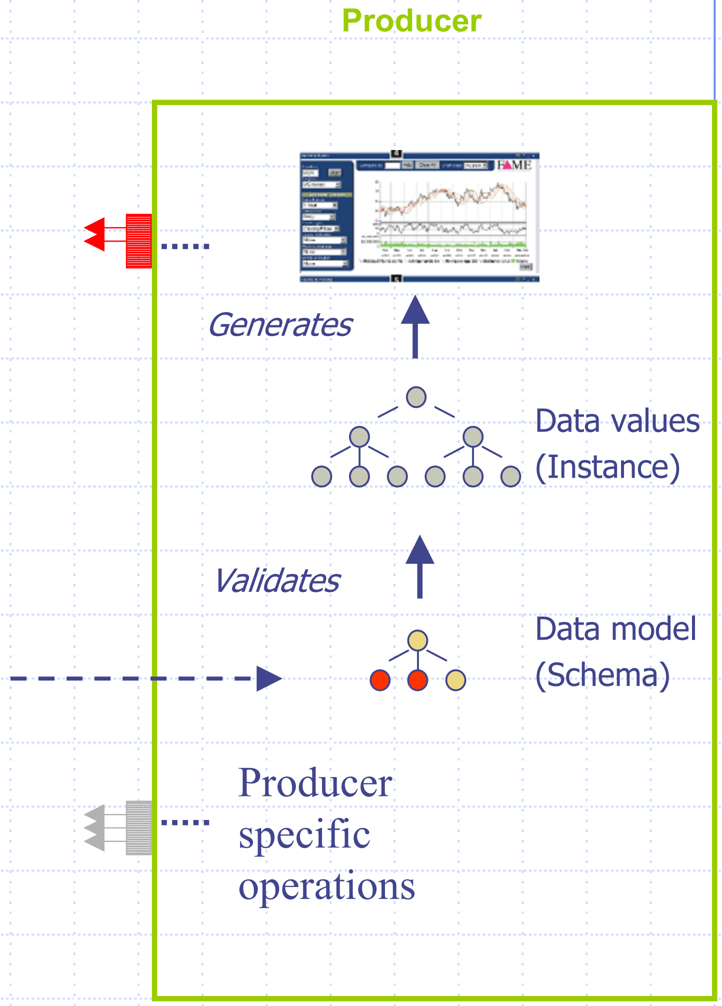
Model customization: Setting Data Values (instance)

- Consumer supplies initial data values for Producer-specific properties
- Producer executes and carries on dialog with end-user
- Producer supplies final data values to Consumer at end of dialog



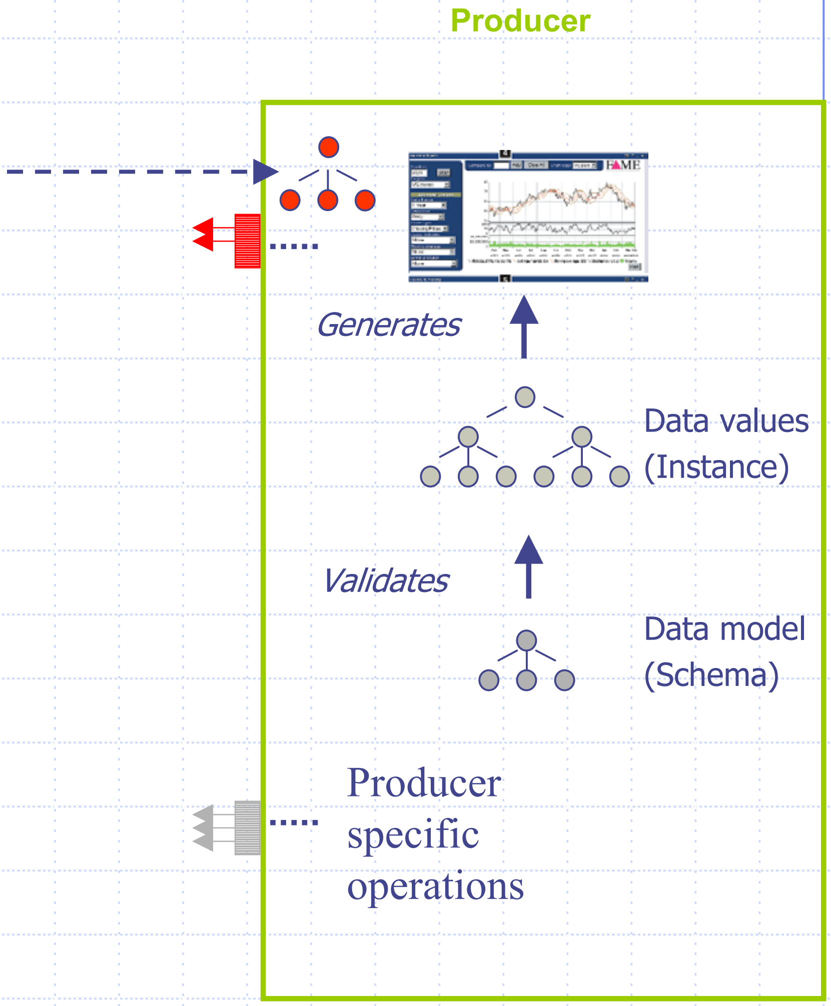
Model customization: Altering types (schema)

- Consumer adds restrictions to data model properties to control data values for Producer-specific properties, e.g. limits pull-down options, narrows data type constraints



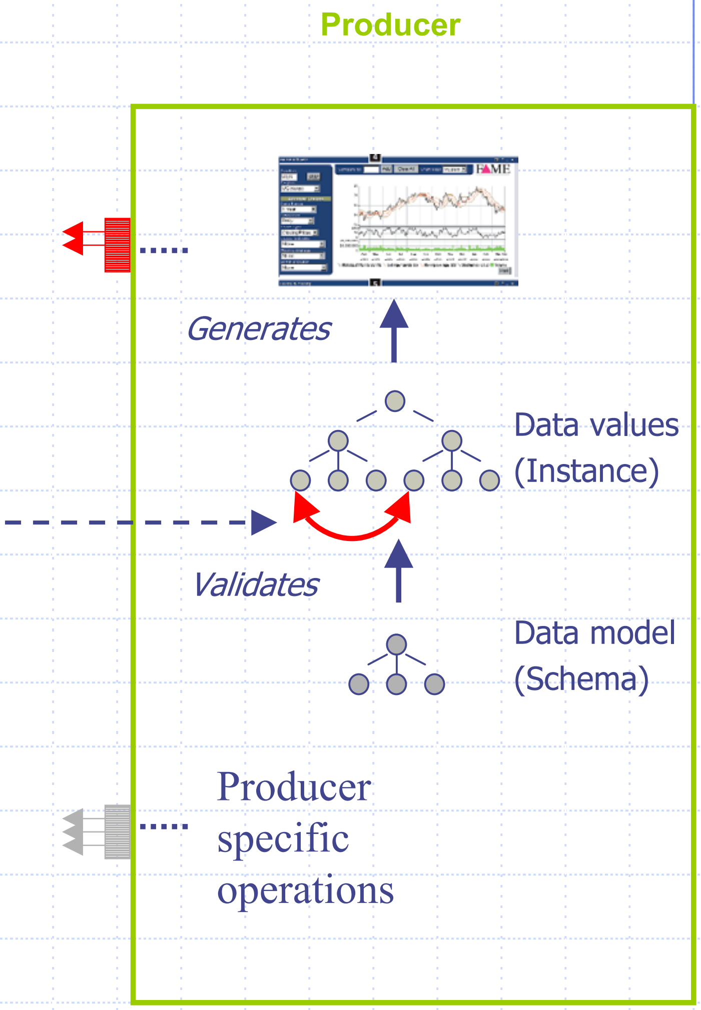
View customization

- Consumer inserts new markup for the Producer‘s view, respecting defined restrictions for the fragments the Producer can accept, e.g. provides formatting for cells in calendar‘s month view.



Control customization

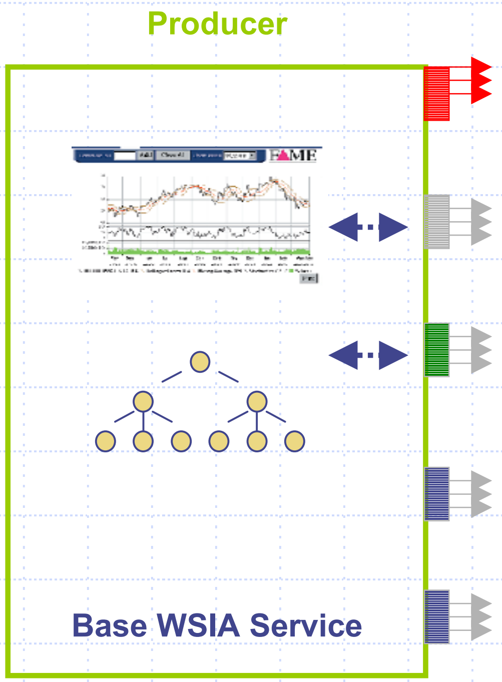
- Consumer inserts new constraints linking values across multiple data elements in the Producer's model



Customization Description Language

- Description ML for WSIA output markup and properties allows
  - Inserting/deleting/modifying output markup and properties
  - Setting formatting preferences, e.g.
    - calendar mode (day/week/month view, Sun/Mon first…)
  - In the future…driving view state, e.g.
    - which tab is selected
    - field focus
    - active field values (not yet validated for action inputs)
- Customizations can be run
  - At the Consumer - for privacy from the Producer; and performance by delegation of updates closer toward the user
  - At the Producer - for complex adaptations; for privacy, or to offload integration effort, from the Consumer

Overview of the WSIA interfaces



Lifecycle: createInstance, destroyInstance
Service description: getInterface, hasInterface

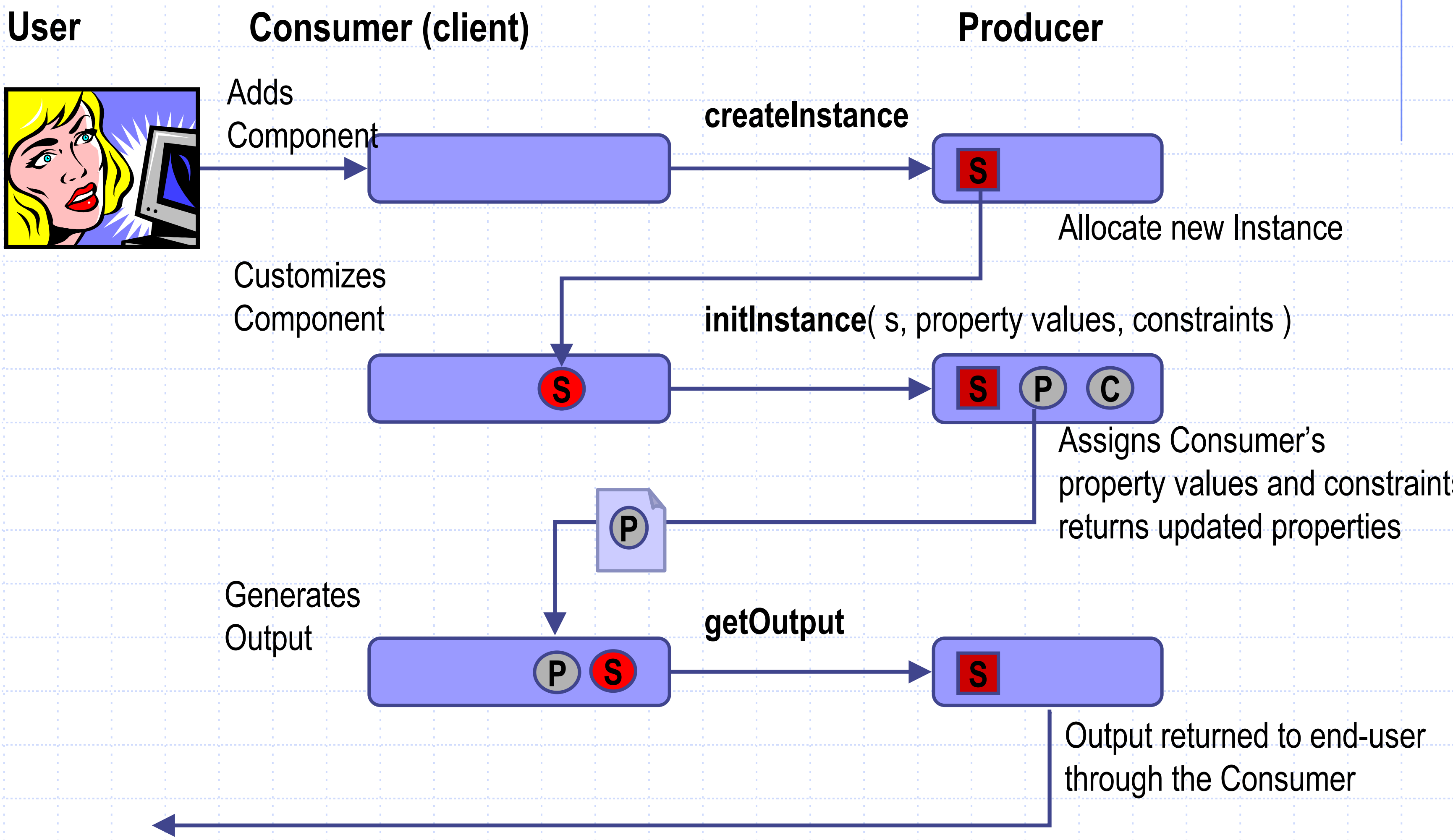
User interaction: getOutput, performAction

Properties: getPropertySchema, setProperties, getProperties

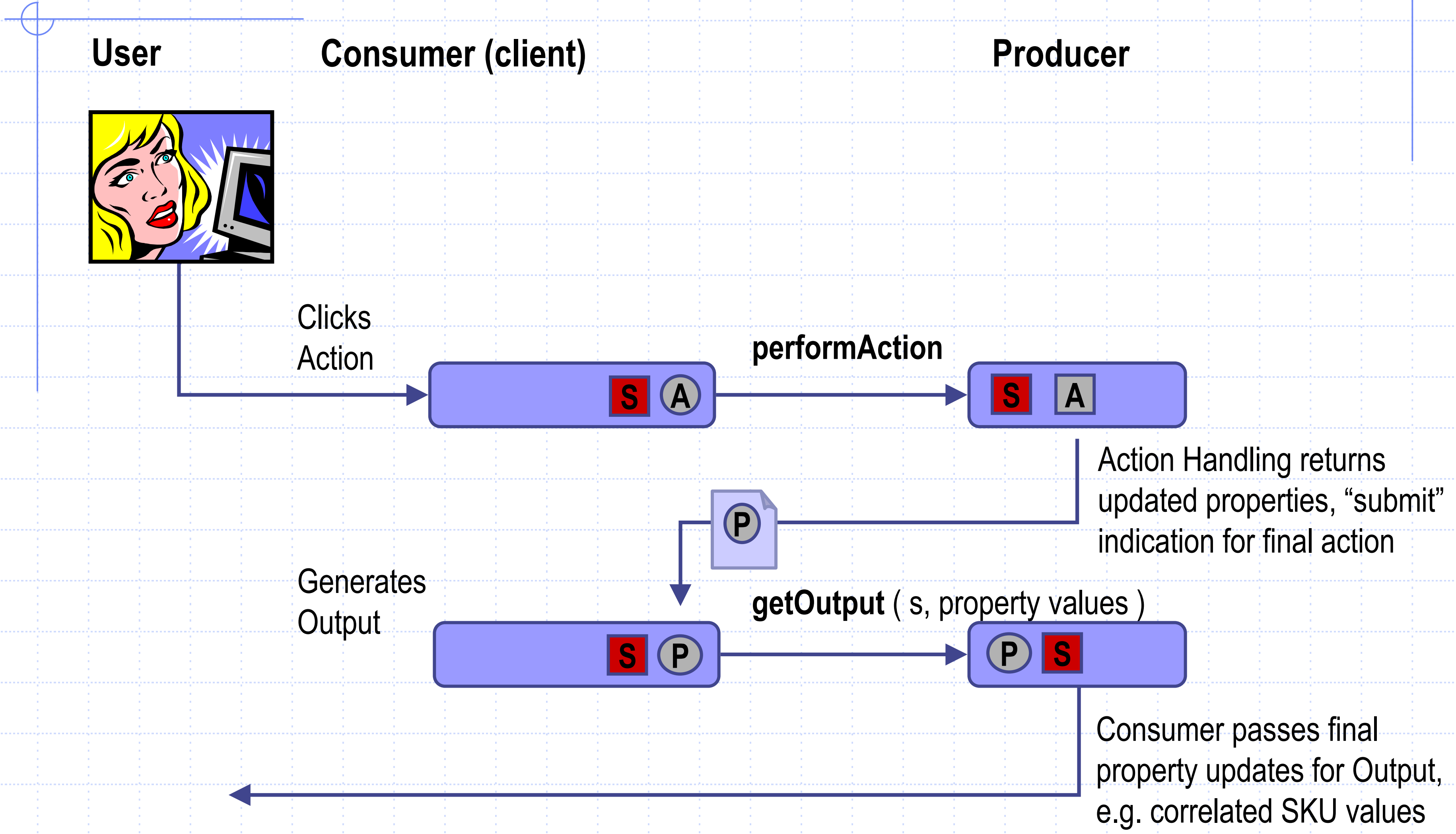
Property synchronization: addEventListener,
removeEventListener, handleEvent

Custom: service specific operations to affect either
properties or output

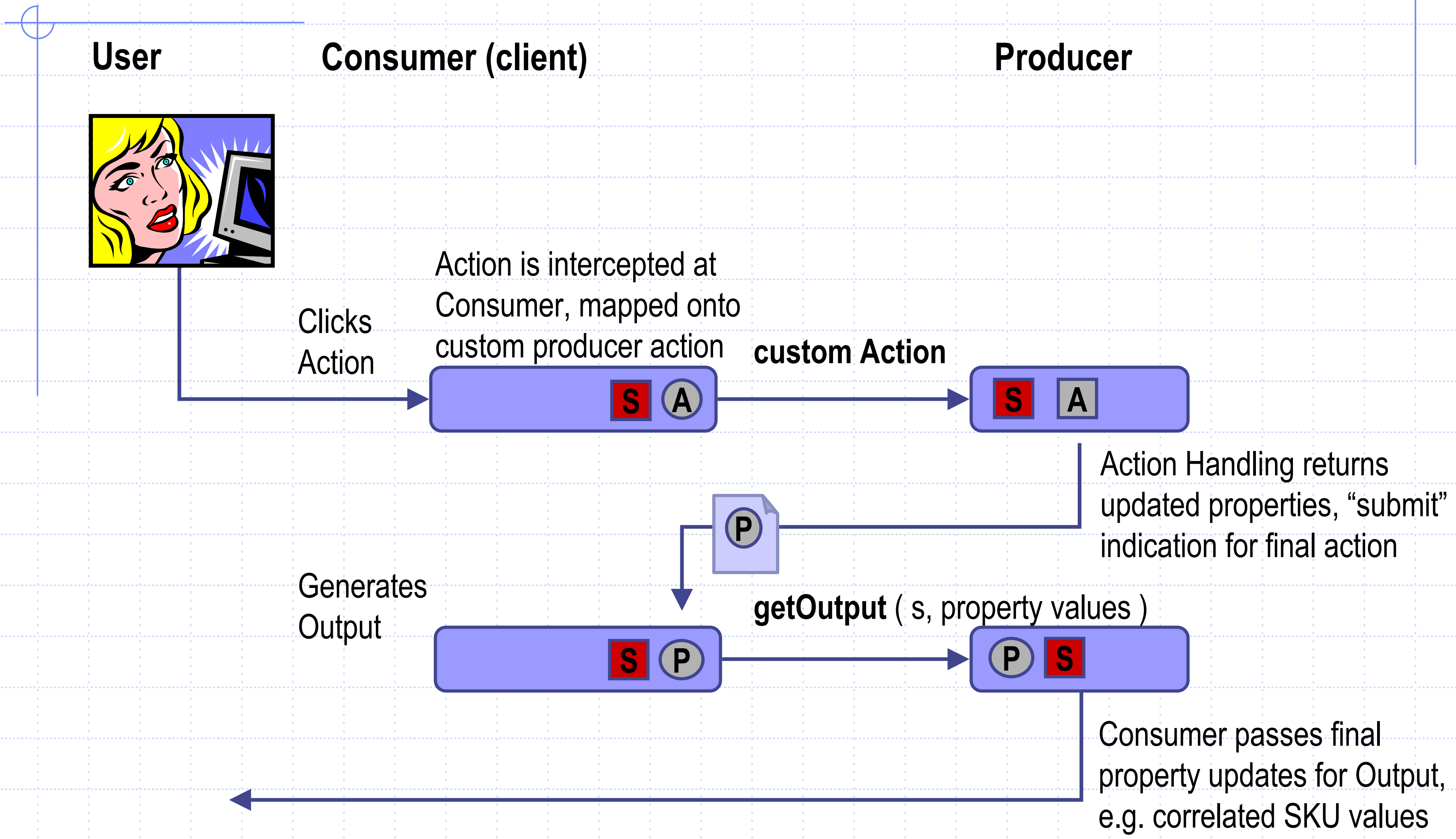
Consumer ⇔ Producer Interaction with Customization: Instantiation and initialization of Producer values and constraints


User
Consumer (client)
Producer
Adds Component
createInstance
S
Allocate new Instance
Customizes Component
initInstance(s, property values, constraints)
S
S
P
C
Assigns Consumer's property values and constraints
returns updated properties
P
Generates Output
getOutput
P
S
S
Output returned to end-user through the Consumer

Consumer ⇔ Producer Interaction with Customization: Two-pass action invocation


User
Consumer (client)
Producer
Clicks Action
S
A
performAction
S
A
Action Handling returns updated properties, “submit” indication for final action
P
Generates Output
getOutput (s, property values)
S
P
P
S
Consumer passes final property updates for Output, e.g. correlated SKU values

Consumer ⇔ Producer Interaction with Customization: Action invocation on Consumer markup embedded within Producer


User
Consumer (client)
Producer
Clicks Action
Action is intercepted at Consumer, mapped onto custom producer action
custom Action
S
A
S
A
Action Handling returns updated properties, "submit" indication for final action
P
Generates Output
getOutput (s, property values)
S
P
P
S
Consumer passes final property updates for Output, e.g. correlated SKU values

Customization protocol questions

- Should initInstance arguments be allowed on createInstance?
  - A separate initInstance allows for reuse of the same instance handle -- is this important?
- Should property constraints be allowed any time during a component's lifecycle?
  - On an action or getOutput invocation?
- How is "submit" indicated on a return value from performAction?
  - What if custom actions are allowed? Common message definition for "submit" is required
- What processing is required for user actions on embedded view fragments?
  - I.e. those view fragments inserted by the Consumer -- can they simply be intercepted by the Consumer?
  - If the Consumer then invokes a custom Producer action, it must have the same return message to inform the Consumer of changed property values and participate in the same 2-pass processing model to generate output