Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

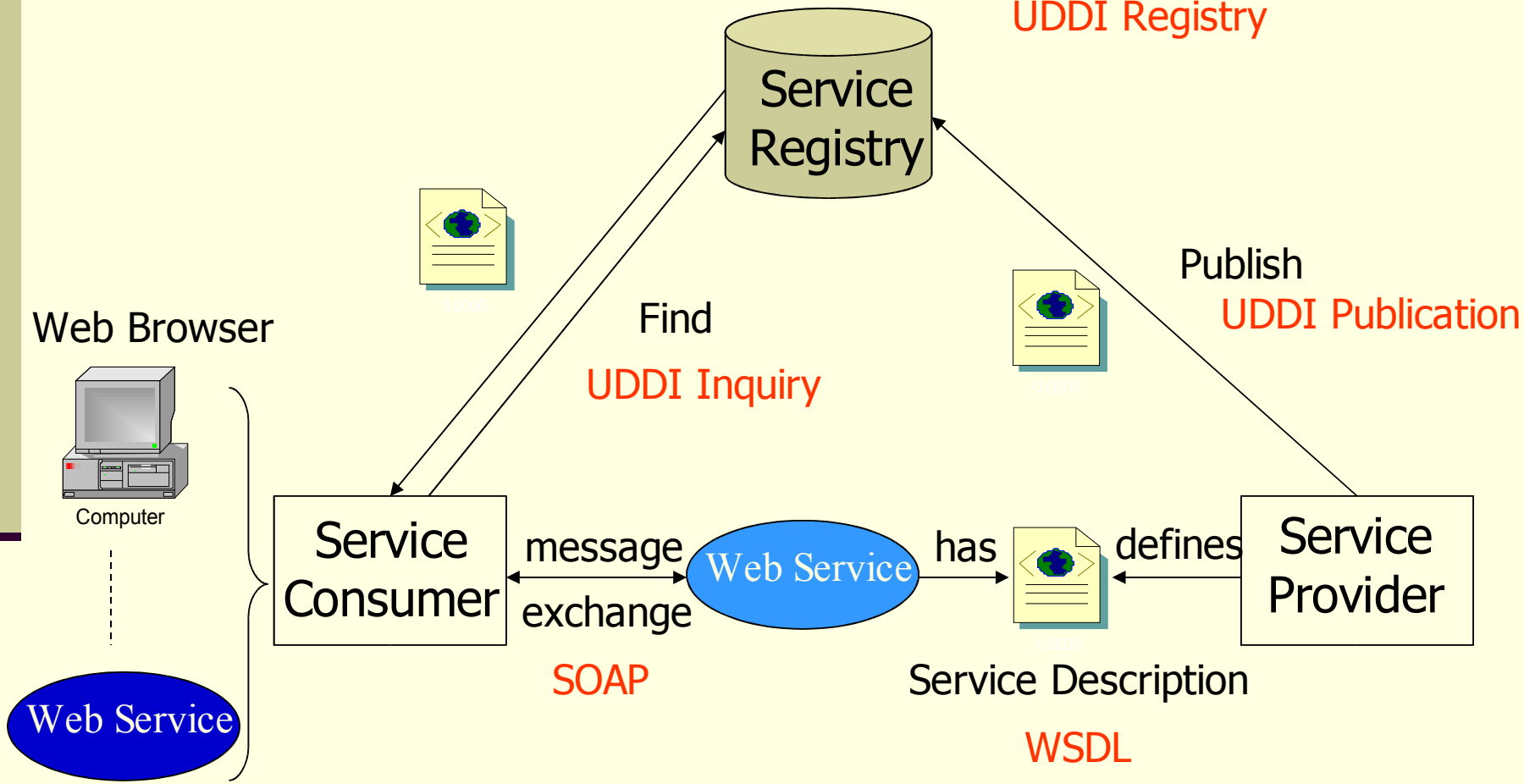
Supporting Dynamic Changes in Web Service Environments

M. Salman Akram, Brahim Medjahed,
Athman Bouguettaya
E-Commerce and E-Government Research Lab
Department of Computer Science, Virginia Tech, USA
{salman, brahim, athman} @vt.edu

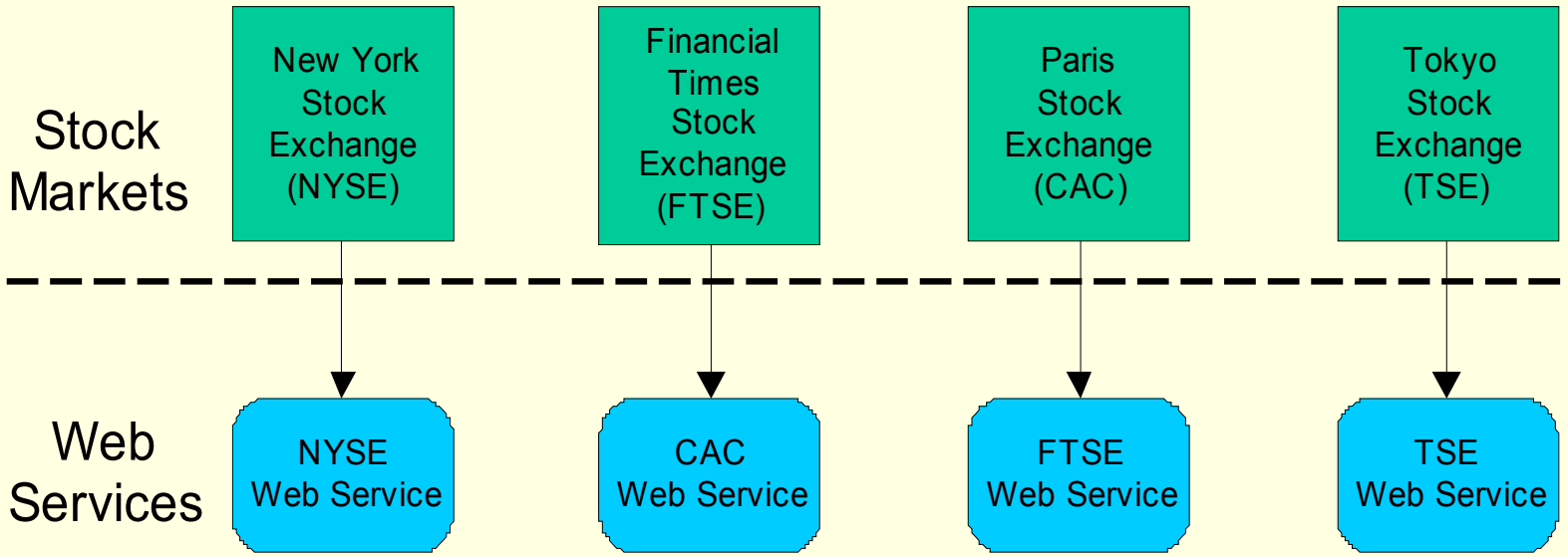
Agenda

- Introduction
- Web Service Requests
- Change Management
- Proposed Architecture
- Related work and Conclusion

The Web Service Model



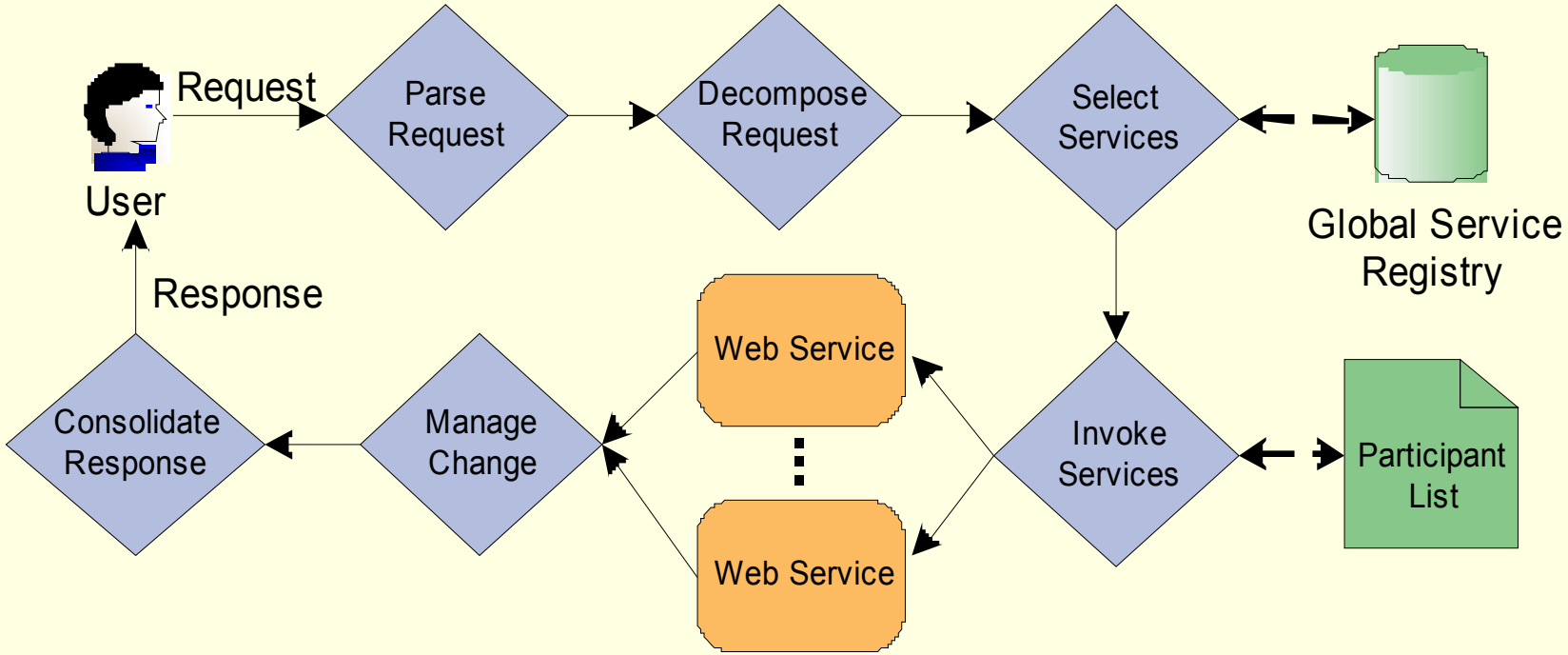
Scenario


Stock Markets
New York Stock Exchange (NYSE)
Financial Times Stock Exchange (FTSE)
Paris Stock Exchange (CAC)
Tokyo Stock Exchange (TSE)
Web Services
NYSE Web Service
CAC Web Service
FTSE Web Service
TSE Web Service

Scenario (cont.)

Sample Request:

Sell 500 shares of IBM stocks at the NYSE *if* a price of $21 dollars is reached within the next half hour



Motivation

- **Exploratory**
  - The process of selecting Web services is **non-deterministic**
  - Web services are a priori **unknown**
  - Services should be determined **dynamically**
- **Volatile**
  - A Web service answering a request at a given time may not be **available** to answer the same request in the future
  - Services may become unavailable in the interval *between* **selection** and **invocation**
  - Service may become unavailable *during* the **execution**
- **Dynamic**
  - Web service **content** provided by the operations may change frequently
  - Change may affect the **overall** execution of the request

Issues

- Web Service Discovery and Selection
  - Select Web services with **appropriate** functionality
  - More than one service might have to be selected
  - Must be discovered in a “reasonable” time

Issues (cont.)

- Adapting to Web Service Changes
  - Detect, propagate, and react to all “significant” changes in Web service environments
- Categories of changes:
  - **Internal**
    - Changes that occur inside a Web service (change in the data provided by a Web service)
    - Example: change in the price of a share
  - **External**
    - Changes that occur outside of a Web service (service availability)
    - Example: temporary or permanent unavailability of a Web service that provides functionality for the NYSE

Using Ontologies for Dynamic Discovery

- **Why Ontologies?**
  - Form a coherent slice of service space
  - Provide sharing of service description knowledge
  - UDDI was not designed to support relationships between Web services
  - Need to be familiar with the tModels in UDDI
- **DAML-S**
  - Provides the ability to organize Web services into ontologies
  - Allows embedding relationship information into service descriptions to facilitate dynamic discovery

Example Ontology Description

```
(1)  <daml:Class rdf:ID="NYSE">
(2)    <rdfs:label>NewYorkStockExchange</rdfs:label>
(3)    <rdfs:subClassOf rdf:resource="&service;"/>
(4)  <daml:Class>

(5)  <rdf:Property rdf:ID="Computer">
(6)    <rdfs:label>Bookstore</rdfs:label>
(7)    <rdfs:subPropertyOf rdf:resource="&profile;serviceCategory"/>
(8)    <rdfs:domain rdf:resource="&service;serviceProfile"/>
(9)    <rdfs:range rdf:resource="&daml;#Thing"/>
(10) </rdf:Property>

(11) <rdf:Property rdf:ID="Travel">
(12)   <rdfs:label>Travel</rdfs:label>
(13)   <rdfs:subPropertyOf rdf:resource="&profile;serviceCategory"/>
(14)   <rdfs:domain rdf:resource="&service;serviceProfile"/>
(15)   <rdfs:range rdf:resource="&daml;#Thing"/>
(16) </rdf:Property>
```

Managing Changes

- What is Change Management?
  - **Detection**
    - Awareness that a change has occurred
    - Subsequent identification of its cause
  - **Propagation**
    - Informing all concerned entities in the system that a change has occurred
  - **Reaction**
    - Executing a compensatory process that brings the system back to safe execution mode

Change Detection

- **Service unavailability** - agents send frequent *alive* messages to participant services
- **Change to operations** - compare service descriptions in the registries with the ones in the system
- **Change in content** - periodic invocation of the an operation and comparing the subsequent results

Change Propagation

- Web services participating in a service request are registered with a participant list
- Participant list is maintained by agents
  - Agents initially add Web service descriptions to the list
  - The Participant list is consulted before a service is invoked
  - Agents remove the service description from the list if change occurs

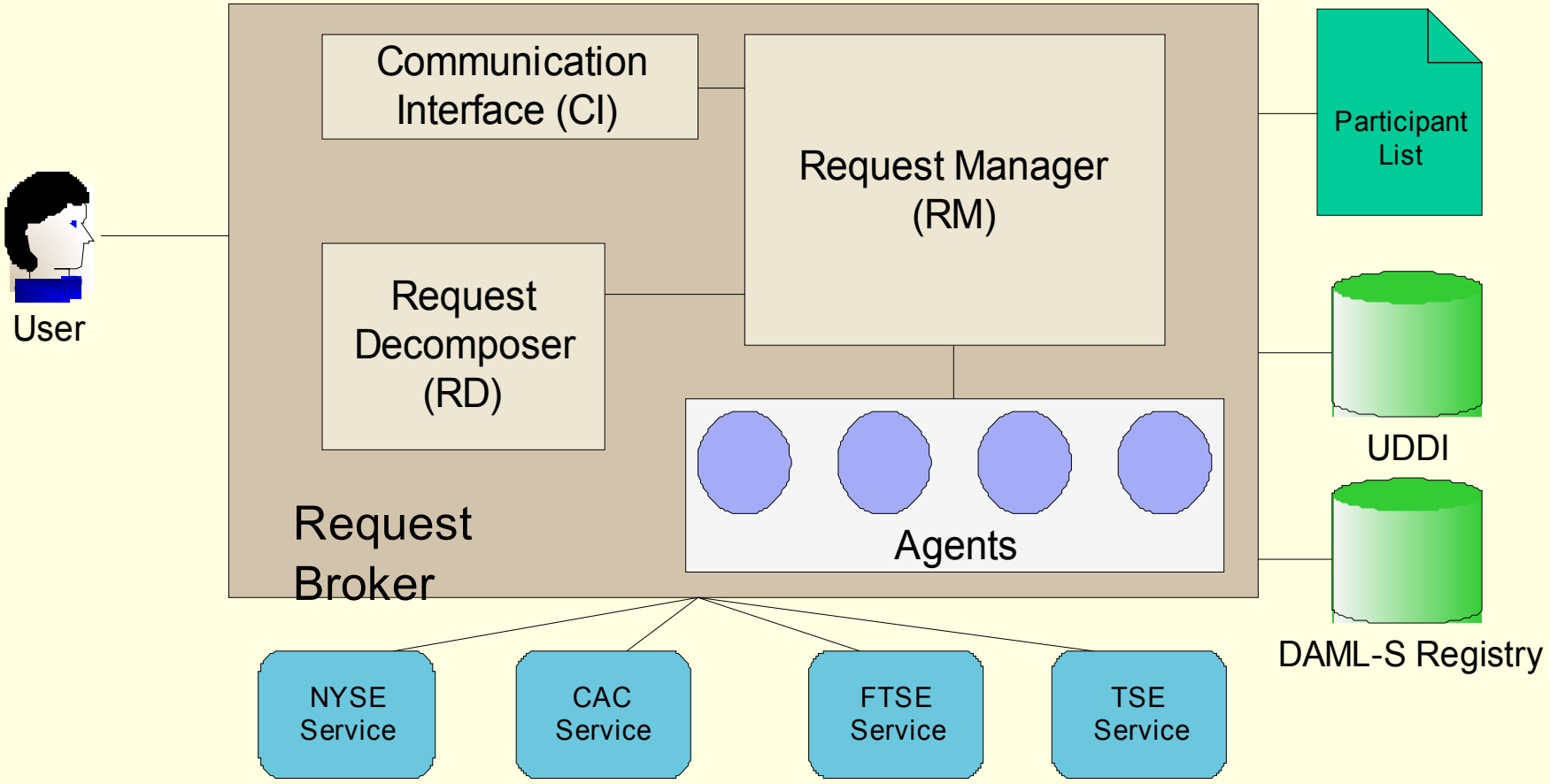
Reaction to Change

- Selection of alternate Web services using ontologies
- Cancellation of request if no alternate service is available
- Reaction to internal change is in the form of reconsolidating the result

Change Management Algorithm

```
Input: request_time, participant_list
{
  time = request_time
  while (time != 0)
    for each Web Service WS in participant_list
      send alive message to WS
      if not alive then
        remove WS from participant_list
        call (Service Selection (service description (WS)))
      break
      global description = WS service description from global service registry
      if service description (WS) not equals global description
        remove WS from participant_list
        call (Service Selection (service description (WS)))
      break
      current data = invoke WS operation
      if current data not equals previous data
        call (Response Consolidation (current Data))
        break
    decrement time
}
```

Architecture



Related Work

- **WebBIS**
  - Proposes mechanisms for *detection*, *propagation*, and *reaction* to change in e-services
  - Uses *Event-Condition-Action* (ECA) rules and *change operations* for change management
  - Our work extends WebBIS to provide support for Web service standards
- **XLANG**
  - Implements exception handling and transaction rollback by initiating *compensation* processes
  - Does not provide support for detection, propagation, and reaction to changes
- **eFlow**
  - Uses the notion of *process library* to compose services
  - Changes need to be predefined *manually* at the time of Web service composition

Conclusion

- We achieved success in fulfilling a service request using:
  - **Dynamic selection** of Web services through the use of *ontologies*
  - **Change management** in Web service environments by using *agents* to detect, propagate, and react to changes

Questions?