MARIO: Middleware for Assembly and Deployment of Multi-platform Flow-Based Applications

Eric Bouillet, Mark Feblowitz, Hanhua Feng,
Anand Ranganathan, Anton Riabov, Octavian Udrea
IBM TJ Watson Research Center, Hawthorne, NY

Zhen Liu
Nokia Research Center, Beijing

Middleware 2009, Urbana-Champaign

System S / Infosphere Streams

→ continuous ingestion

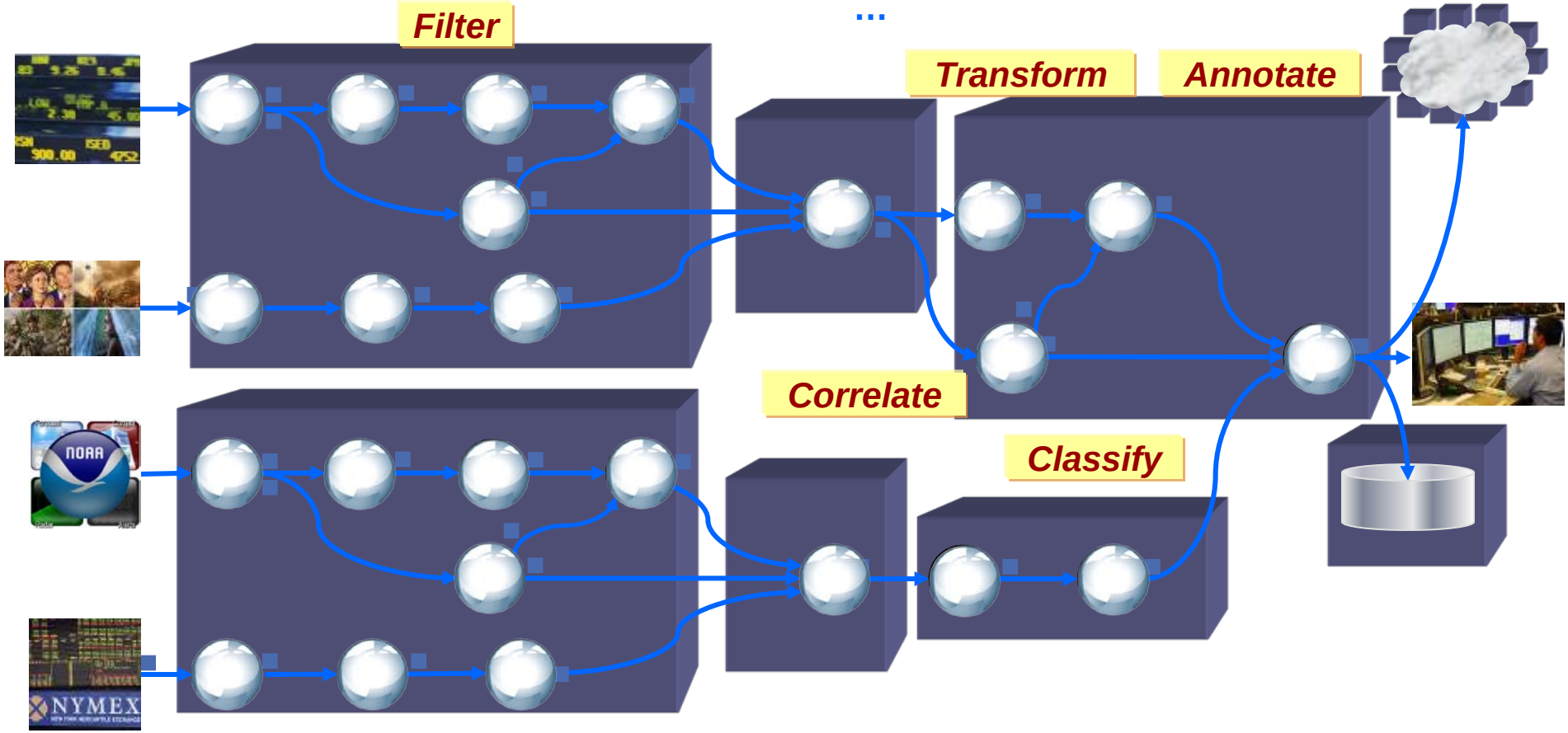
→ continuous analysis

infrastructure provides services for

scheduling analytics across h/w nodes

establishing streaming connectivity

…



achieve scale

by partitioning applications into components

by distributing across stream-connected hardware nodes

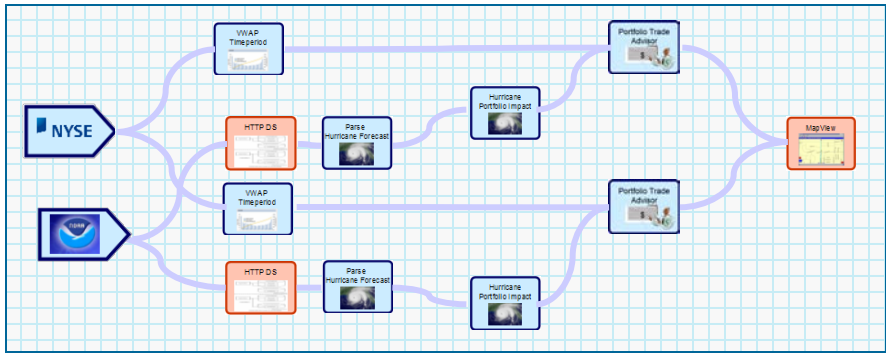
where appropriate,

elements can be “fused” together

for lower communication latencies

However, very few solutions involve just using stream processing

- **Need to interoperate with other systems**
  - SOA and Web Services
  - Other Event-Processing Systems
  - ETL (Extract-Transform-Load) systems
  - Mashups
  - Grid



- **Luckily, many of these systems are flow-based as well**

- **Key Idea :**
  - Can we have a single flow that spans all these platforms?
  - Easier to construct the flows and manage the interactions

2 key challenges in use of flows

- **Assembly**
  - How do we put together a valid flow from a set of available components?
- **Deployment**
  - How do we orchestrate and schedule the flow across one or more platforms?

Assembly and Deployment have been tackled for Single Platform Flows

- **Assembly**
  - Text (code) – BPEL, SPADE, GSFL
  - Various drag and drop assembly tools
    - Yahoo Pipes, IBM DAMIA, IBM Websphere Message Broker Toolkit, IBM Data Stage
  - Automatic Assembly
    - Semantic Web Service Composition
- **Deployment**
  - SODA for System S Stream Processing System
  - Various Grid Workflow scheduling systems
  - Various Web Service Orchestration Systems

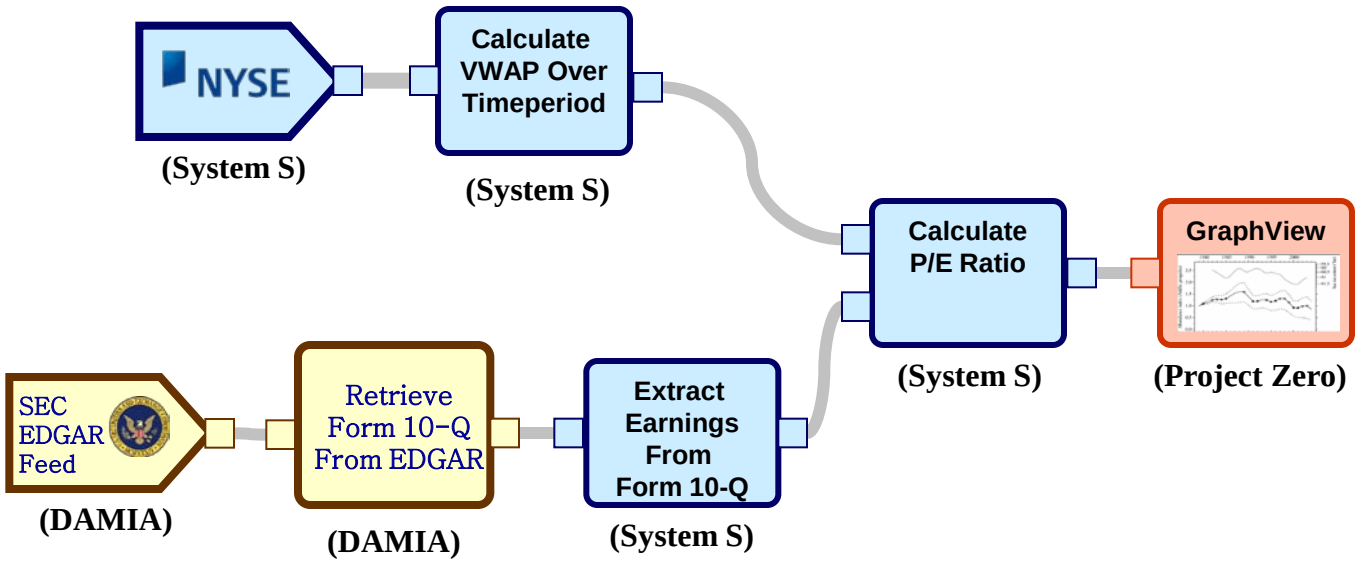
- **However, the problem has not been tackled for Multi Platform Flows**

Multi Platform Flows

- **Organizations have components spread across different platforms**
  - Use different models or protocols for information exchange and processing.
    - Push/Pull, Batch/Incremental
    - CORBA, HTTP, SOAP, REST, …
- **Many solutions require composing flows that span multiple platforms**
  - Take advantage of specialized components existing in different platforms
  - Take advantage of specific platform features for information processing

- **However, assembly and deployment of multi-platform flows are complex**
  - How to split processing across the different platforms?
  - How to make the different platforms inter-operate?

Example of multi-platform flow – calculate real-time P/E ratio for a company



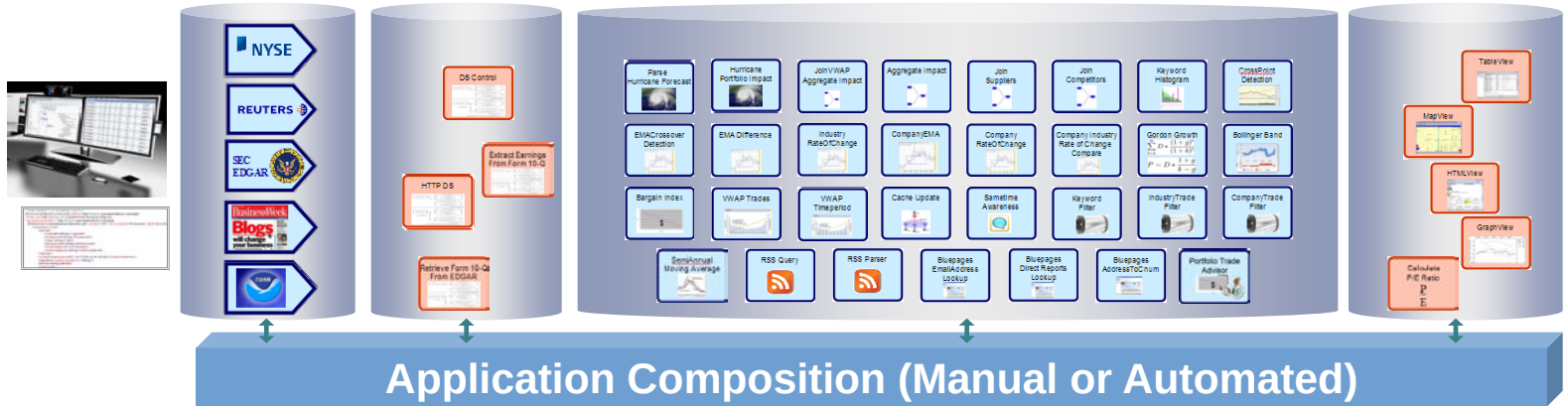
- **IBM DAMIA**
  - Assemble, aggregate, transform data feeds from the Internet and enterprise data sources,
  - Request-response model
- **System S**
  - Platform to ingest, filter, analyze, and correlate massive volumes of continuous data streams
  - Event-driven model
- **Project Zero**
  - Create, assemble and execute web 2.0 applications based on SOA.
  - Request-response model

Multi-Platform Assembly and Deployment with MARIO

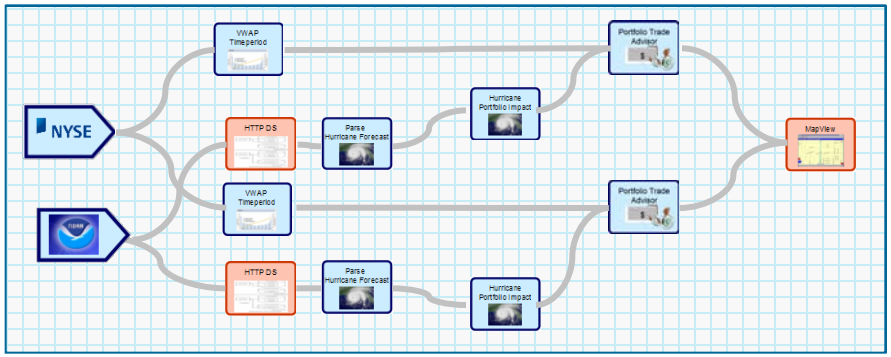
- **Assembly is platform-independent**
  - End-users can assemble multi-platform flows
    - Don’t have to worry about the details of the underlying platforms
- **Deployment is platform-dependent**
  - Middleware decides how to deploy the multi-platform flow.

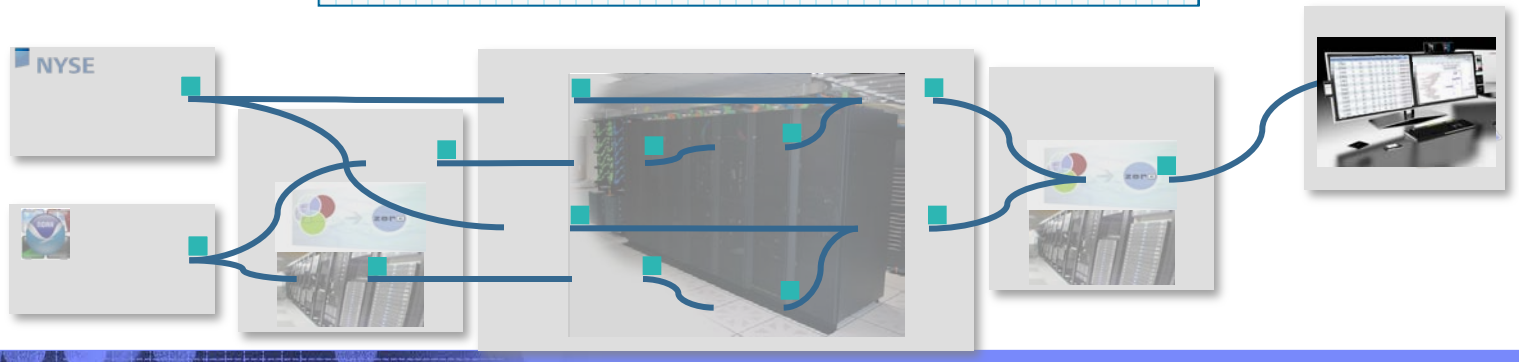
MARIO = Mashup Automation with Runtime Invocation and Orchestration

MARIO Mashup Automation with Runtime Orchestration & Invocation



- **Ease of use**
- **Shorter Time to Deployment**
- **Reusable Software Components**

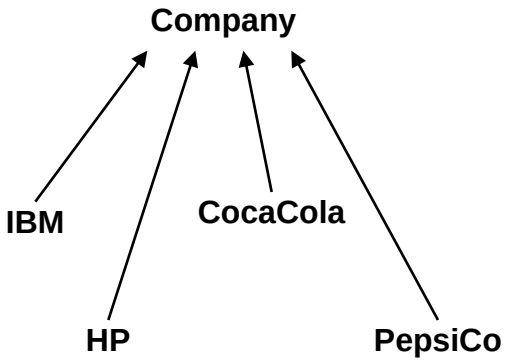


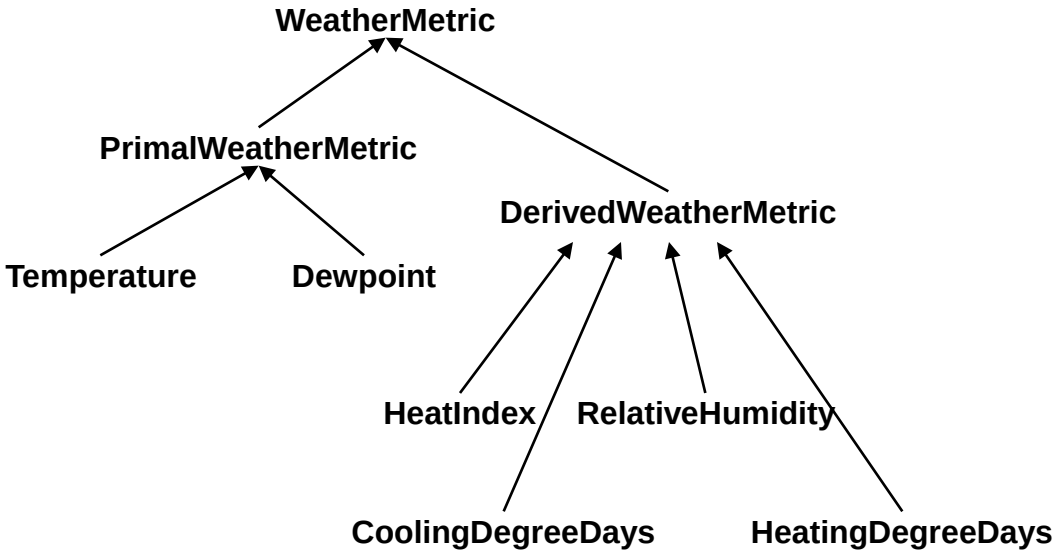


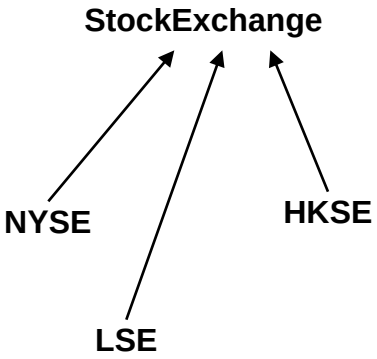
Common platform-independent Component model

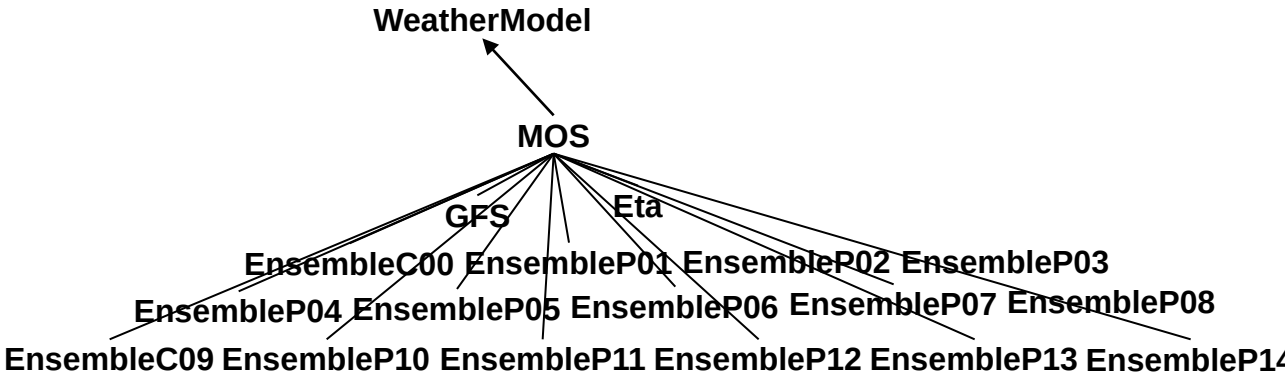
- **Includes both assembly and deployment instructions.**
  - The assembly instructions are general and platform-independent
  - Deployment instructions are platform-specific
- **Assembly instructions give semantic constraints on when components can be connected**
  - In the form of semantic, tag-based constraints on the inputs and outputs of the component.
- **Deployment instructions describe**
  - how to instantiate or invoke the component on a certain platform,
  - how to configure (or parameterize) it for a given flow
  - how to handle the inputs to and outputs from the component.
  - Written in a platform-specific scripting or instruction language (e.g. BPEL, SPADE, etc).

Tag Hierarchy (Directed Acyclic Graph)

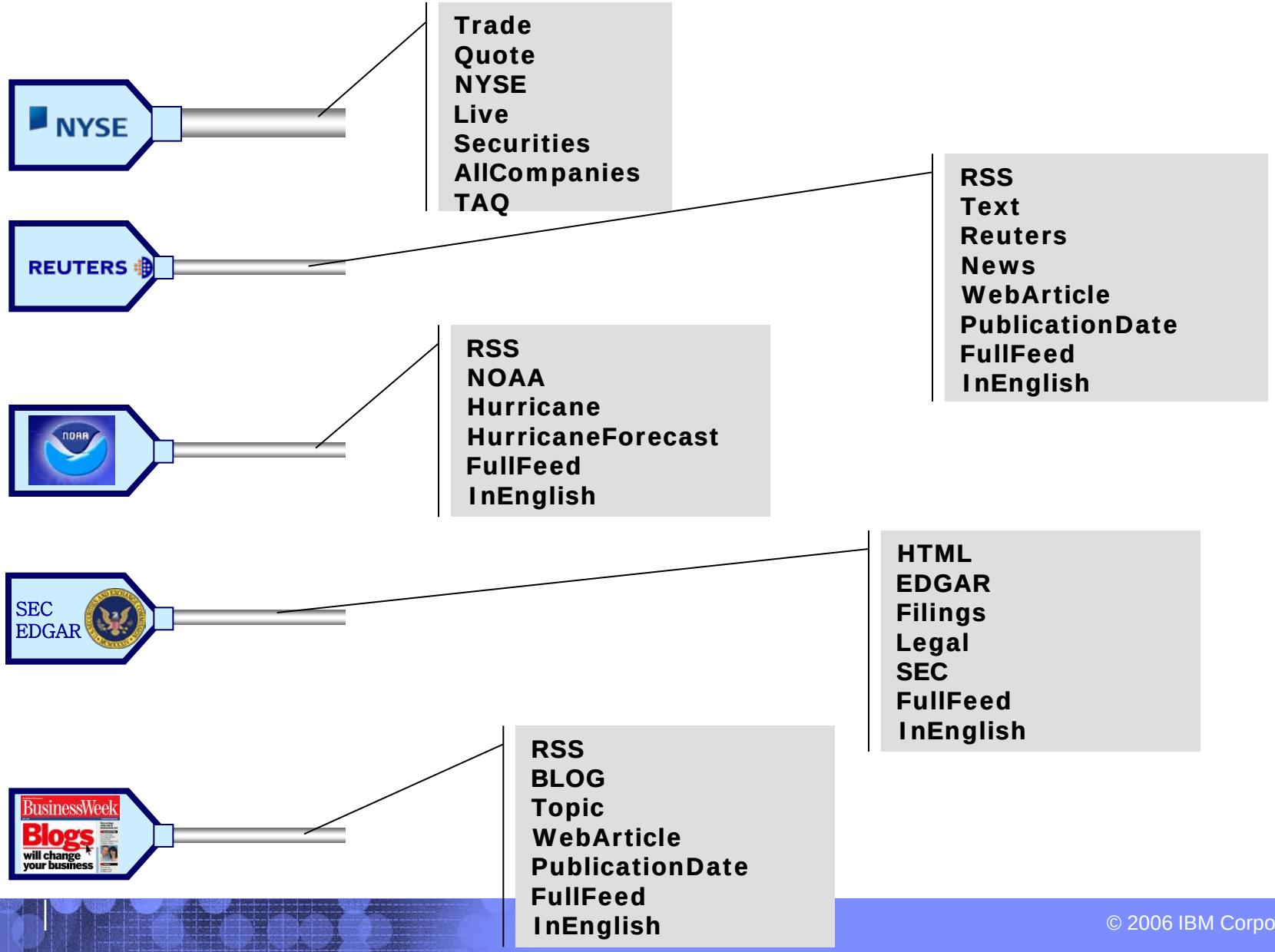




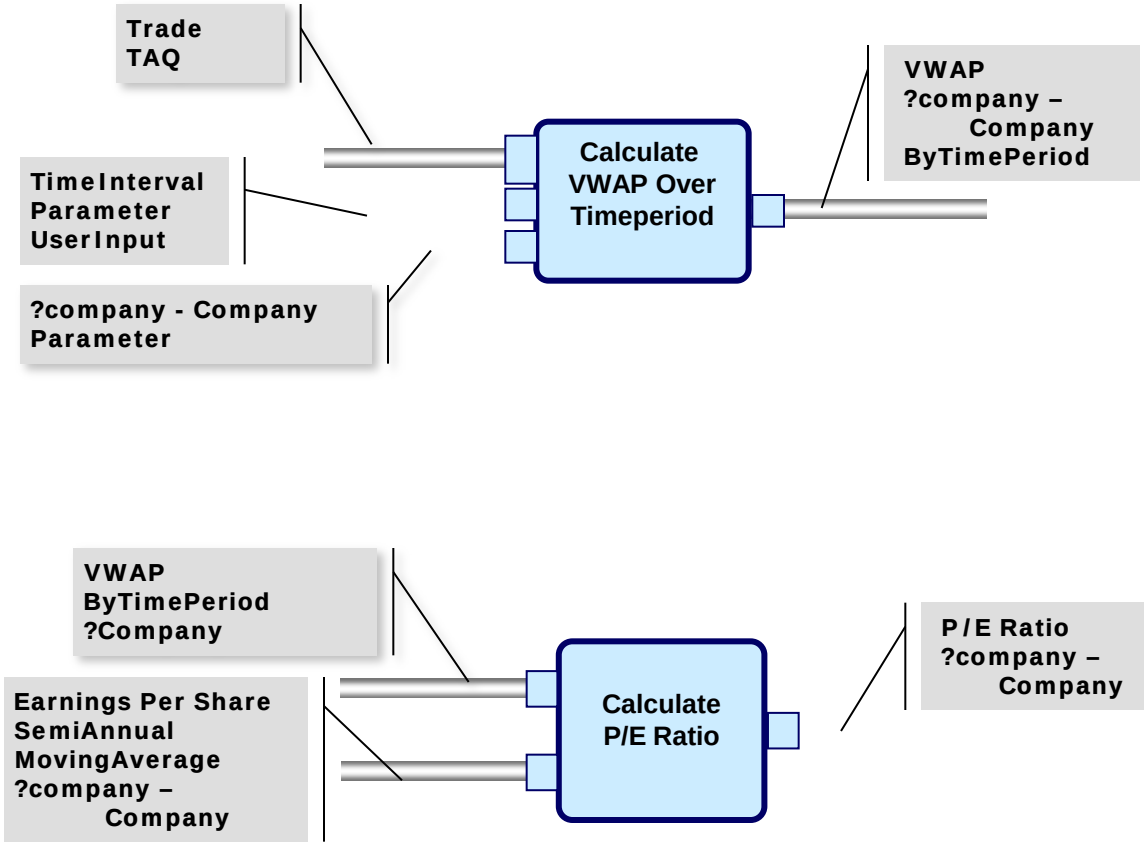


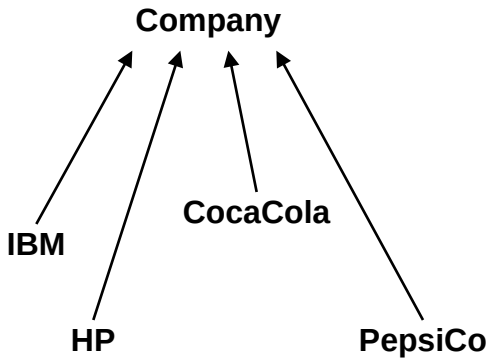


Data Sources – Tagged for Useful Content

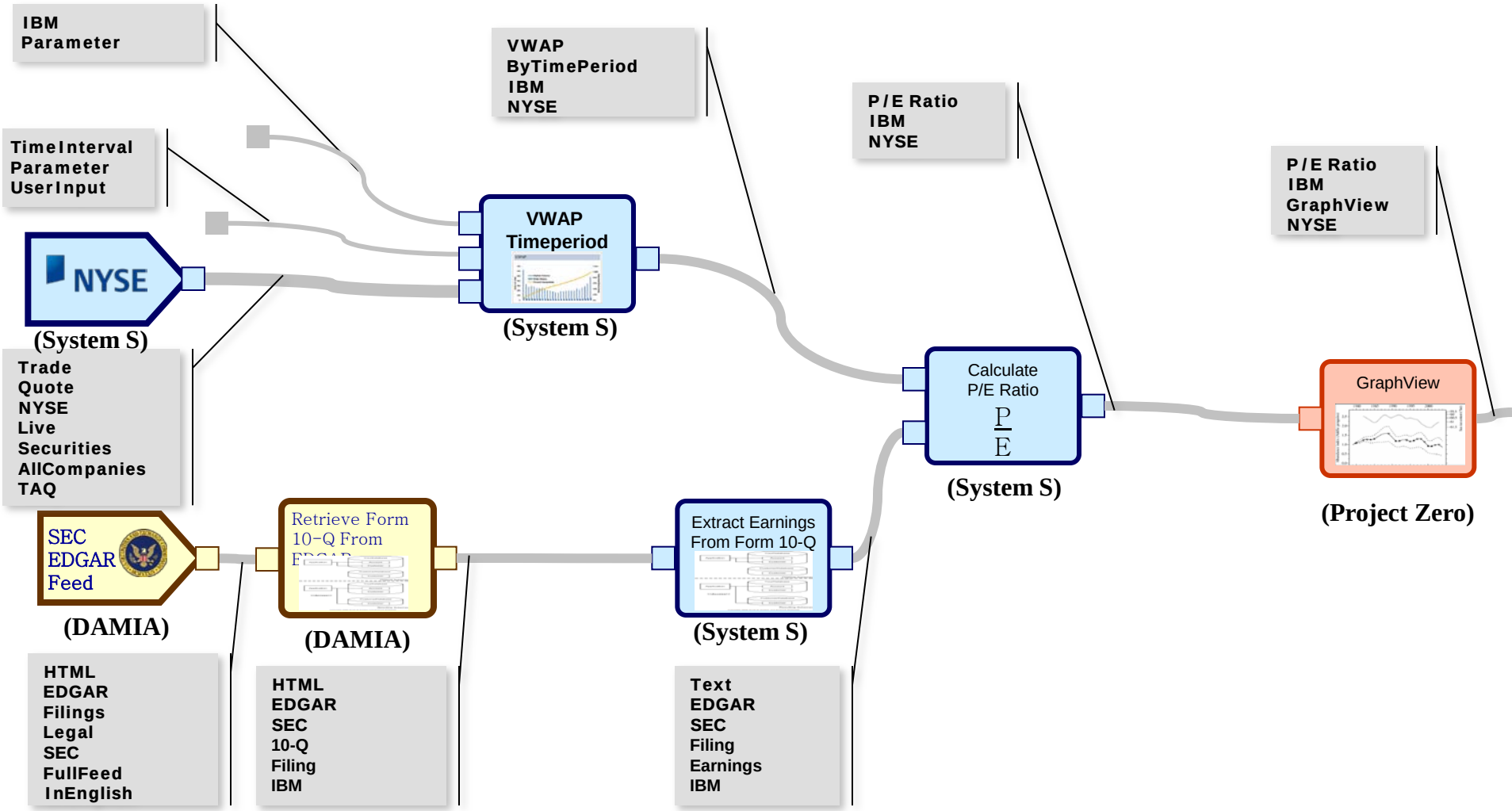


Model of components

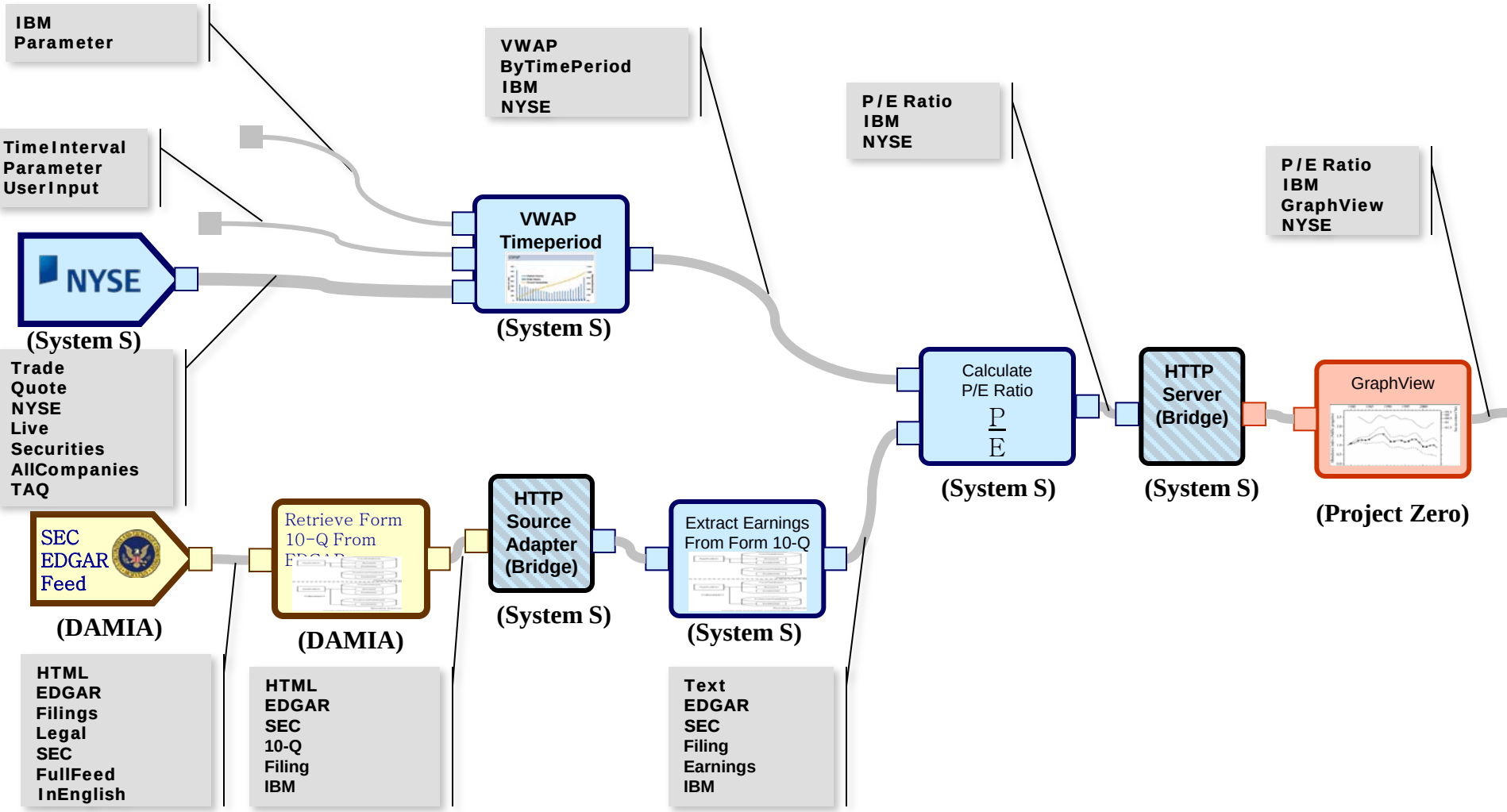




Assembly – manual or automatic – to get real-time P/E ratio of IBM



Need to insert bridging component



Deployment model – executable code and flow code

- **Each component is associated with executable code**
  - E.g. written in a language like C++ or Java that describes how it processes the input data to produce output data.
- **Also, each component is associated with deployment instructions in a platform-specific flow language**
  - Describe how this executable code can be instantiated or invoked on the platform as part of a larger flow.

- **In System S,**
  - Component executable code in C++ or Java.
  - Deployment instructions in SPADE
    - describes how to invoke executable code and how it is connected together in the flow.
- **For web service workflows**
  - Each component (web service) can be implemented in different ways
  - Deployment instructions in BPEL fragment
    - Describes invocation of a service with a certain input message to produce an output message.

Component Description Language

```
<?xml version="1.0" encoding="UTF-8"?>
<component  name="P_by_E_ratio">
<title>Calculate P/E ratio from real-time price and last earnings</title>

<!--Assembly Instructions-->
<var name="?company" type="Company"/>
<var name="?stockExchange" type="StockExchange"/>
<input name="PriceInputFromExchange"  tags="ByTimePeriod ?company ?stockExchange VWAP"/>
<input name="EarningsInput"  tags="EarningsPerShare SemiAnnual MovingAverage ?company"/>
<output name="P_by_E_Output" tags="P/E Ratio ?company ?stockExchange"/>

<!--Deployment Instructions in the SPADE language for System S-->
<binding type="system_s"> <![CDATA[
stream @P_by_E_Output@(P_E_ratio : Float)
    := Join(@PriceInputFromExchange@ <count(1)>; @EarningsInput@ <count(0)>) [true]
           {@PriceInputFromExchange@.price / @EarningsInput@.earnings }
]]></binding></component>
```

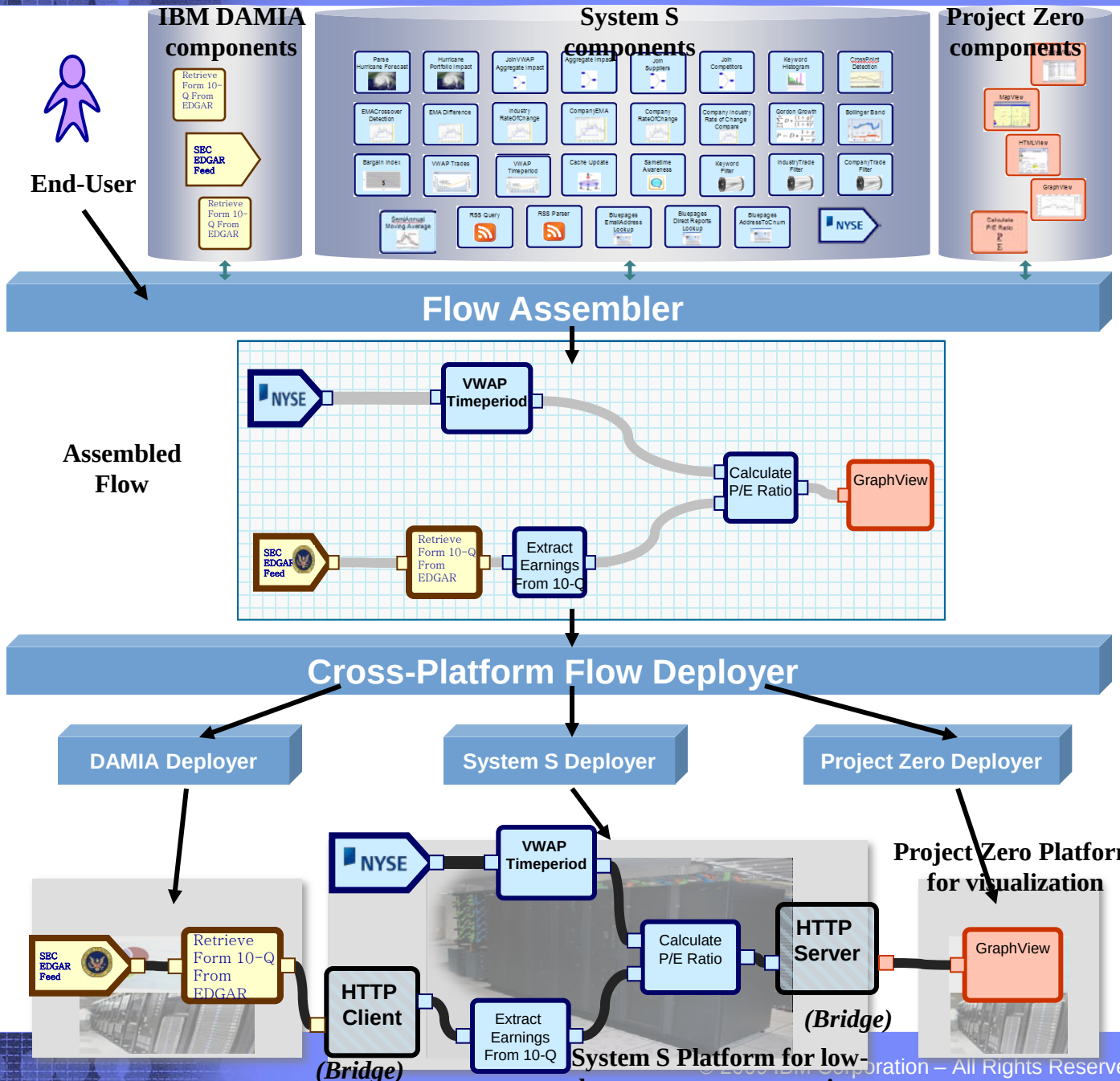
Input and Output tags

Assembly

Deployment

SPADE fragment indicating input streams, output streams and parameters in an operator invocation

Architecture



Flow Lifecycle

- **Assembly**
  - Manual or automated
  - Possibly using tags
- **Multi-Platform Flow Partitioning**
  - Cross-Platform Flow Deployer partitions the complete assembled multi-platform flow into platform-specific sub-flows.
- **Platform Specific Deployment**
  - Platform-specific deployers translate a sub-flow into a platform-specific flow-script,
    - Makes use of the code fragments in deployment section of the component description.
  - Deployment performed in flow order
    - Pass dynamically generated output references from one platform to another.
- **Insertion of Bridging Components**
  - Instantiation of additional components in the sending sub-flow and/or the receiving sub-flow.
  - Buffering strategies (e.g. streaming platform to a request-response platform)
  - Polling strategies (e.g. request-response platform to a streaming platform).

More on MARIO

- **MARIO has an OSGi plug-in architecture**
  - Each platform specific deployer is a plugin
  - Each kind of bridge is a plugin
    - Can potentially be used to bridge between different pairs of platforms
- **Pairs of platforms can be associated with a bridge**
  - The bridge can be configured based on the requirements of the two platforms
- **MARIO is not involved in moving data during actual flow execution**
  - It is only involved in setting up the platform-specific subflows and the bridges

A Financial Services Case Study

- **163 distinct components**
  - System S, DAMIA, Project Zero
- **Flow sizes range from 5-150 components**
- **Over 100,000 possible flows (based on tag constraints)**
- **Development and annotation done by a team of 5 people**

- **2 kinds of bridges**
  - *DAMIA to System S*
    - consists of a HTTP client on System S ; accesses a RSS feed URL exposed by a DAMIA sub-flow.
  - *System S to Project Zero*
    - a lightweight HTTP server on System S ; aggregates output from the System S sub-flow into an RSS feed that can be accessed by the Project Zero sub-flow.

- **Name server used by sub-flows to discover one another**

Challenges

- **Accurate Component Descriptions**
  - Kept in sync as code evolves
- **Platform Bridging**
  - Tune size of buffer, frequency of polling in push – pull bridging
  - Potentially large number of bridges
- **Application Design**
  - designing a set of modular, reusable components that can be assembled into different flows
- **Use of a common, general, data model and uniform interfaces**
  - better to have components with very loose type constraints and use self-describing data models (XML, tuples,…)
  - better to use simple data structures (that can be easily bridged)
  - Reflection to know what to do with an incoming data item

Conclusion

- **MARIO : middleware that supports the assembly and deployment of information processing flows that can potentially span multiple platforms.**
- **Addresses a key problem in many organizations, where the proliferation of legacy and new systems makes it difficult for end-users to create multi-platform applications.**
- **Component model that includes both assembly and deployment information**
- **MARIO partitions a multi-platform flow into single-platform sub-flows, deploys them individually and bridges them.**

Questions?

- **Papa Del's Pizza – the Champagne of Pizzas**
- **If you would like to obtain Infosphere Streams, see me ….**