Utilizing Commercial Object Libraries within Loosely-Coupled, Event-Based Systems

Jie Ren, Richard Taylor
Institute for Software Research
University of California, Irvine

Outline

- Motivation
- Background
  - Event-based systems
  - Commercial object library: COM
- Accidental and essential Issues
- An integration framework
- Validating the framework
- Conclusion

Motivation

- Software Architecture, Components, and Connectors
- Many benefits of event-based connectors
- But the majority of software is still based on procedure calls
- Need to bridge
  - Integrate
  - Evolve
  - Explore

Event-based Systems

- Components send events to each other
- Connectors provide messaging infrastructure
- Benefits
  - Heterogeneous components
  - Loosely-coupling
  - Easy evolution
- Sample systems: SIENA, KnowNow, C2

A Commercial Object Library: COM

- The dominant communication mechanism: DCE RPC, CORBA, COM(+), RMI, Web Service
- COM's basic concepts
  - Interface
  - Class
  - Object
  - Apartment

Issues in Integration

- Accidental and essential difficulties
- Accidental issues
  - Platforms: process, machine, OS, protocol
  - Programming Languages: JIntegra
- Essential issues: architectural difference
  - Lack of explicit reference
    - References of different forms: naming, binding
  - Asynchrony
    - Architectural and implementation asynchrony

Limitations of Built-in Integration

- COM's newer functionalities
  - Stubs for asynchronous calls
  - Event Service, MSMQ
- No dynamic events
- No event routing

A bridging framework

- COM-compatible interfaces describing concepts in event-based systems
- IEvent
- IComponent
- IConnector
- Classes implementing these interfaces

Key interface: IConnector

```
Interface IConnector {
   HRESULT HandleEvent(IEvent *);
   HRESULT Attach(IComponnet *);
   HRESULT Detach(IComponent *);
   HRESULT Attach(IConnector *);
   HRESUTL Detach(IConnector *);
   HRESULT Publish(IEvent *evt, IComponent *pub);
   HRESULT Subscribe(IEvent *pt, IComponent *sub);
}
```

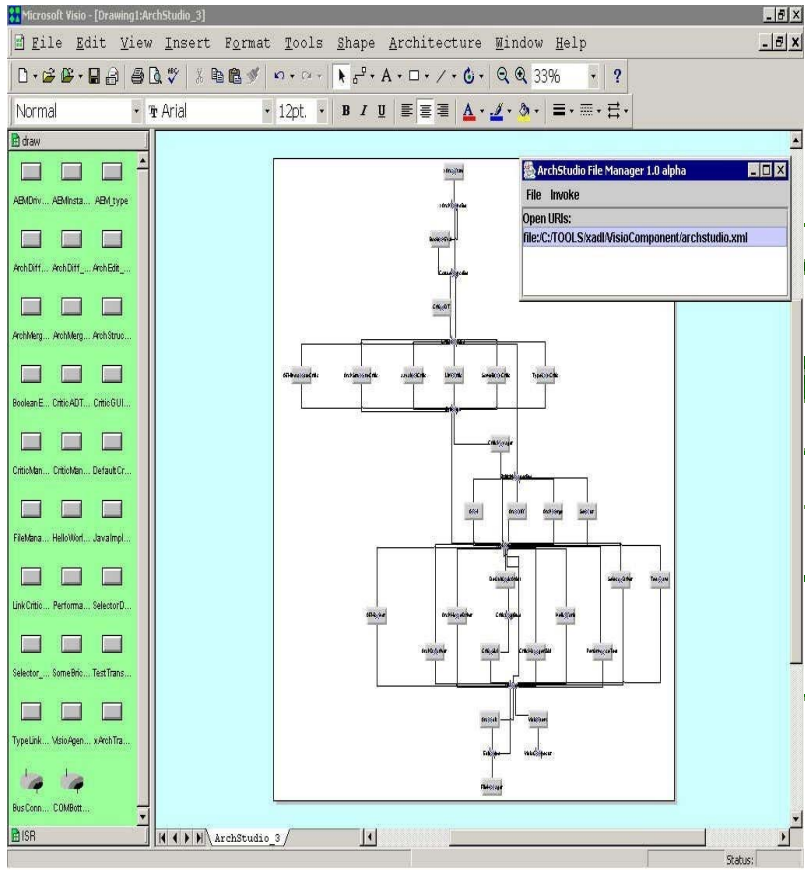
- Route events
- Configure components and connectors
- Publish and subscribe events

References and Asynchrony

- No explicit references between components
  - Only connectors know the neighboring components and connectors
- Sending event is non-blocking
  - `SendAndWait` is provided for convenience

Evaluation: Visio for ArchStudio

- Using the framework to integrate Microsoft Visio as the graphical frond-end of ArchStudio



Conclusion

- Integrating event-based systems and object libraries is important
- The challenge lies in the essential architectural differences: explicit reference and synchronous operation
- An initial bridging framework
- Future work: model and security