A Multi-View Approach for Embedded Information System Security

Manuel Munier

LIUPPA
University of Pau - France

CRiSIS 2010

October 10-13, 2010

Montréal, Québec, Canada

Table of contents

1. Intelligent Documents
   - Context of information sharing
   - Object oriented approach
2. Privacy vs. Integrity
   - Dilemma for Information System Security
   - Multi-view approach
   - Secure Versioned Repository model
   - Benefits
3. Conclusion & Perspectives

Table of contents

1. Intelligent Documents
   - Context of information sharing
   - Object oriented approach
2. Privacy vs. Integrity
   - Dilemma for Information System Security
   - Multi-view approach
   - Secure Versioned Repository model
   - Benefits
3. Conclusion & Perspectives

Context of information sharing

- Information sharing
  - collaborative work for enterprises: reports, medical records, tender documents,. . .
  - documents can go outside the company where they have been designed (export from IS). . . and return (import updated documents)
- Specific needs
  - multi-site enterprises, virtual enterprises, nomadic users
  - usability with legacy applications: email attachment, USB memory stick, share resource on a WebDAV server,. . .
  - users can update shared documents ($\neq$ multimedia DRM)

$\Rightarrow$ "Classical" centralized architectures do not suit these needs

Object oriented approach

- OO approach to encapsulate
  - **data**: content of the document itself
  - security control **components**: access control, usage control, traceability, collaborative work management,. . .
- Usage
  - to "open" such a document, the user should provide her/his license
  - security control components are configured according to user's permissions (contained in the license)
  - they check all the accesses to information (embedded IS)
  - user can forward the document to another user (who handles the document according to his own license)

This paper

- Focus on the data model for the embedded IS
- What this article deals with
  - multi-view approach to ensure both confidentialty & integrity
  - formal model to store data & calculate views
  - mapping of user actions to "low level" actions
- What is not addressed in this paper
  - details of embedded components to enforce security controls
  - merging of concurrent updates made on different occurrences of the same document
  - expression and implementation of security policies

Table of contents

1. Intelligent Documents
   - Context of information sharing
   - Object oriented approach

2. Privacy vs. Integrity
   - Dilemma for Information System Security
   - Multi-view approach
   - Secure Versioned Repository model
   - Benefits

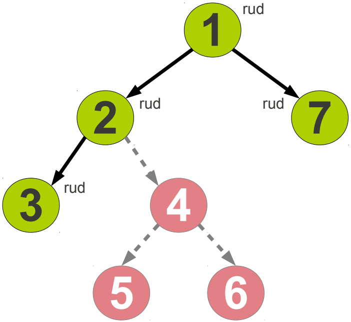
3. Conclusion & Perspectives

Dilemma for Information System Security

- **Confidentiality**: How to prevent the disclosure of information to unauthorized individuals (or systems)
  - breach of access control: someone can perform actions without the proper permissions
  - system behavior allows one to deduce the existence of hidden information
- **Integrity**: How to avoid data to be modified without authorization
  - someone accidentally (or with malicious intent) modifies/deletes data by side effects of a legitimate action

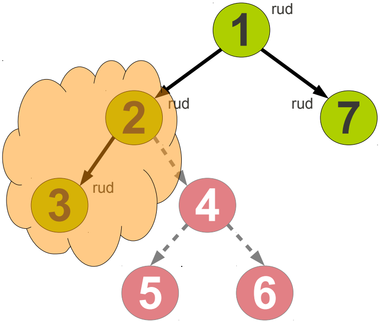
Example: removing nodes in data tree

- User can access nodes 1,2,3,7 with permissions <u>**r**</u>ead, <u>**u**</u>pdate and <u>**d**</u>elete
- He's not aware of nodes 4,5,6
- What happens if he decides to delete the node 2 ?



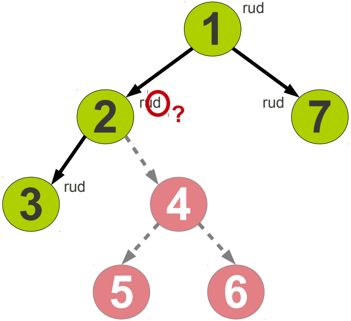
Example: removing nodes in data tree

- If the system accepts to remove nodes 2 and 3, what happens for node 4 ?
- Breach of integrity: node 4 is no longer attached to the tree



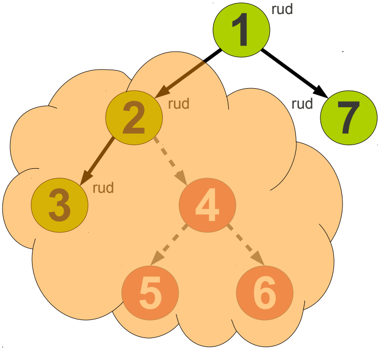
Example: removing nodes in data tree

- User is not allowed to delete node 4 (and its descendants)
- If the system refuses to remove nodes 2 and 3 to preserve the integrity of the data, then user can deduce the existence of hidden information (nodes 4,5,6)



Example: removing nodes in data tree

- If the system decides to remove nodes 4,5,6 to preserve the integrity, then user deleted unauthorized data (by side effects)

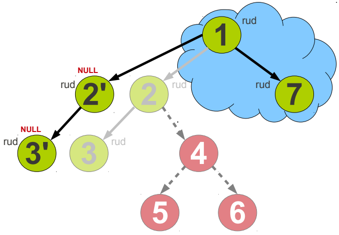


Multi-view approach

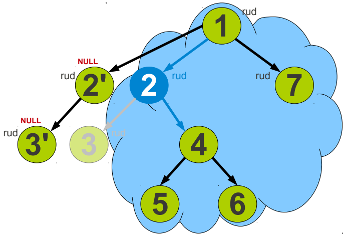
- We decouple "*what the user sees*" from "*what is stored*"
  - versions & relationships
    at the data store layer, all versions of each object are kept with their own relationships
  - computation of views
    a user has only a partial view of data contained in the store
  - mapping of user actions
    user actions (on user's view) have to be translated into basic actions (on the data store): create new versions, update relationships,...
- Goals
  - the user's actions have the intended effect on his view
  - the system preserves the integrity of data (e.g. relationships between nodes)

Multi-view approach

1. User Anna can't access nodes 4,5,6
   - After removing nodes 2,3 her view only contains nodes 1,7
   - Node 2' is the new version of node 2; value "NULL" indicates this node has been deleted and should no longer appear in Anna's view



2. User Bob can access nodes all nodes
   - Anna deleted nodes 2,3
   - Bob's view still contains node 2 to preserve the integrity of relationships between nodes 1,2,4



Secure Versioned Repository model

- Purpose of our model
  1. describe how to store data (versions & relationships)
  2. define how to compute user views
  3. translate user operations into actions on the store
- We define a formal model
  - to ensure properties on views w.r.t. user permissions
  - to formally describe the operations (like advanced transaction models for databases)
  - later to put (within security policy) some kind of access/usage control on semantic relationships

SeVeRe: data

1. Data model
   - like in CM tools we maintain multiple versions of each of data with their version relationships
   - data are not independent of each other
     - semantic relationships can denote various kinds of associations:

       `tree` (structural relation like "*father/child*" or "*container/content*")

       `use` (semantic relation like "*a code source use a library*", e.g. `#include`)
     - they are linked to versions, i.e. "data occurence" and not "logical data"
   - predicate $hold(uid, ob, p)$: permissions could be managed by an external model (e.g. ACLs)

SeVeRe: views

2. View computation

   a *Access Set*

   - this view contains all versions (and relationships) the user can access (he owns the permission <u>**a**</u>ccess)

Access Set (versions only)

$$O^{as} = \{o_{id,vid} \in O^{rep} \mid hold(uid, o_{id,vid}, \texttt{'a'})\}$$

SeVeRe: views

2. View computation

   b *Base View Set*

   - this view contains only the last version for each branch of versions (found in the *access set*)
   - "NULL" versions (i.e. deleted data) are removed

Base View Set **(versions only)**

$$\begin{cases} Last_{o_{id}} & = \quad \{o_{id,v} \neq \mathtt{NULL} \mid (o_{id} \in O^{as}) \;\wedge\; (\nexists\, o_{id,v'} \in O^{as} \mid o_{id,v} \succ o_{id,v'})\} \\ O^{bvs} & = \quad \bigcup_{o_{id} \in O^{as}} Last_{o_{id}} \end{cases}$$

SeVeRe: views

2. View computation
   - c *Extended View Set*
     - from the *access set* we reintroduce some versions not retained in the *base view set*
     - this aims to preserve integrity w.r.t. semantic relationships (e.g. node 2 in the previous example)

Extended View Set (versions only)

$$O^{evs} = O^{bvs} \cup \{ob_{1,x} \in O^{as} \mid (ob_{1,x} \rightarrow ob_{2,y}) \in R^{as} \wedge ob_{2,y} \in O^{evs}\}$$

SeVeRe: user operations

3. Mapping of user operations

   a. *Delete*

      - when a user deletes a version, this one (and its ancestors) does not have to appear any more in the base view set of this user

Property ***Delete***

$$delete_{uid}(ob_{x,y}) \in H \;\Rightarrow\; \forall\, ob_{x,z}\; (ob_{x,z} \succ^{*} ob_{x,y}) \Rightarrow (ob_{x,z} \notin BaseViewSet_{uid})$$

- to implement the *delete* operation we use "low level" actions to create a new version (with `"NULL"` as value) and to manage child and semantic relationships

SeVeRe: user operations

3. Mapping of user operations

   b *Update*

   - when the user *uid* invoques the $update_{uid}(ob_{x,y}, value)$ operation on his view, the expected effect is the disappearance of the version $ob_{x,y}$ which will be replaced by the version $ob_{x,y'}$ (successor of $ob_{x,y}$) with the given value

Property *Update*

$$update_{uid}(ob_{x,y}, value) \in H \;\Rightarrow\; \begin{cases} \quad ob_{x,y} \notin BaseViewSet_{uid} \\ \wedge \quad \exists\, ob_{x,y'} \in BaseViewSet_{uid} \;\mid\; (ob_{x,y'} = value) \wedge \;(ob_{x,y} \succ ob_{x,y'}) \end{cases}$$

- to implement the *update* operation we use "low level" actions to create a new version and to manage child and semantic relationships

Benefits

- This model is designed to simultaneously preserve the confidentiality and the integrity of data
  - version and relationship management
  - support for structured data (semantic relationships)
  - operations have the expected effects on the user's view regardless of what is done "behind"

- Clear separation of:
  - the data structure (versions, relationships, views)
  - the security policy (e.g. permissions for access control)

    the model relies on the predicat $hold(uid, ob, p)$
  - the implementation of user operations on views

Table of contents

1. Intelligent Documents
   - Context of information sharing
   - Object oriented approach
2. Privacy vs. Integrity
   - Dilemma for Information System Security
   - Multi-view approach
   - Secure Versioned Repository model
   - Benefits
3. Conclusion & Perspectives

Applications

- This work was implemented within a prototype of secure versioned repository (SeVeRe)
- The model has been extended to support operations on groups of objects

⇒ *Users can store structured documents like XML (where every node is represented by an object) and manipulate them via routines in the checkout/checkin style at the level of a whole document or as part of the document (and not node by node)*

Future works

- Define security policies taking advantage of possibilities offered by this model

  e.g. use metadata recorded during the user's actions for contextual decision making (cf. Or-BAC model)

- Extend the model to support some kind of access control on relationships too
- Experiment our SeVeRe prototype in the FLUOR project
  - collaborative work based on intelligent documents embedding a small information system built from our model
  - http://fluor.no-ip.fr/index.php

  this work was supported by the French ministry for research under the ANR-SESUR 2008-2010 project FLUOR

Manuel Munier

A Multi-View Approach for Embedded Information System Security

Thank you for your attention.

Annex 1

Case study

- 3 user groups:

group A	Alice, Alfred, Anna: they develop the program
group B	Bob, Bart: they develop the library
group C	Charly Clark: they write the report

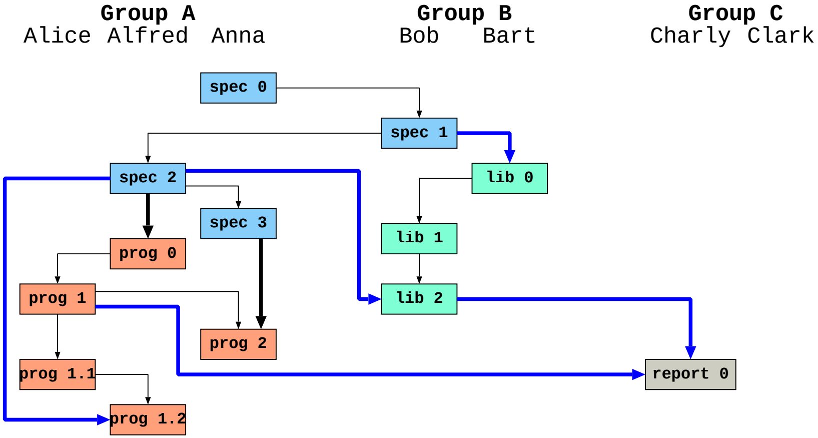
- they operate on 4 different resources:

specification	open to members of groups A and B
library	groups B and C
program	groups A and C
report	group C only

- resources are not independent of each other ⇒ relationships:
  - *spec* → *prog* (i.e. the program depends on the specification)
  - *spec* → *lib*
  - *prog* → *report*
  - *lib* → *report*

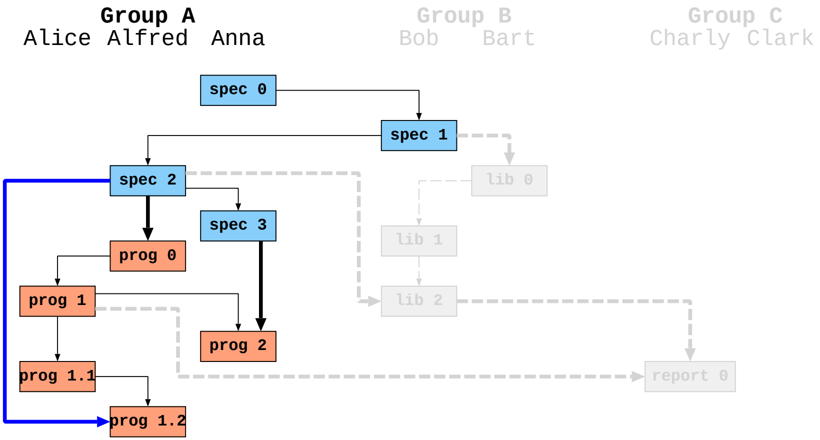
Annex 1

Case study: repository content



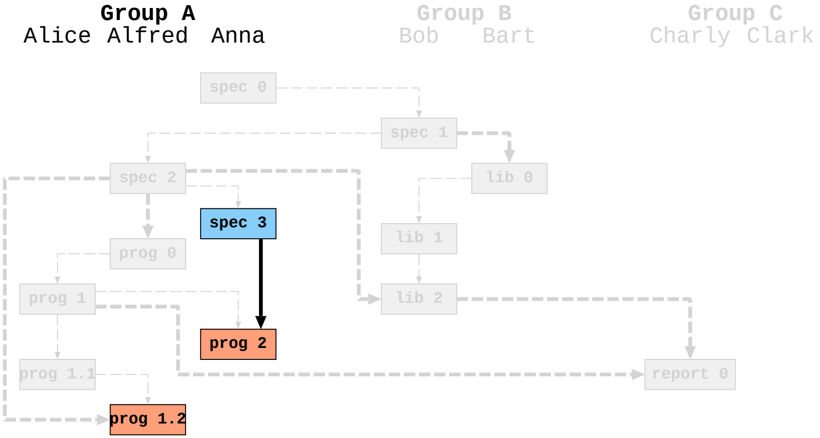
Annex 1

Case study: access set for group A



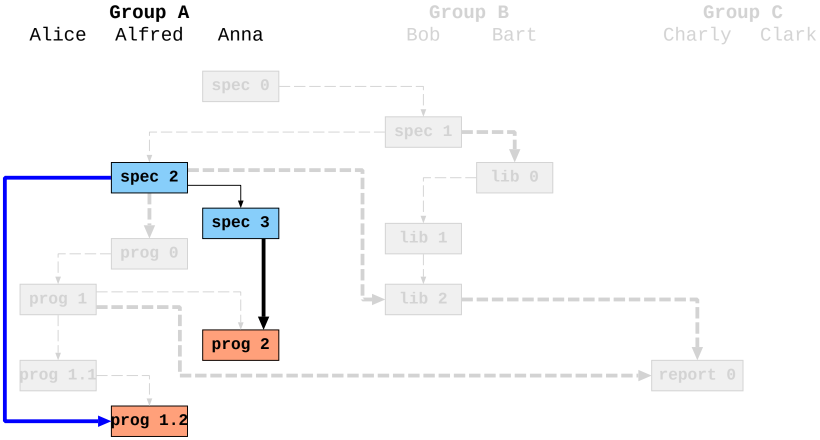
Annex 1

Case study: base view set for group A



Annex 1

Case study: extended view set for group A



Annex 2

User operation $delete_{uid}(ob_{x,y})$

Property *Delete*

$$delete_{uid}(ob_{x,y}) \in H \;\Rightarrow\; \forall\ ob_{x,z}\ (ob_{x,z} \succ^{*} ob_{x,y}) \Rightarrow (ob_{x,z} \notin BaseViewSet_{uid})$$

$rep.addVersion(ob_{x,y'}, NULL)$

$rep.addVRel(\langle ob_{x,y}, ob_{x,y'}\rangle)$

Annex 2

User operation $update_{uid}(ob_{x,y}, value)$

Property ***Update***

$$update_{uid}(ob_{x,y}, value) \in H \;\Rightarrow\; \begin{cases} \quad ob_{x,y} \notin BaseViewSet_{uid} \\ \wedge \quad \exists\ ob_{x,y'} \in BaseViewSet_{uid} \;\mid\; (ob_{x,y'} = value) \wedge (ob_{x,y} \succ ob_{x,y'}) \end{cases}$$

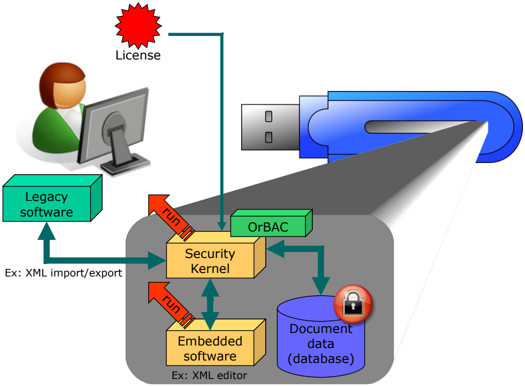
```
$rep.addVersion(ob_{x,y'}, value)$
$rep.addVRel(\langle ob_{x,y}, ob_{x,y'} \rangle)$

for each $\langle ob_{a,b}, ob_{x,y}, dep \rangle \in rep.getSRel(ob_{x,y})$ do
   $rep.addSRel(\langle ob_{a,b}, ob_{x,y'}, dep \rangle)$
done

for each $\langle ob_{x,y}, ob_{a,b}, dep \rangle \in rep.getSRel(ob_{x,y})$ do
   if $propagateOutgoingDep(dep) = true$ then
      $rep.delSRel(\langle ob_{x,y}, ob_{a,b}, dep \rangle)$
      $rep.addSRel(\langle ob_{x,y'}, ob_{a,b}, dep \rangle)$
   fi
done
```

Annex 3

FLUOR intelligent document architecture


License
Legacy software
Ex: XML import/export
run
OrBAC
Security Kernel
run
Embedded software
Ex: XML editor
Document data (database)