A Connector-Centric Approach to Architectural Access Control

Jie Ren

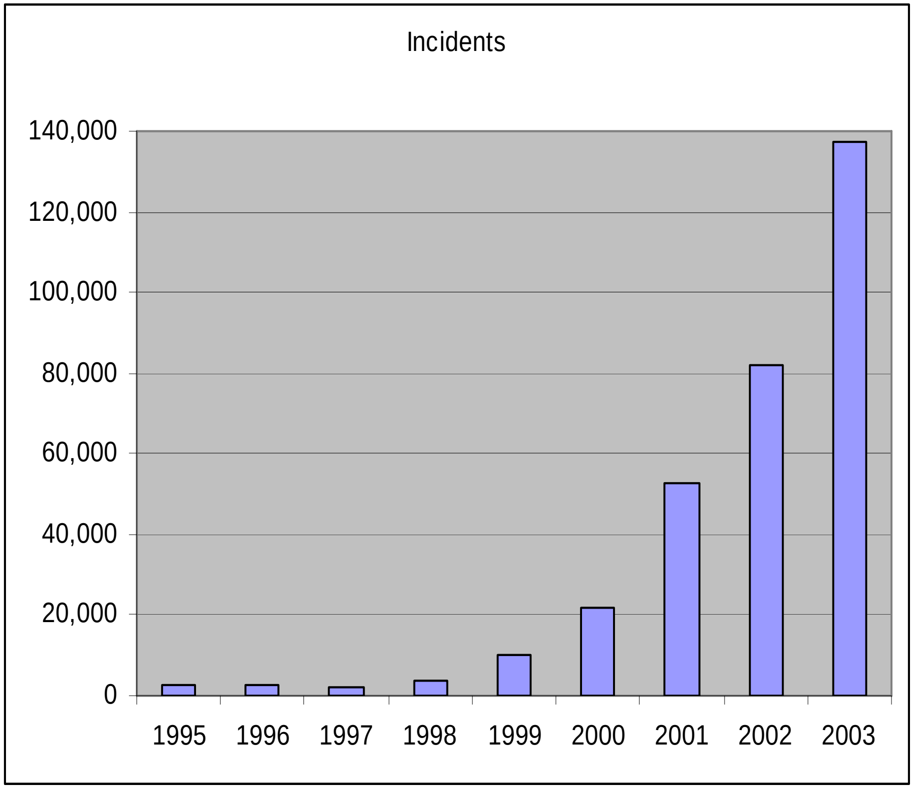
Department of Informatics

University of California, Irvine

Outline

- Overview
  - Architecture and Security
  - Software connectors
  - Hypotheses, approach, validation, contribution
- Architectural Access Control
  - Model: Subject, Principal, Resource, Privilege, Safeguard, Policy
  - Language: xADL, XACML, and Secure xADL
  - Contexts: neighborhood, type, container, architecture
  - Algorithm: interface access and privilege propagation
- Advanced concepts
  - RBAC, trust, content-based, architectural execution
- Tool support
- Case studies
- Conclusion

Security Incidents Reported to CERT



Re-architecting boosts security!

Table 1. Secure by design.

POTENTIAL PROBLEM	PROTECTION MECHANISM	DESIGN PRINCIPLES
The underlying `dll` (`ntdll.dll`) was not vulnerable because...	Code was made more conservative during the Security Push.	Check precondition
Even if it were vulnerable...	Internet Information Services (IIS) 6.0 is not running by default on Windows Server 2003.	Secure by default
Even if it were running...	IIS 6.0 does not have WebDAV enabled by default.	Secure by default
Even if Web-based Distributed Authoring and Versioning (WebDAV) had been enabled...	The maximum URL length in IIS 6.0 is 16 Kbytes by default (> 64 Kbytes needed for the exploit).	Tighten precondition, secure by default
Even if the buffer were large enough...	The process halts rather than executes malicious code due to buffer-overrun detection code inserted by the compiler.	Tighten postcondition, check precondition
Even if there were an exploitable buffer overrun...	It would have occurred in w3wp.exe, which is running as a network service (rather than as admininstrator).	Least privilege (Data courtesy of David Aucsmith.)

Wing, IEEE Security & Privacy, 2003

Problem

- Architectural Access Control:
  - How can we describe and check access control issues at the software architecture level?

Main Goal

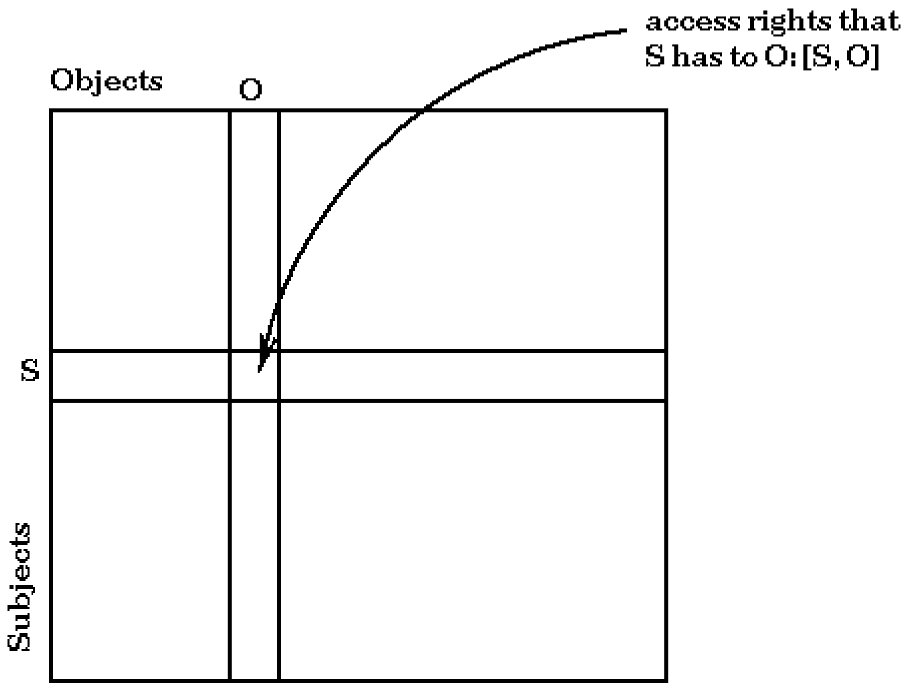
- Integrate security and software architecture
  - Integrate
  - Security: integrity through access control
  - Architecture level: abstraction
  - Software engineering perspective: how to express, check, and enforce

Security Overview

- Security
  - confidentiality, integrity, availability
- Security policy, model, mechanism
- Reference Monitor and Trusted Computing Base
  - Anderson 1972

Classic Discretionary Access Control

- Lampson 1971
- Subject
- Object
- Privilege



Component and Architecture Security

- Component-based Software Engineering
  - Computer Security Contract, Khan 2001
  - cTLA Contract, Herrmann 2003
- Software Architecture
  - ASTER, Bidan and Issarny 1997
  - System Architecture Model, Deng et al. 2003
  - SADL, Moriconi et al. 1997
  - Law-Governed Architecture, Minsky 1998
- Mostly cryptography, insufficient access control

Connectors

- Why connectors
  - Model the fundamental communication issue
- Should they be first class citizens?
  - Capture and reuse
- Existing work
  - Taxonomy: Mehta 2000
  - Assembly Language: Mehta 2004
  - Constructions: Lopes 2003
  - Transformation: Spitznagel 2001
- Shortcoming: insufficient access control
  - Dependability: Spitznagel 2004

Hypotheses

- Hypothesis 1: An architectural connector may serve as a suitable construct to model architectural access control
- Hypothesis 2: The connector-centric approach can be applied to different types of componentized and networked software systems
- Hypothesis 3: With connector propagating privileges, the access control check algorithm can check the suitability of accessing interfaces
- Hypothesis 4: In an event-based architecture style, connectors can route events in accordance with the secure delivery requirements

Approach

- A connector-centric approach to describe and enforce Architectural Access Control
  - Combine software architecture and security research
  - Adopt an integrated access control model: classic, role-based, trust management
  - Secure xADL, based on xADL and XACML
  - Architectural contexts
  - Architectural execution
  - Connector-centric description and enforcement
  - Tool support

Validation

- Algorithm analysis
  - Based on graph reachability
- Four case studies
  - Development of secure coalition
    - Connector for secure message delivery
  - Development of Impromptu
    - Composite connector among heterogeneous components
  - Modeling of Firefox component security
    - Algorithm to check critical path with the connector
  - Modeling of DCOM security
    - Connectors for networked components

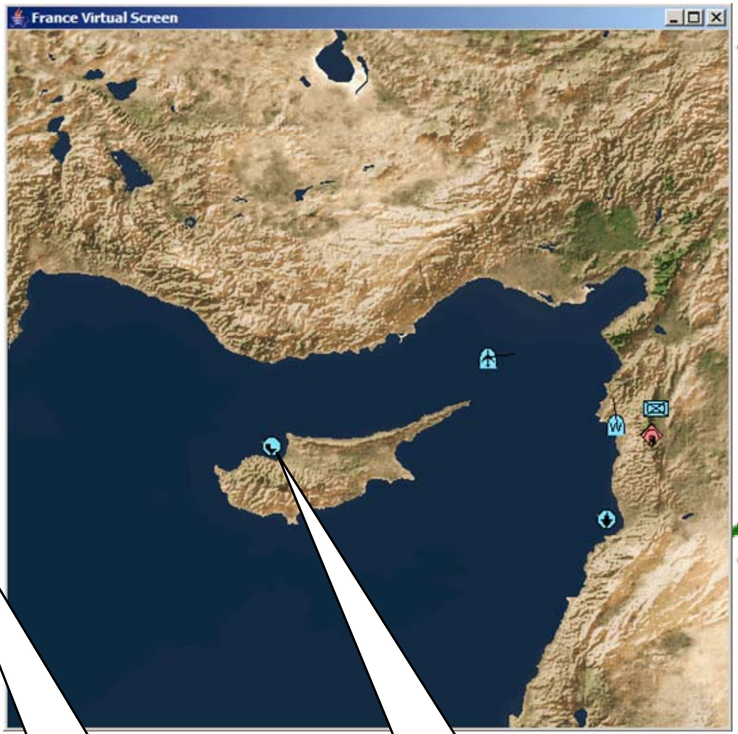
Contributions

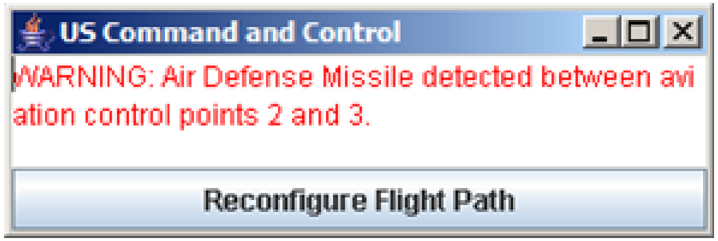
- A novel approach to the design and analysis of the access control property for software architectures
- A usable formalism for modeling and reasoning about architectural access control
- An algorithm for checking whether the architectural model maintains proper access control at design-time
- A suite of usable tools to design and analyze secure software

Architectural Access Control

- Basic concepts, applied in architecture
  - Subject, Principal, Resource, Permission/Privilege/Safeguard, Policy
- Secure xADL
  - xADL
  - XACML
  - Language design
- Contexts
  - Neighborhood, type, container, architecture
- Check algorithm
- Central role of connectors

Running Example: Coalition





Message from France

Message from US

Concepts: Subject

- A ***subject*** is the user on whose behalf software executes
- Missing from traditional software architecture:
  - All of its components and connectors execute under the same subject
  - The subject can be determined at design-time
  - It generally will not change during runtime, either inadvertently or intentionally
  - Even if there is a change, it has no impact on the software architecture

Concepts: Principal

- A subject can take multiple ***principals***, which encapsulate the credentials that a subject possesses to acquire permissions
- Different types of principals
- Summary credentials and concrete credentials
- Missing from previous architectures

Concepts: Resource

- A ***resource*** is an entity whose access should be protected
- Passive: files, sockets, etc.
- Active: components, connectors, interfaces
  - Relevant to architecture

Concepts: Privilege

- ***Permissions*** describe a possible operation on an object
- ***Privilege*** describes what permissions a component possesses depending on the executing subject
- Privilege escalation vulnerabilities
- Two types of privileges:
  - Traditional: read file, open sockets, etc.
  - Architectural: access, instantiation, connection, message routing, introspection, etc.

Concepts: Safeguard

- ***Safeguards*** are permissions that are required to access the interfaces of the protected components and connectors
- Architectural access control check

Concepts: Policy

- A ***policy*** specifies what privileges a subject, with a given set of principals, should have to access resources protected by safeguards
- Numerous existing studies in the security community
- We focus on software engineering applicability for architectural modeling

Overview of xADL

- XML-based extensible architecture description language
- Component and connector
- Types
- Signatures and interfaces
- Sub-architecture
- Design-time and run-time
- Tool support: ArchStudio
- Extensible: configuration, execution

Overview of XACML

- Conceptual framework for access control models
  - Based on set theory and first order logic
- Extensible
- Formal semantics
- Matching rule for request
  - Policy Enforcement Point (PEP) and Policy Decision Point (PDP)
  - PolicySet, Policy, Rule
  - Match on Subject, Resource, Action
- Combining algorithms
- Open Standard from OASIS

Secure xADL

- The first effort to model these security concepts directly in an architectural description language
- Viewed from XACML: a profile for the software architecture domain
- Viewed from xADL: a new schema with elements necessary for access control

Syntax of Secure xADL

```
<complexType name="SecurityPropertyType">
  <sequence>
    <element name="subject" type="Subject"/>
    <element name="principals" type="Principals"/>
    <element name="privileges" type="Privileges"/>
    <element name="policies" type="Policies"/>
  </sequence>
<complexType>
<complexType name="SecureConnectorType">
  <complexContent>
    <extension base="ConnectorType">
      <sequence>
        <element mame="security"
           type="SecurityPropertyType"/>
      </sequence>
    </extension>
<!-- similar constructs for component,
structure, and instance -->
```

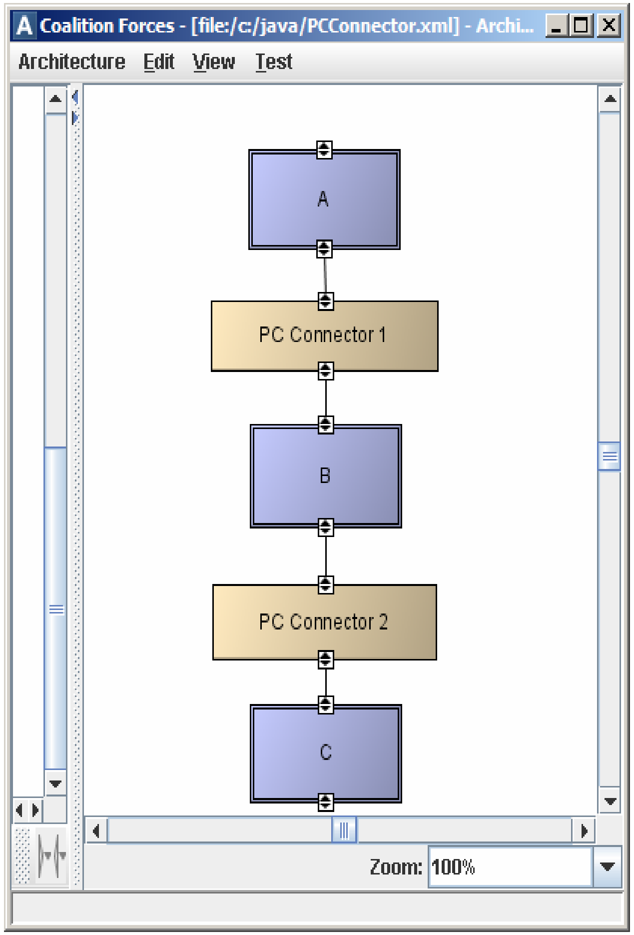
Rationales for Language Design

- Concepts
  - Architecture, access control
- Extensibility
  - xADL, XACML
- XACML flexible in combining policies
- Tool support
  - ArchStudio
  - Evaluation engine and editor

The Larger Contexts

- Access control decisions might be based on entities other than the decision maker and the protected resource. These relationships are the *contexts.*
- XACML's combining algorithms supply a framework to combine these contexts

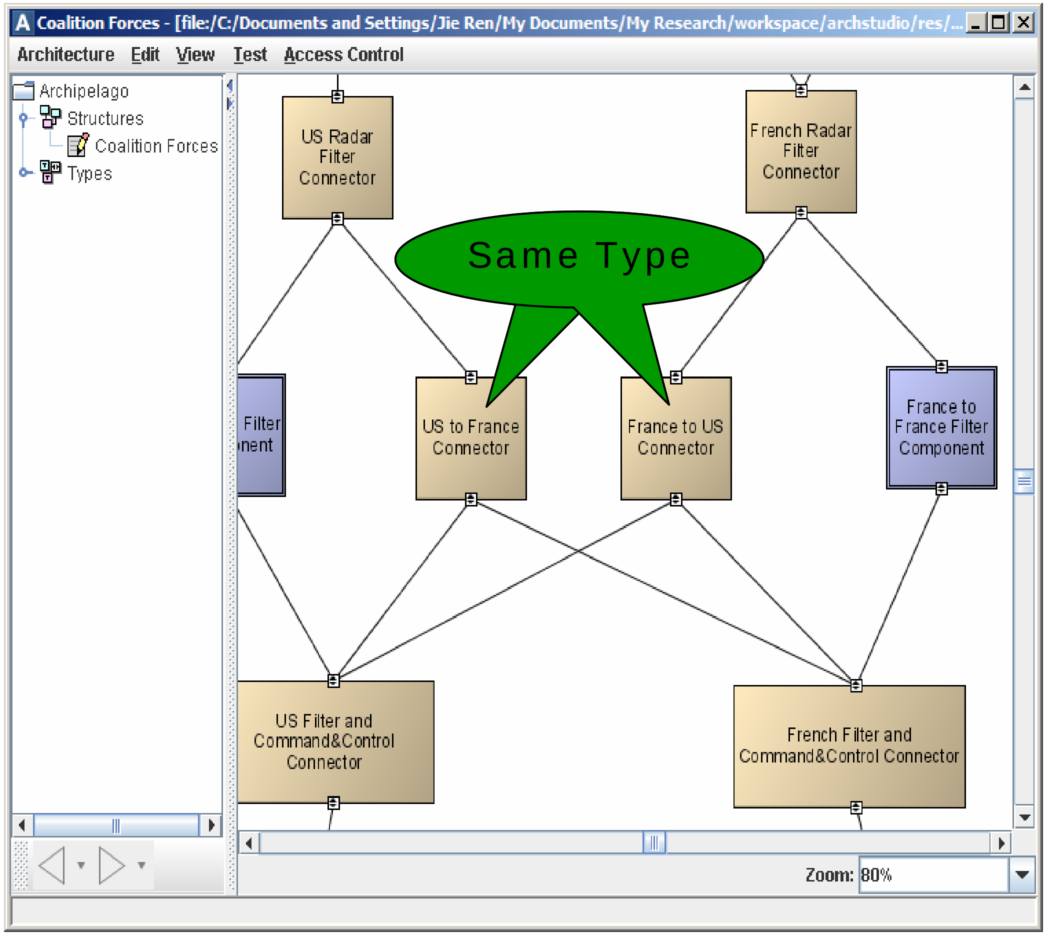
Neighborhood Context



Four Types of Contexts

1. The nearby components and connectors of the component and the connector
2. The type of the component and the connector
3. The explicitly modeled sub-architecture that contains the component and the connector
4. The global architecture

Coalition with Two Connectors



```
<connectorType id="SecureC2Connector_type" xsi:type="SecureConnectorType">
    <principal>NATO</principal>
      <PolicySet PolicySetId="InstantiateConnectorType"
                 PolicyCombiningAlgId="deny-overrides">
        <Policy RuleCombiningAlgId="deny-overrides">
          <Rule Effect="Deny">
            <SubjectMatch MatchId="string-equal">
              <AttributeValue>SecureManagedSystem
              <AttributeDesignator>subject-id
            <ActionMatch MatchId="string-equal">
              <AttributeValue>AddBrick<AttributeDesignator>action-id
            <Condition FunctionId="not">
              <Apply FunctionId="string-is-in">
                <AttributeValue>NATO</AttributeValue>
                <AttributeDesignator>principal
```

Type Policy

```
<connector id="UStoFranceConnector" xsi:type="SecureConnector">
    <principal>US</principal>
      <PolicySet PolicyCombiningAlgId="deny-overrides">
        <Policy RuleCombiningAlgId="deny-overrides">
          <Rule Effect="Deny">
            <SubjectMatch MatchId="string-equal">
              <AttributeValue>SecureManagedSystem
            <ActionMatch MatchId="string-equal">
              <AttributeValue>AddBrick<AttributeDesignator>action-id
            <Condition FunctionId="not">
              <Apply FunctionId="string-is-in">
                <AttributeValue>US</AttributeValue>
                <AttributeDesignator>principal
        <PolicySetIdReference>InstantiateConnectorType
```

Instance Policy

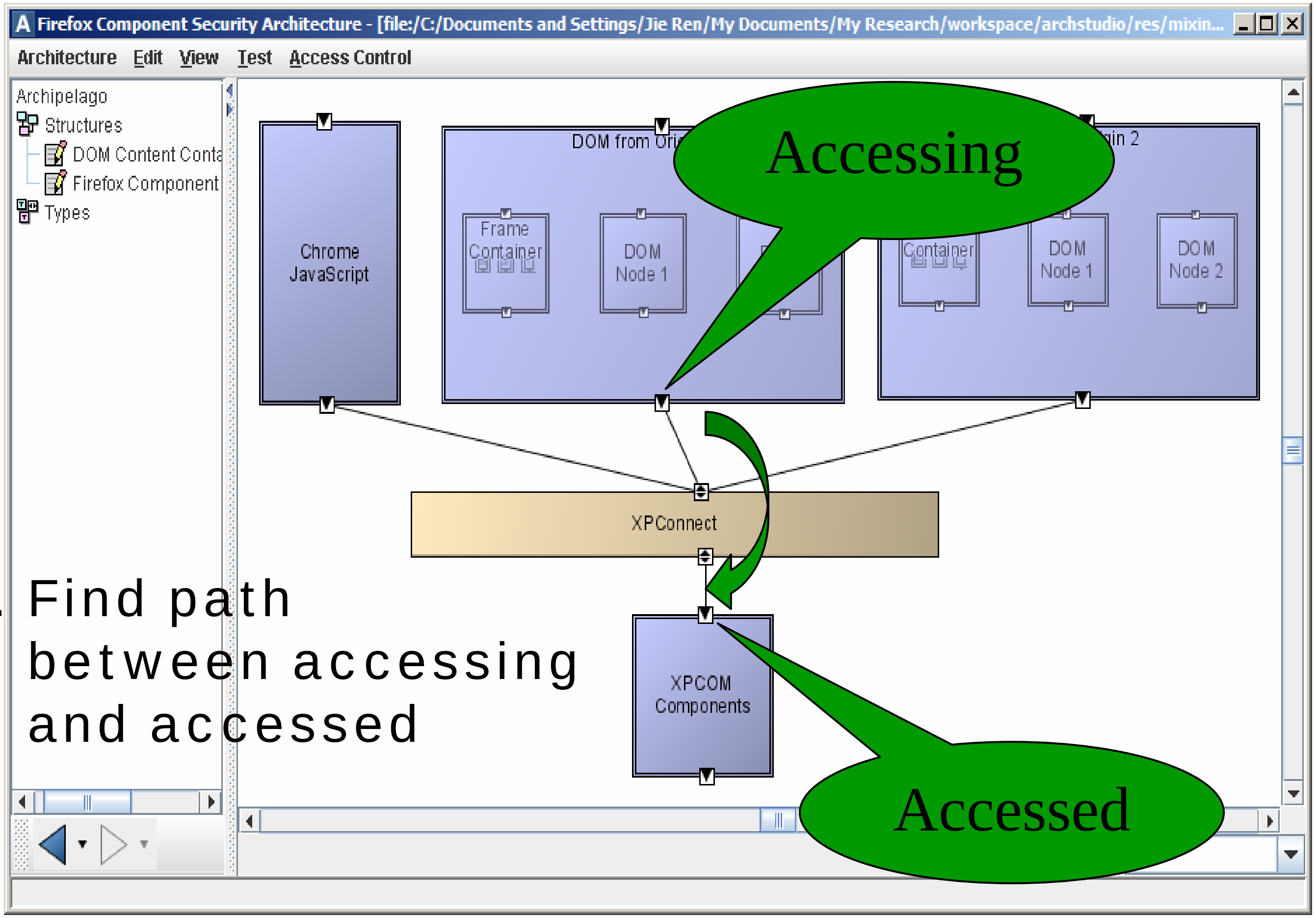
Algorithm to Check Architectural Access

- Given a secure software architecture description written in Secure xADL, if a component A wants to access another component B, should the access be allowed?
- Applying situations
  - Currently design-time, possibly run-time
  - Global, not local
  - Connector propagates privileges

Algorithm 1

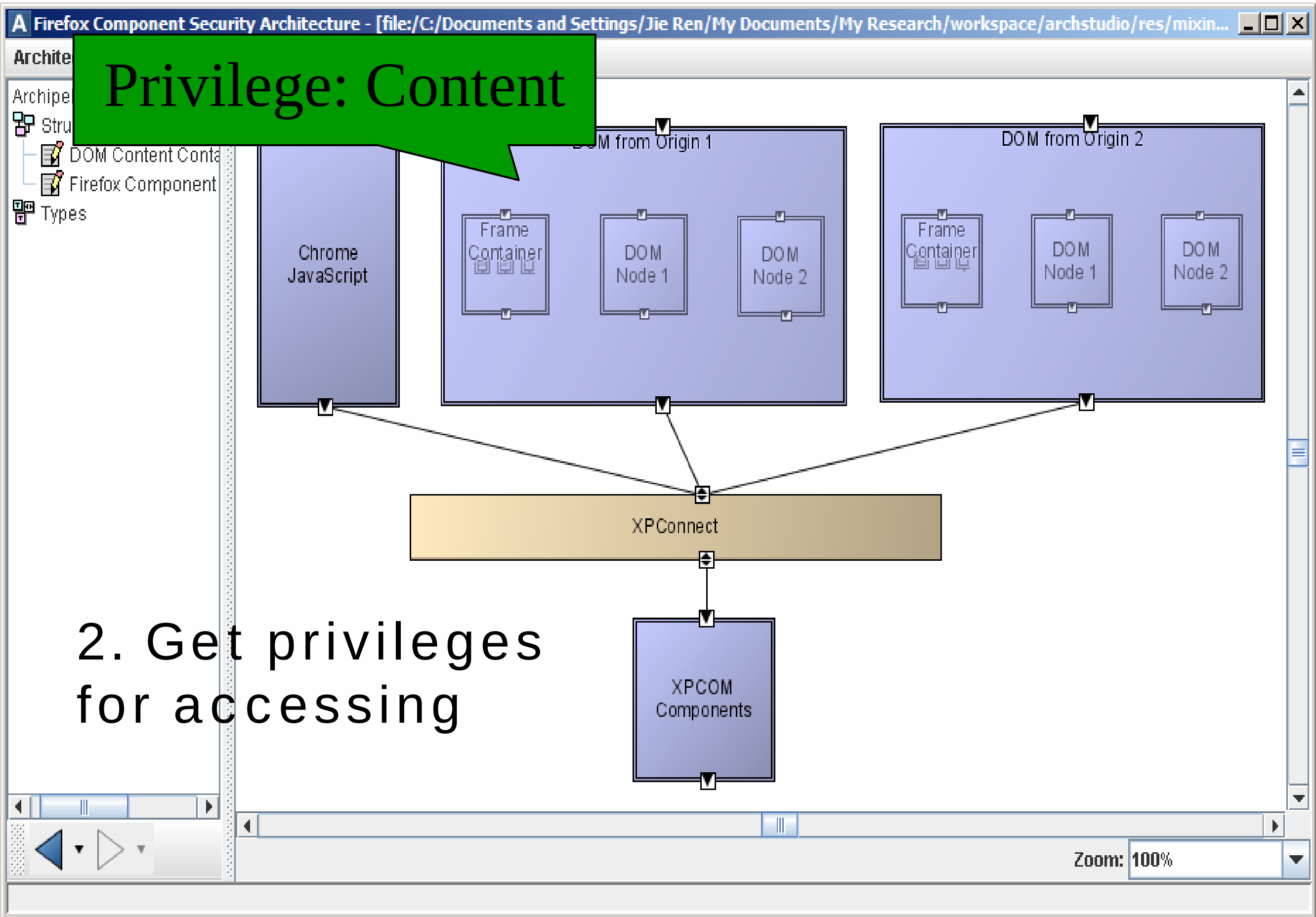
```
Input: an outgoing interface,
    Accessing, and an incoming
    interface, Accessed

Output: grant if the Accessing can
    access the Accessed, deny if
    the Accessing cannot access
    the Accessed

Begin
  if (there is no path between
    Accessing and Accessed)
    return deny;
  if (Accessing and Accessed are
      connected directly)
    DirectAccessing = Accessing;
  else
    DirectAccessing = the constituent
       nearest to Accessed in the path;
  Get AccumulatedPrivileges for
    DirectAccessing from the owning
    component, the type, the containing
    sub-architecture, the complete
    architecture, and the connected
    constituents;
  Get AccumulatedSafeguards for
    Accessed from the owning
    constituent, the type,
    the containing
    sub-architecture, and the
    complete architecture;
  Get AccumulatedPolicy for
    Accessed from
    similar sources;
  if (AccumulatedPolicy exists)
    if (AccumulatedPolicy
        grants access)
      return grant;
    else
      return deny;
  else
    if (AccumulatedPrivileges
         contains
        AccumulatedSafeguards)
      return grant;
    else
      return deny;
End
```

Applying Algorithm: Firefox



1. Find path between accessing and accessed

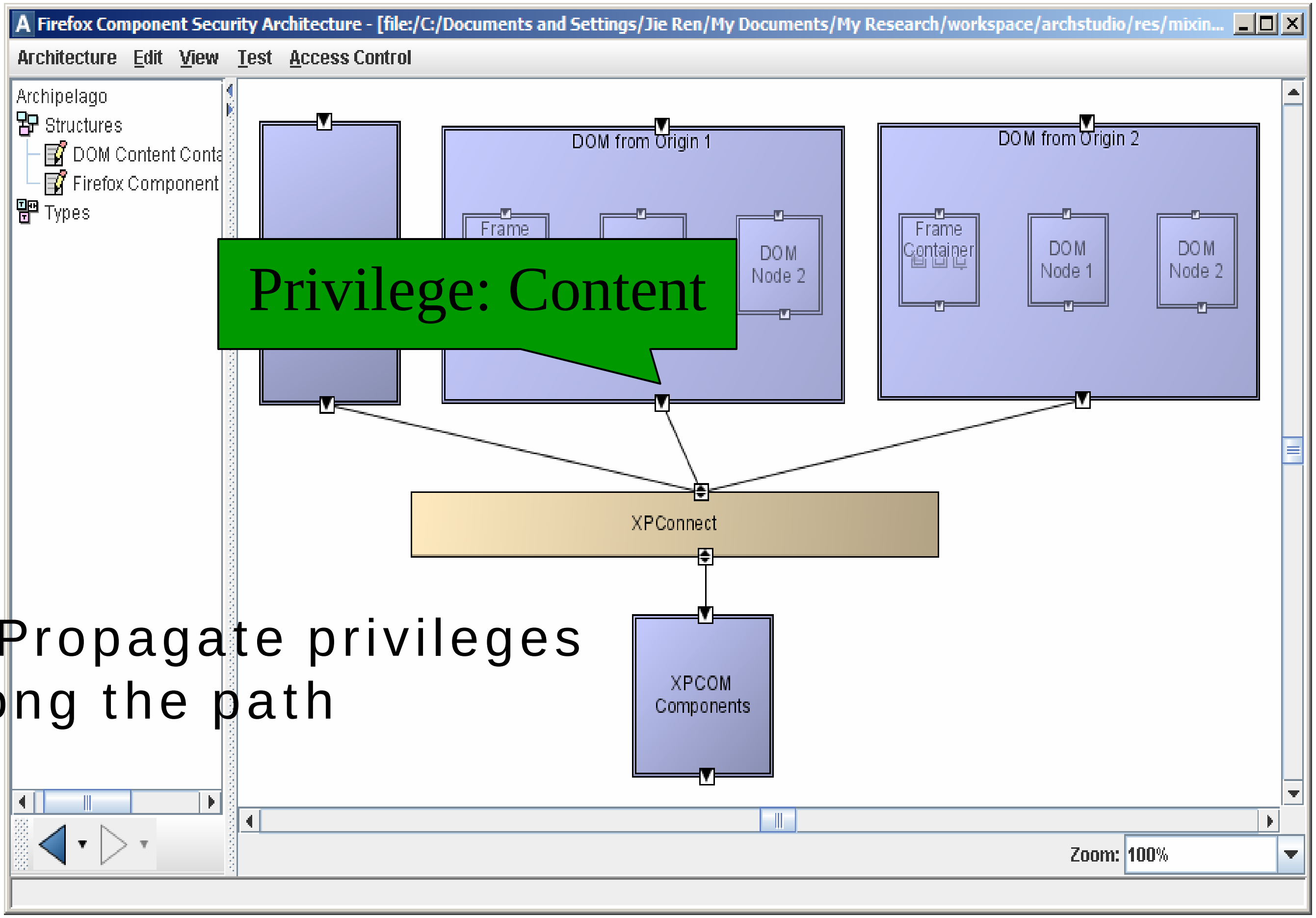
Applying Algorithm: Firefox



Privilege: Content

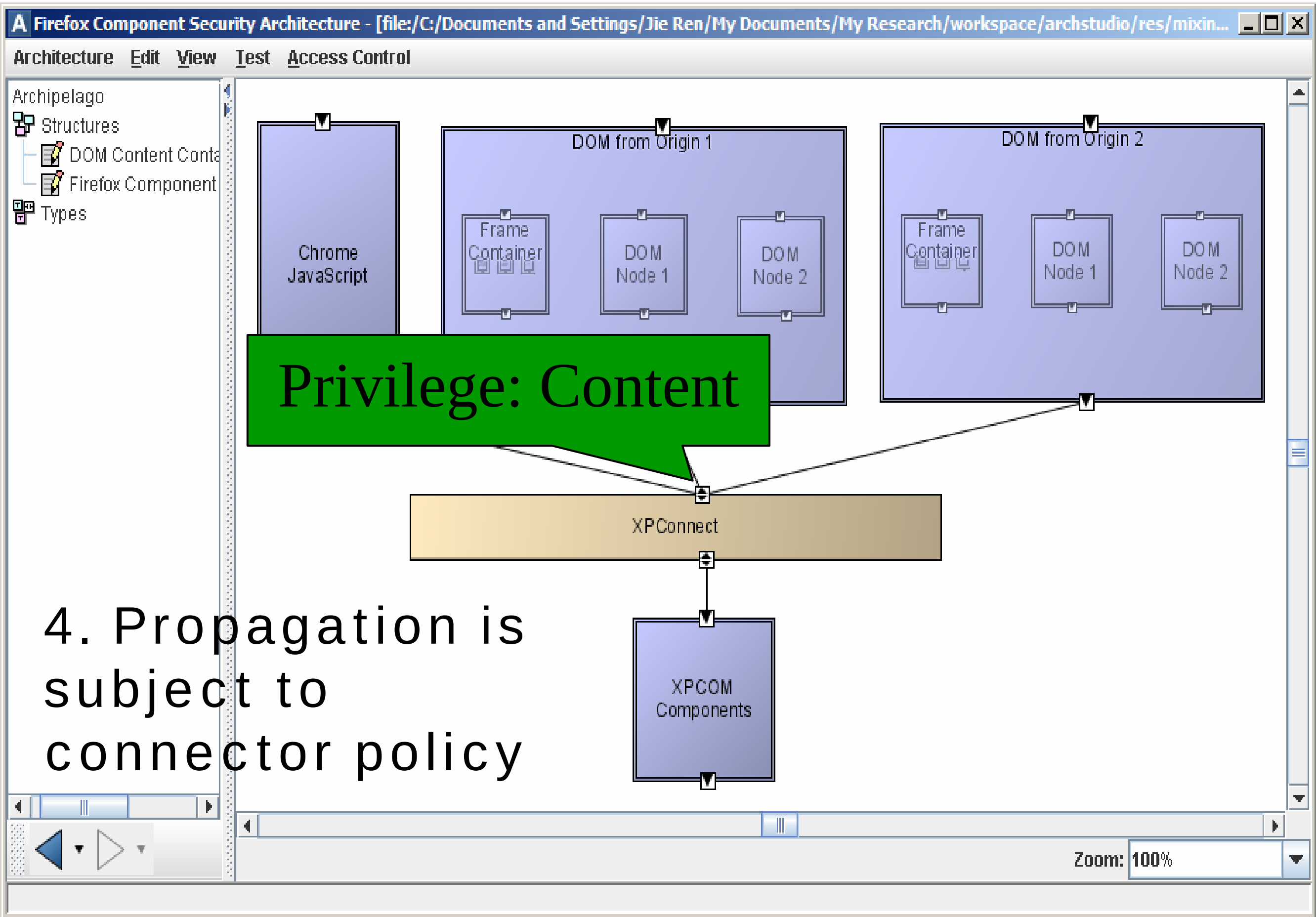
2. Get privileges for accessing

Applying Algorithm: Firefox



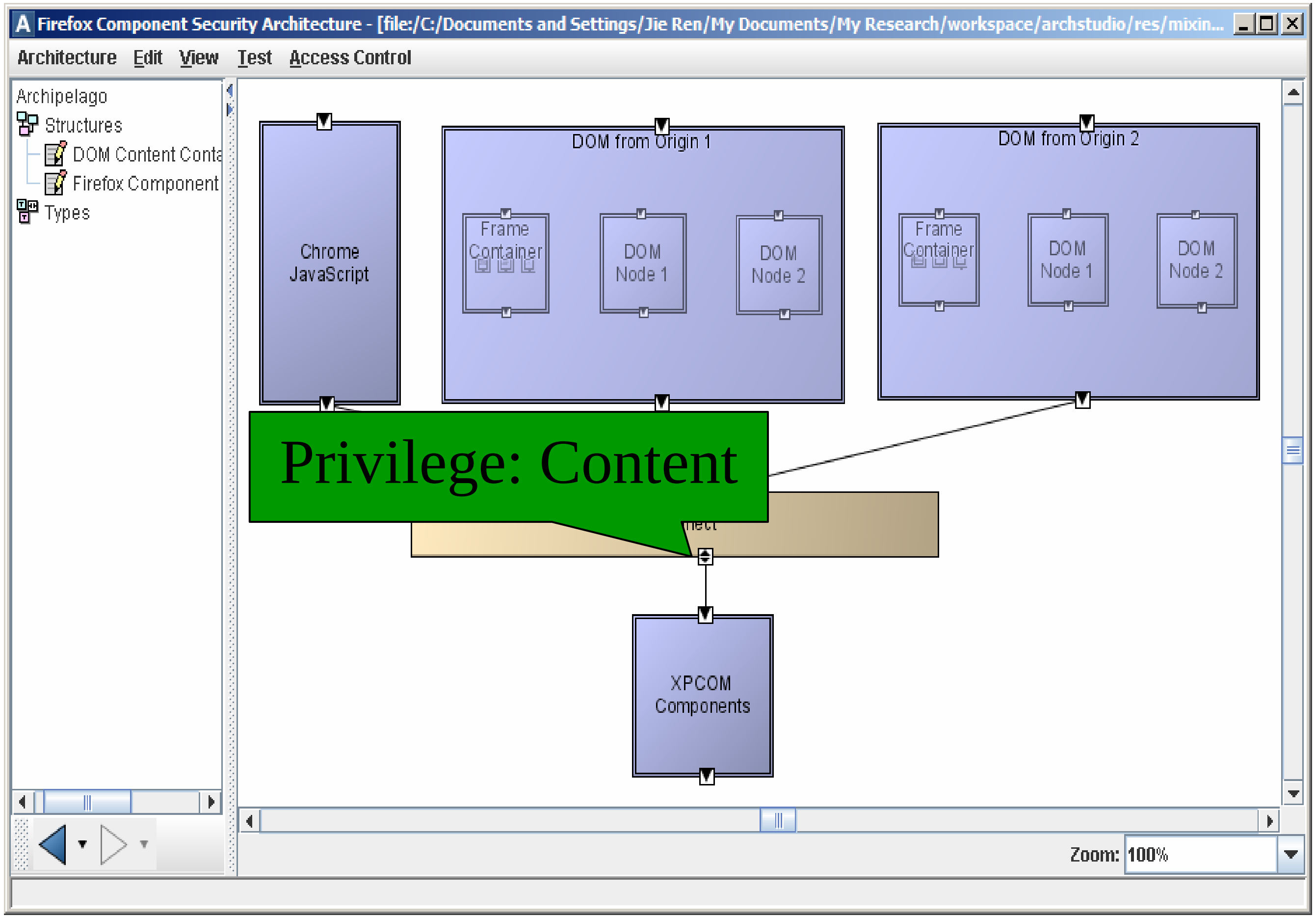
3. Propagate privileges along the path

Applying Algorithm: Firefox

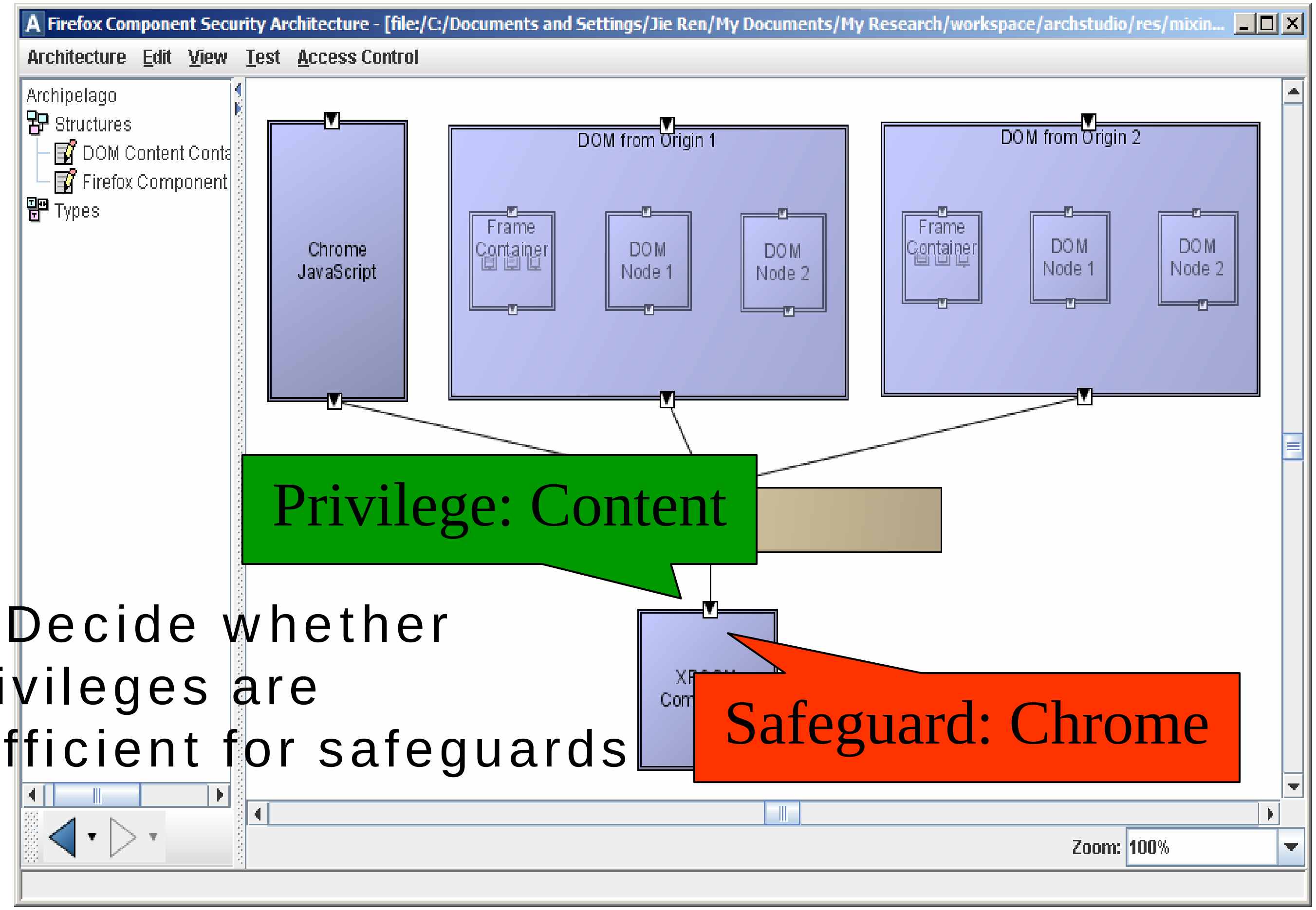


4. Propagation is subject to connector policy

Applying Algorithm: Firefox



Applying Algorithm: Firefox

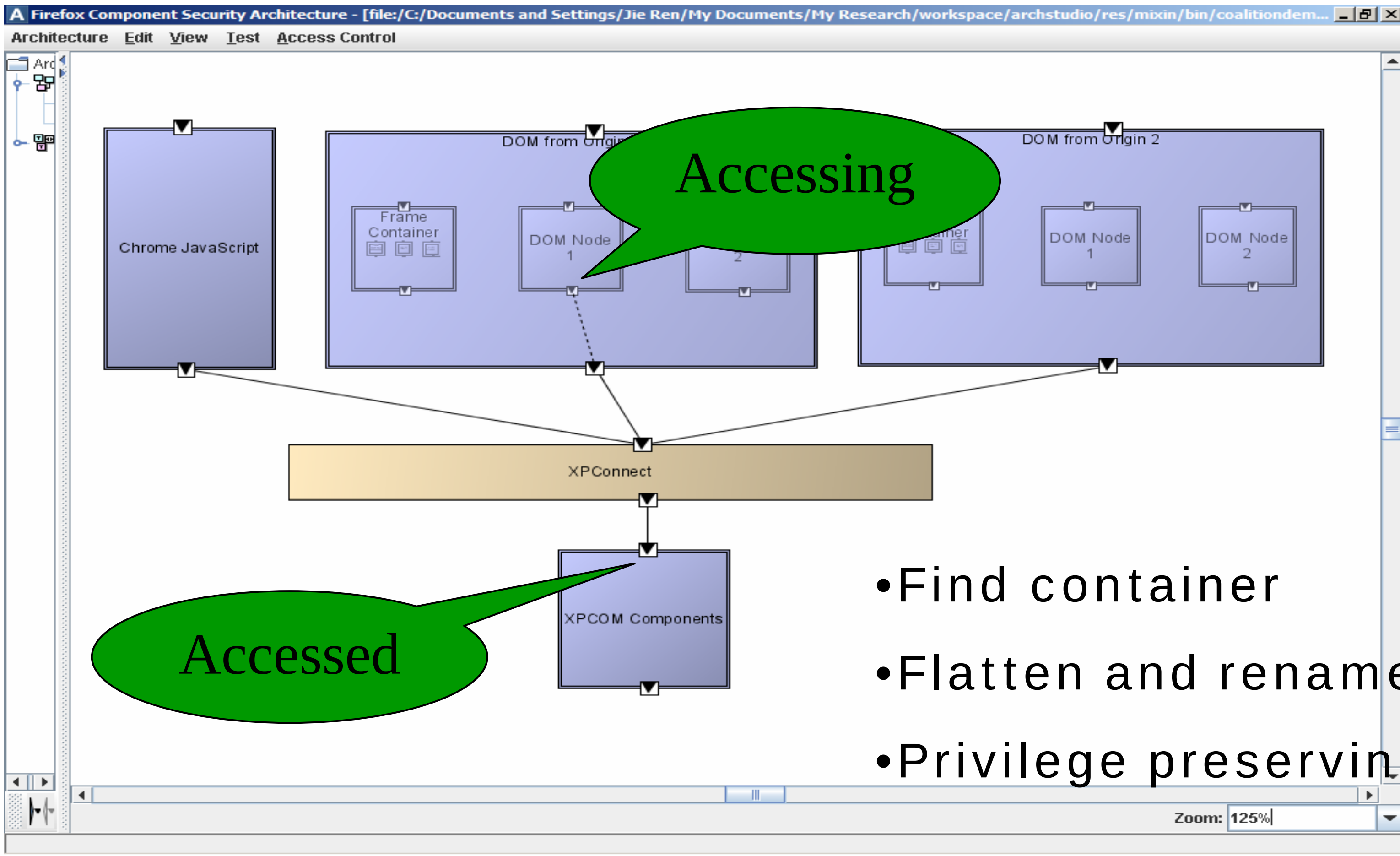


5. Decide whether privileges are sufficient for safeguards

Algorithm 2

```
Input:  an outgoing interface, Accessing, and
        an incoming interface, Accessed

Output: grant if the Accessing can access the Accessed,
        deny if the Accessing cannot access the Accessed

Begin
  if (Accessing and Accessed belong to the same architecture structure)
    container = the architecture structure
  else if (use top level architecture)
    container = top level architecture
  else
    container = least common container
  if (container contains other architecture structures) {
      replace constituents of subarchitectured types with
          the sub-architecture;
      rename the constituents of the sub-architectures if there
          are multiple instances of them;
      connect the outer signatures and the inner interfaces
          as privilege preserving
  }
  calculate the reachability closure of the expanded
      container interface graph
  return Algorithm1(Accessing, Accessed)
End;
```

Check with Subarchitecture



- Find container
- Flatten and rename
- Privilege preserving

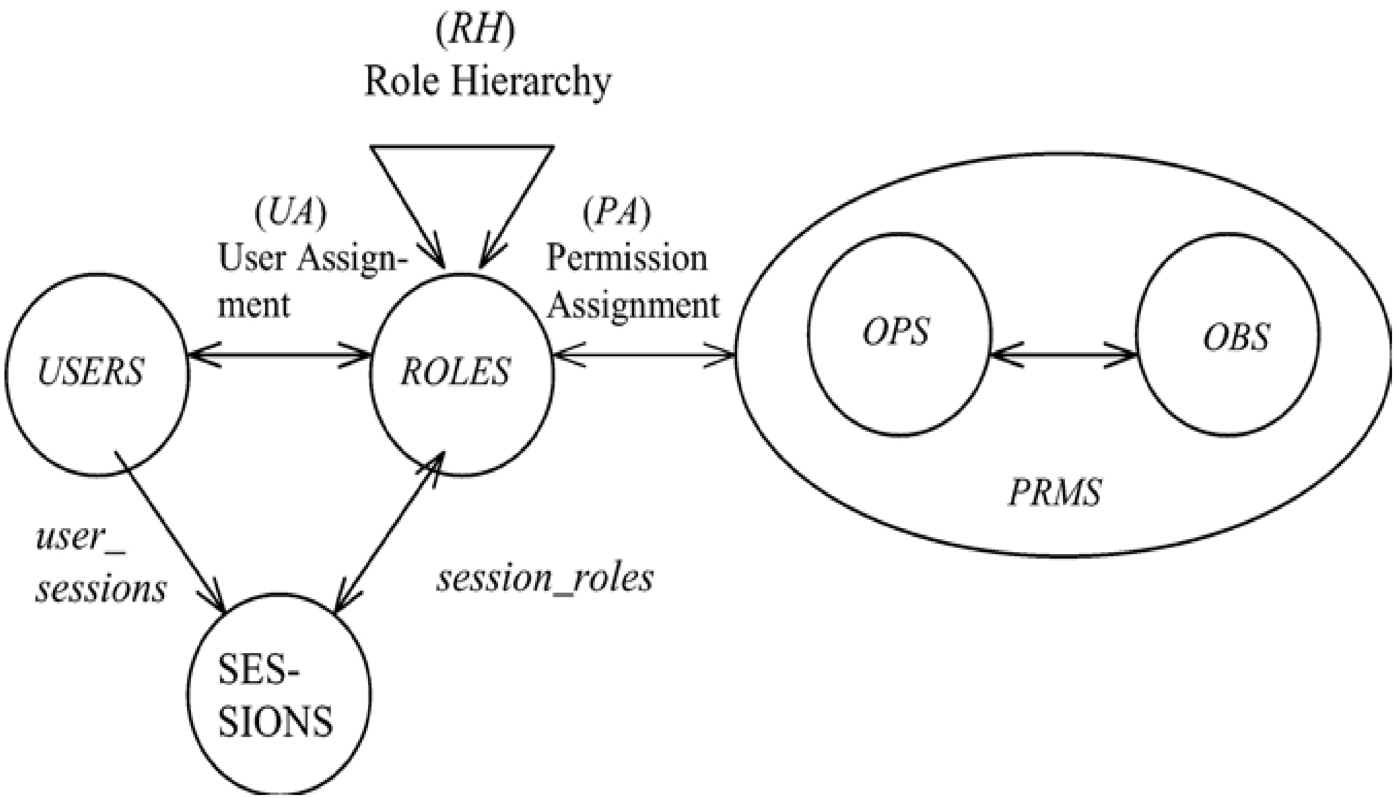
Validity of the Algorithm

- Reachability of a privilege graph
  - A privilege of an outgoing interface
  - A safeguard of an incoming interface
  - Connectors decide edges
- Sources of privileges and safeguards
  - Architectural contexts
- Assumptions
  - A single, loop-free path between the interfaces
  - Need manual help from architects in other cases

Advanced Modeling Concepts

- Four areas:
  - Handling large scale access through roles
  - Handling heterogeneous access through trust management
  - Handling content-based access
  - Handling architectural execution
- All can be modeled with the language and checked with the algorithm

Role-based Access Control



Roles in Secure xADL

- Roles as in the XACML RBAC Profile
  - Role Policy Set: restrict subject
  - Permission Policy Set: restrict resource and action
  - PolicySetIdReference
- Roles as principals
  - RPS and PPS
  - UA

Trust Management

- Handle authentication and authorization in a decentralized environment
- PolicyMaker, KeyNote, SD3
- A local decision maker makes a decision based on a credential presented by a remote party
- The credential is generally a certificate signed by the local decision maker
- A local policy is uniformly treated as a signed credential

Role-based Trust Management

- Ninghui Li 2003
- Based on set theory and logic
- Basic rule: $R_1.D_1 \leftarrow R_2.D_2$
- Trust as Roles
  - A foreign role can behave like a local role
- A natural extension to RBAC
  - Role equivalence similar to role inheritance

An Integrated Access Control Model

- Classic Access Control
  - Subject, object, privilege
- Role-based Access Control
  - Use a role as an indirection
- Role-based Trust Management
  - Trust relationship between roles of different domains

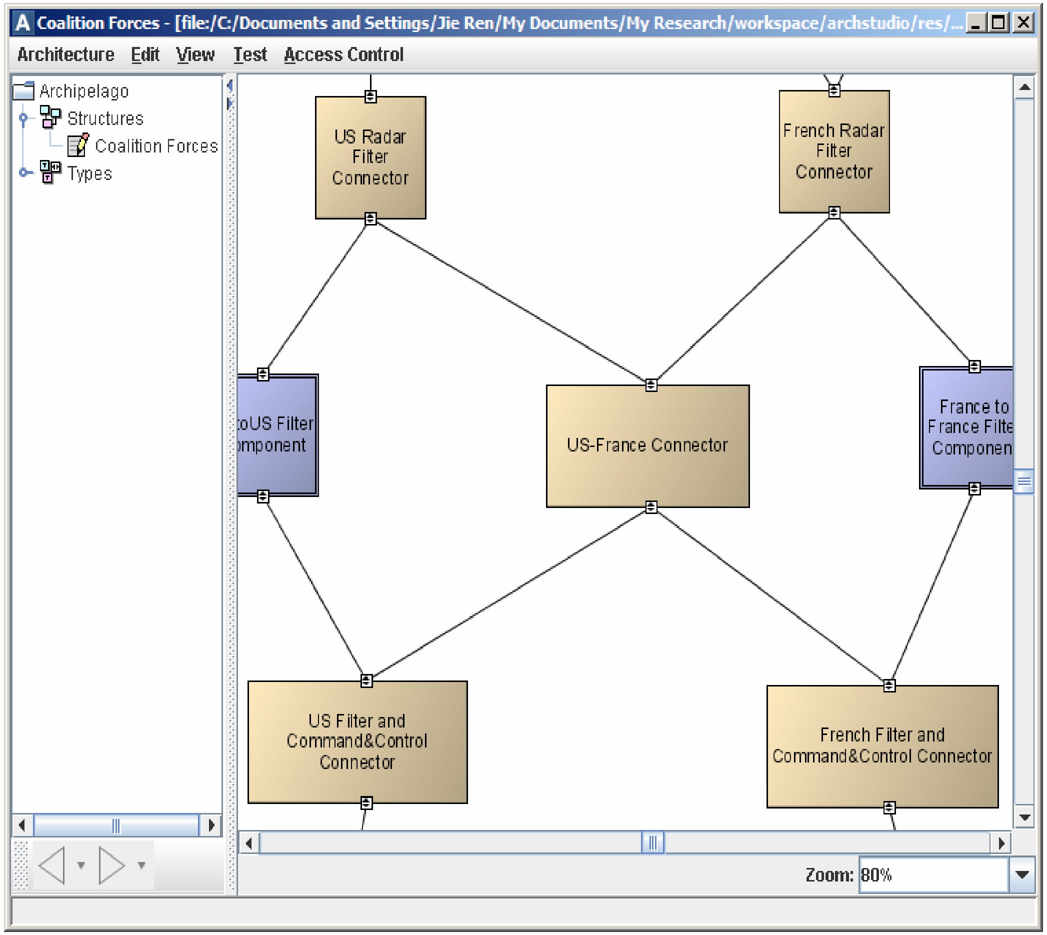
Content-based Access

- Interface-level access does not always provide enough information
- Inspecting content passing through interfaces could be necessary
- Event-based interfaces
  - Top and bottom
  - Request and notification

Architectural Execution

- Architectural Instantiation
  - Style neutral
- Architectural Connection
  - Style neutral
- Message Routing
  - Style specific

Coalition with One Connector



```
<connector id="USFranceConnector" xsi:type="SecureConnector">
    <principal>France</principal>
    <principal>US</principal>
    <policies>
      <PolicySet PolicySetId="InternalRouting"
                 PolicyCombiningAlgId="permit-overrides">
        <Policy RuleCombiningAlgId="permit-overrides">
          <Rule Effect="Deny" />
      <PolicySet PolicySetId="PPS:France"
                 PolicyCombiningAlgId="permit-overrides">
        <Policy RuleCombiningAlgId="permit-overrides">
          <Rule Effect="Permit">
            <SubjectMatch MatchId="string-equal">
              <AttributeValue>USFranceConnector
              <AttributeDesignator>subject-id
            <ResourceMatch MatchId="string-equal">
              <AttributeValue>RouteMessage
              <AttributeDesignator>resource-id
            <ActionMatch MatchId="string-equal">
              <AttributeValue>xadl:action:RouteMessage
              <AttributeDesignator>action-id
            <Condition FunctionId="string-equal">
              <AttributeValue>Air Defense Missile
              <AttributeSelector RequestContextPath=
    "//context:ResourceContent/security:routeMessage/
     messages:namedProperty[messages:name='type']/
     messages:value/text()"/>
      <PolicySet PolicySetId="PPS:US" PolicyCombiningAlgId="permit-overrides">
```

Role-based Access Control

Content-based Routing

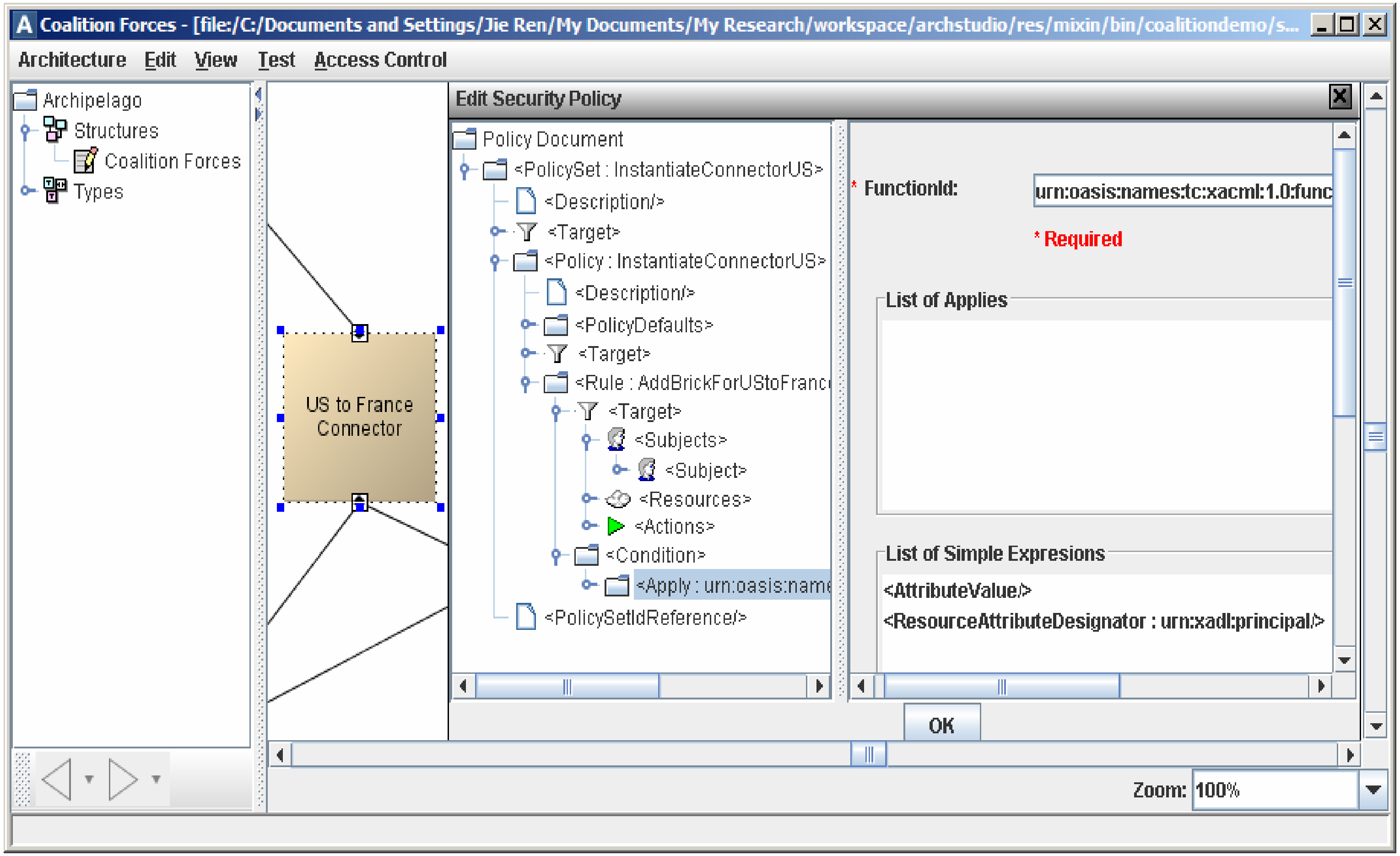
Central Role of Connectors

- Propagate privileges in architectural access check
- Route messages according to established policies
- Participate in deciding architectural connections
- Decide what subjects the connected components are executing for
- Regulate whether components have sufficient privileges to communicate through the connectors
- Provide secure interaction between insecure components

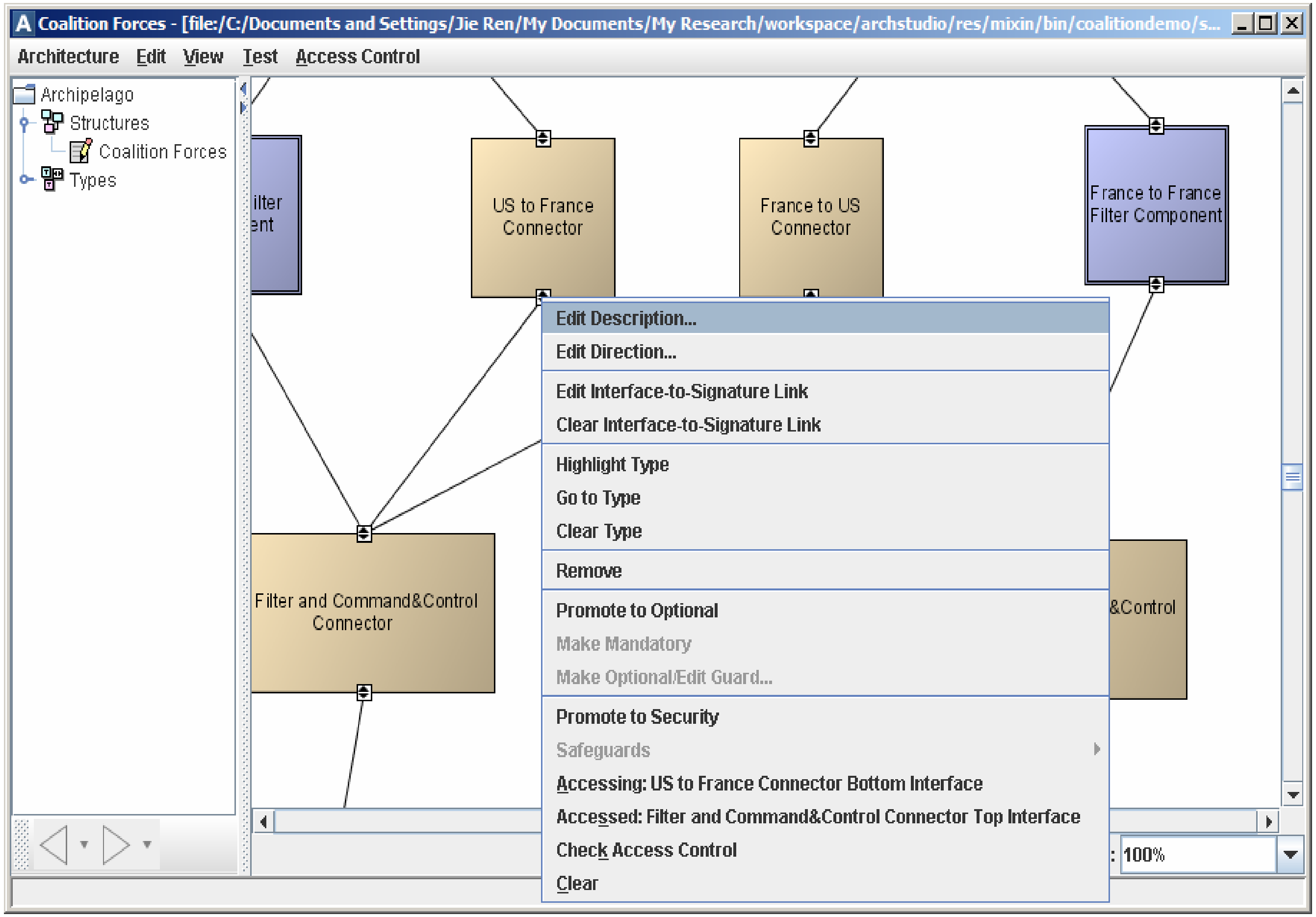
Tool Support

- Evaluation Engines
- Extending ArchStudio
  - Design-time support
    - Editors
    - Analyzer
  - Run-time support
    - PDP and PEP
    - c2.fw.secure
    - Secure Architecture Controller
    - Instantiation, connection, messaging

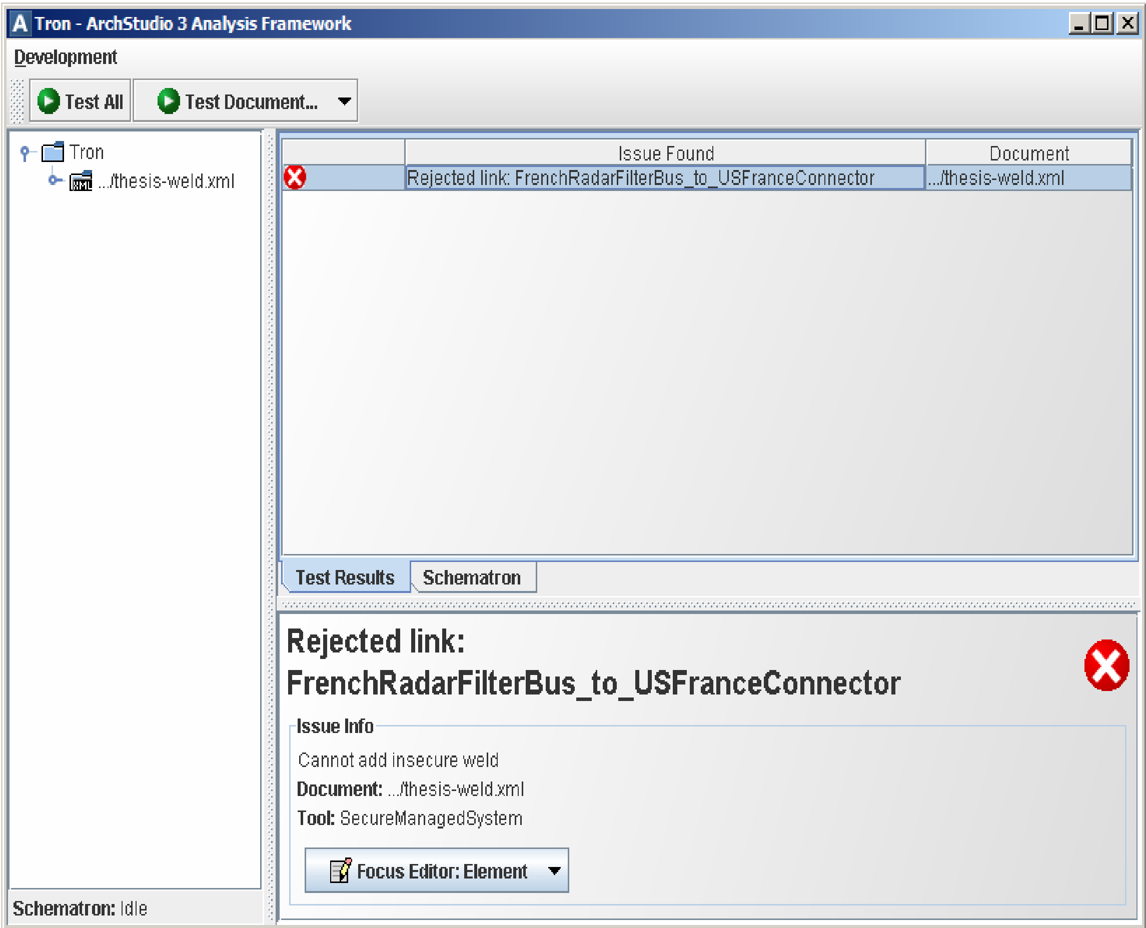
Policy Editor



Static Analysis



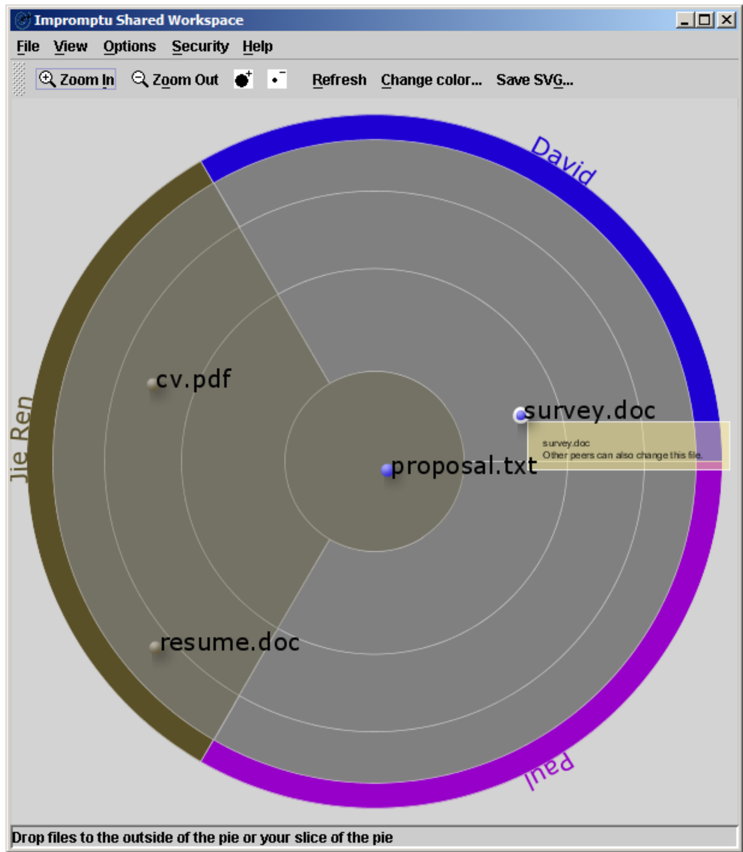
Instantiation and Connection Exceptions



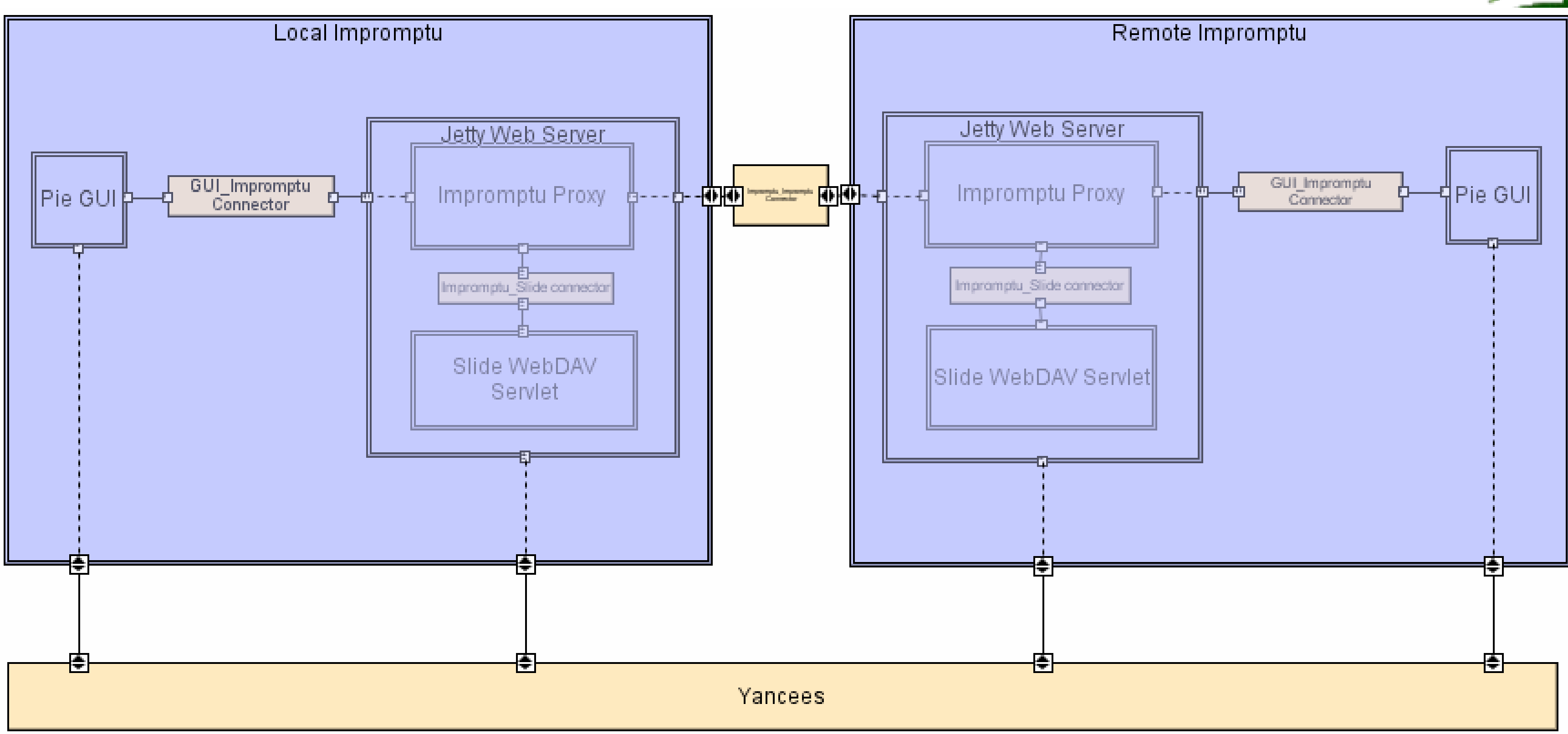
Case Studies

- Coalition
  - Developed, fully supported by ArchStudio
- Impromptu
  - Developed, reusing third party components
- Firefox Component Security
- DCOM Security

Case Study: Impromptu



Impromptu Components and Connectors



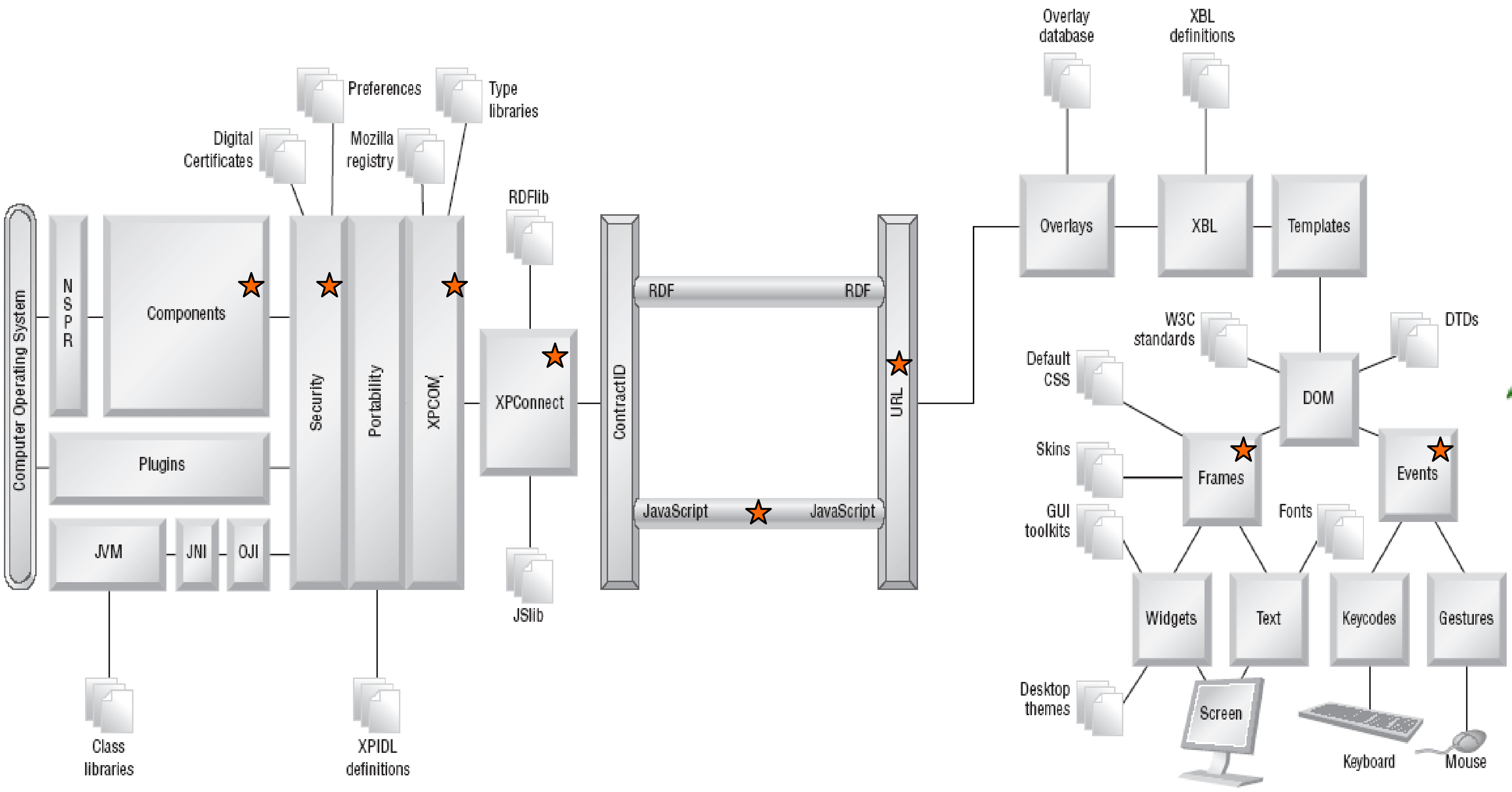
First Secure Connector

- Roles: me, other
- WebDAV connector
- Use IP address to separate me from other

Second Composite Connector

- Standard compliant
- Composite
  - HTTP Digest Authentication
  - web.xml authorization on HTTP methods
  - WebDAV ACL authorization on permissions
- Enable all types of files, with the WebDAV file system driver support

Case Study: Firefox



Firefox Platform

- XPCOM
  - Cross platform component model
- JavaScript
  - Browser and extension
- XPConnect
  - Bidirectional bridge between XPCOM components and JavaScript objects

Trust Boundaries

- The boundary between chrome and content
- The boundary between contents from different origins
  - Same origin: scheme, host, port

Principals

- Subject principal and object principal
- System principal, null principal

Container and Node

- Document Object Model
- Document and Frame
  - Principal based on origin
- Node
  - Inherit principal
- Components collection

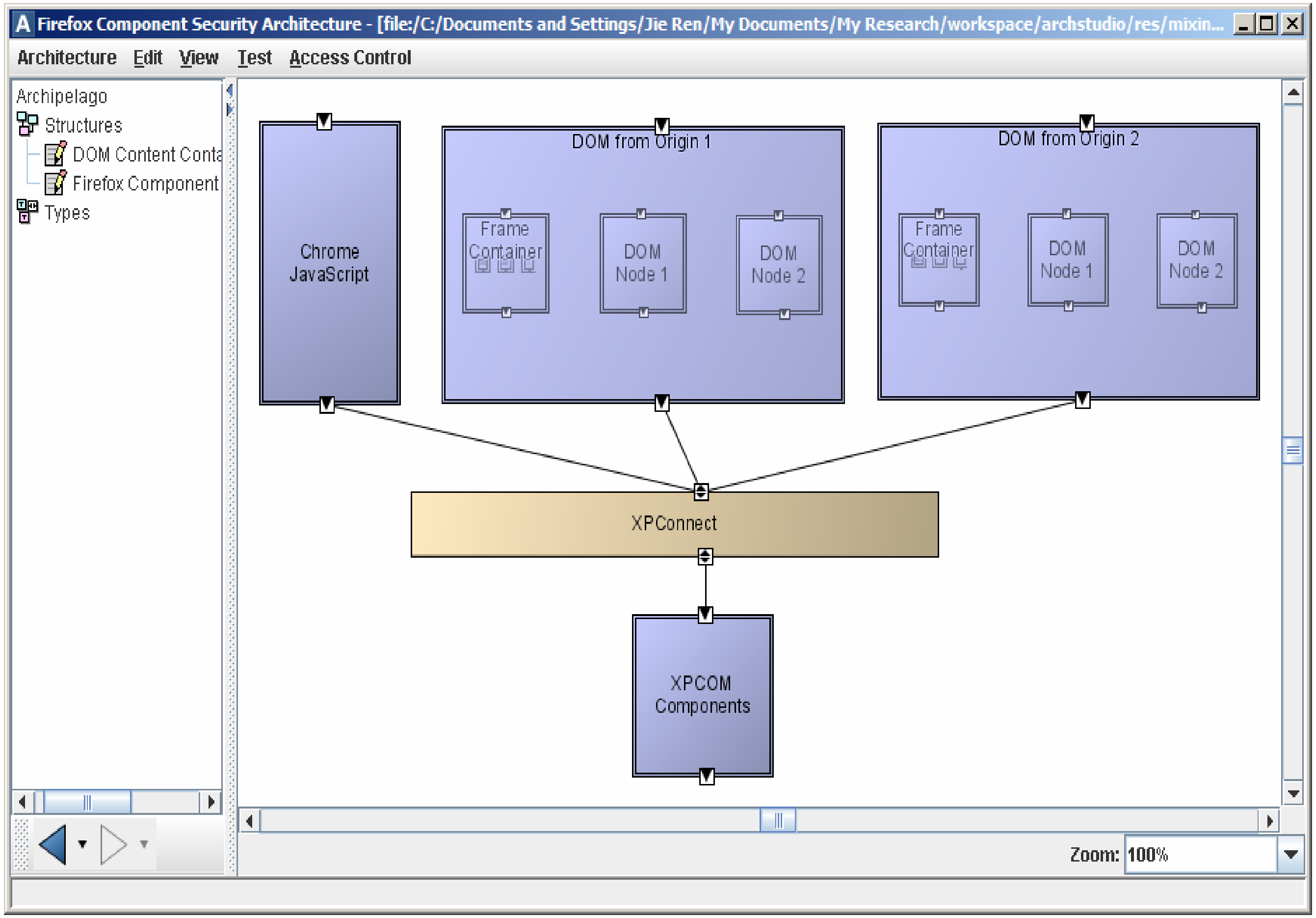
Script Security Manager

- Part of XPConnect
- Discover object principals and subject principals
- Architectural Access Control
  - – DOM access
    - Check subject principal and object principal
  - – Instantiation by Creation
  - – Instantiation by LoadURI

Firefox Security Policy

```
<component id="ChromeCode">
  <subject>ChromeCode</subject>
  <principal>Chrome</principal>
<component id="ContentCode">
  <subject>URI</subject>
  <principal>Content</principal>
<component id="SignedContentCode">
  <subject>SignedURI</subject>
  <principal>Chrome</principal>
<connector id="XPConnectSecurityManager" xsi:type="SecureConnector">
  <PolicySet PolicySetId="PPS:Chrome" PolicyCombiningAlgId="permit-overrides">
    <Policy RuleCombiningAlgId="permit-overrides">
      <Rule Effect="Permit">
        <Subjects>
          <Subject><SubjectMatch MatchId="string-equal">
              <AttributeValue>ChromeCode<AttributeDesignator>subject-id
          <Subject><SubjectMatch MatchId="string-equal">
              <AttributeValue>SignedURI<AttributeDesignator>subject-id
        <AnyResource />
        <AnyAction />
  <PolicySet PolicySetId="PPS:Content" PolicyCombiningAlgId="deny-overrides">
    <Policy RuleCombiningAlgId="deny-overrides">
      <Rule Effect="Permit">
        <SubjectMatch MatchId="string-equal"><AttributeValue>URI
              <AttributeDesignator>subject-id
        <ResourceMatch MatchId="string-equal"><AttributeValue>URI
              <AttributeDesignator>resource-id
        <ActionMatch MatchId="string-equal"><AttributeValue>AccessDOM
              <AttributeDesignator>action-id
      <Rule Effect="Deny">
```

XPConnect: Architectural Connector



Summary

- Problem: Architectural Access Control
  - How can we describe and check access control issues at the software architecture level?
- Approach:
  - A unified access control model: classic, role, trust
  - Subject, Principal, Resource, Privilege, Safeguard, and Policy
  - Contexts
  - Algorithm to check access control
  - Content-based access
  - Architectural execution
  - Connector-centric: propagation, connection, messaging
  - Tool support

Contributions

- A novel approach to the design and analysis of the access control property for software architectures
- A usable formalism for modeling and reasoning about architectural access control
- An algorithm for checking whether the architectural model maintains proper access control at design-time
- A suite of usable tools to design and analyze secure software

Future Work

- Different types of connectors
- Different mechanisms to construct connectors
- Security as an aspect
- Reflective architectural model
- Dynamic architecture
- Policy conflict resolution