Adaptive Security with Aspect Oriented Programming

Maarten Rits, SAP Research

http:// mobius.inria.fr

Contract n° IST 015905

This work was funded in part by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2005-015905 MOBIUS project.
This presentation reflects only the author's views the Community is not liable for any use that may be made of the information contained therein

Mobius - Consortium

“We aim to develop the technology for establishing trust and security for the next generation of global computers, using the Proof Carrying Code (PCC) paradigm.”

- INRIA **(INRIA) France (C)**
- TLS Technologies **(TLS) Poland**
- ETH Zurich **(ETH) Switzerland**
- Radboud Universiteit Nijmegen **(RUN) Netherlands**
- Ludwig-Maximilian Universitat, Munich **(LMU) Germany**
- University of Edinburgh **(UEDIN) United Kingdom**
- Institute of Cybernetics, Tallinn **(IoC) Estonia**
- Chalmers Technical University **(CTH) Sweden**
- Imperial College, London **(IC) United Kingdom**
- University College Dublin **(UCD) Ireland**
- University of Warsaw **(WU) Poland**
- Trusted Logic **(TL) France**
- France Telecom **(FT) France**
- Universidad Politecnica de Madrid **(UPM) Spain**
- SAP AG **(SAP) Germany**
- RWTH Aachen **(RWTH) Germany**

Overview

- Security with Aspect Oriented Programming:
  - Application-Specific Security (Task 1.4)
- Metadata Driven Development Process
- Case Study: A Defense PLM Application

Security with Aspect Oriented Programming

- Problem:
  - Extend applications to satisfy new requirements
  - Extension cannot be modularized but affects functionality crosscutting the application
  - Increased implementation effort
- Solution:
  - AOSD decreases development effort by specifying extensions systematically in a single class (*aspect*) and then automatically modifying the involved method calls throughout the application
  - Extensions can be specified before, after or around the method call
  - Modifications can be introduced at compile-, load- or run-time

- AOSD is a means to systematically address evolving security requirements & to align with separation-of-concerns principle

AOSD Fundamentals


- Point: position within code
- Pointcuts: specific, selected points in application code
  - Execution: picks out points defining method execution (callee side)
  - Call: points defining method execution (caller side)
  - Set: picks out points defining field modification
  - Get: picks out points defining field access
  - Cflow: picks out points defining a control flow
  - Handler: picks out points defining where an exception is caught in a catch clause
- Composed Pointcut: using logical operators
- Advices: define what to do at the pointcuts
  - Around: invoked *around* the point (original code replaced)
  - Before: invoked *before* the point is reached
  - After: invoked *after* the point has been reached

Example of a pointcut


```
Cflow composition expresses the idea of the stack trace
(with a finer level of selection up to parameters types)

@Execution * Data.get*(..) IN * Display.addTable(Data)


public class Display {
        public void addTable(Data some_data) {
          int id = some_data.getId();
          …
        }
}

aDisplay.addTable(some_data);

some_data.getId();
```

In the control flow of Display.addTable()

Invocation of Data.getId()

Will match this call

But not this call

Pragmatics …

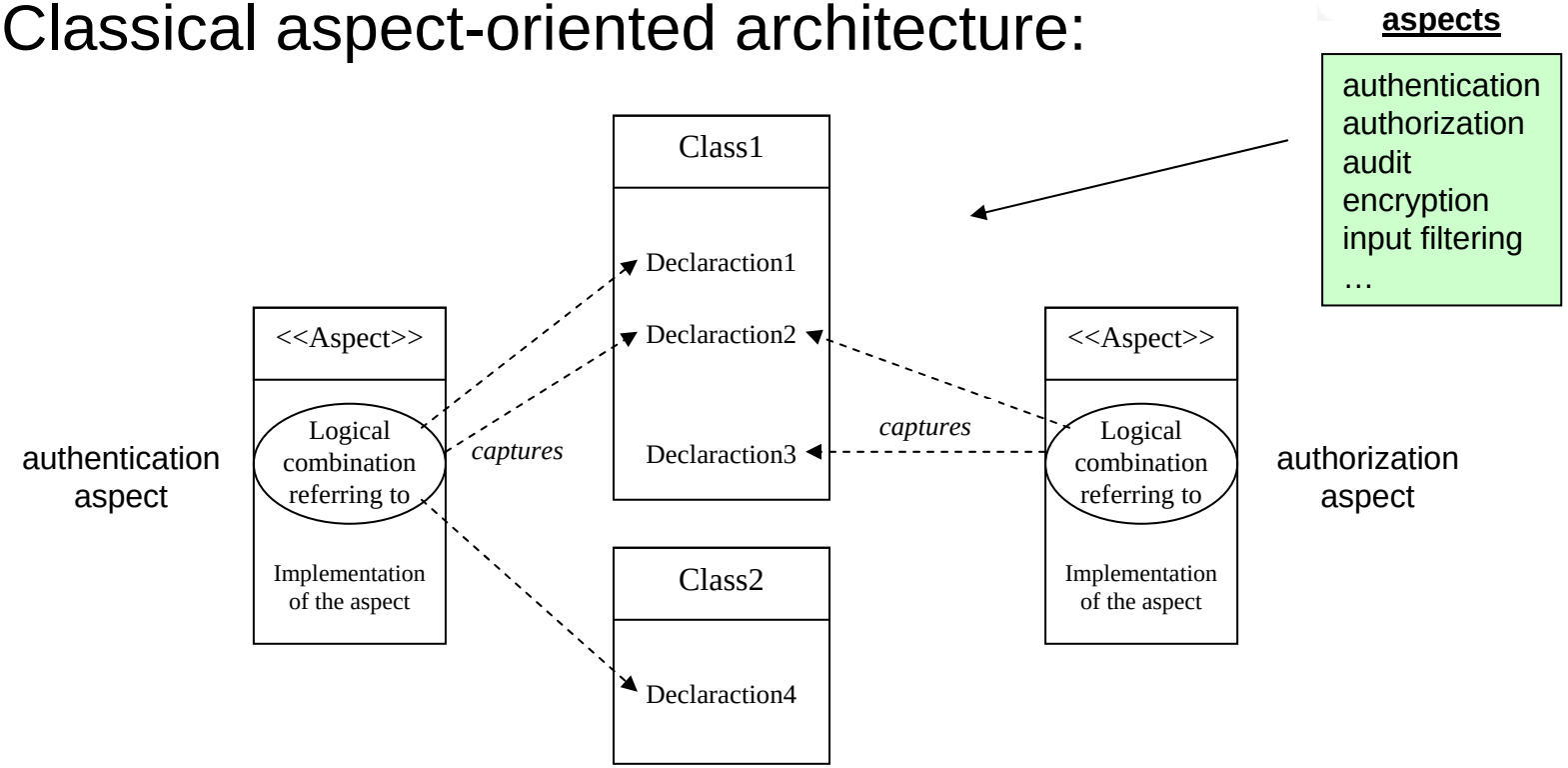
- AOSD is available for use in Java environments and .NET compliant languages
- AspectJ plug-in for Eclipse/NetWeaver development platform
- Performance impact
  - Compile-, and load-time:
    - The overhead of five *around* advices applied to a *method call* point
      - AspectJ 1.1.1 0.000097 ms/call

    (Source: BEA Systems)
  - Run-time:
    - The overhead of five *around* advices applied to a *method call* point
      - AspectWerkz 0.10 RC1 0.000163 ms/call
      - JBoss AOP 1.0Beta 0.000263 ms/call

    (Source: BEA Systems)

Metadata Driven Development Process

- Why use Metadata ?
- Classical aspect-oriented architecture:



- Can we improve the direct relationship between the pointcuts declarations and the core code ?


- Metadata
  - Since Version 1.5 (“Tiger”), Java supports annotations. It also provides a reflection API.

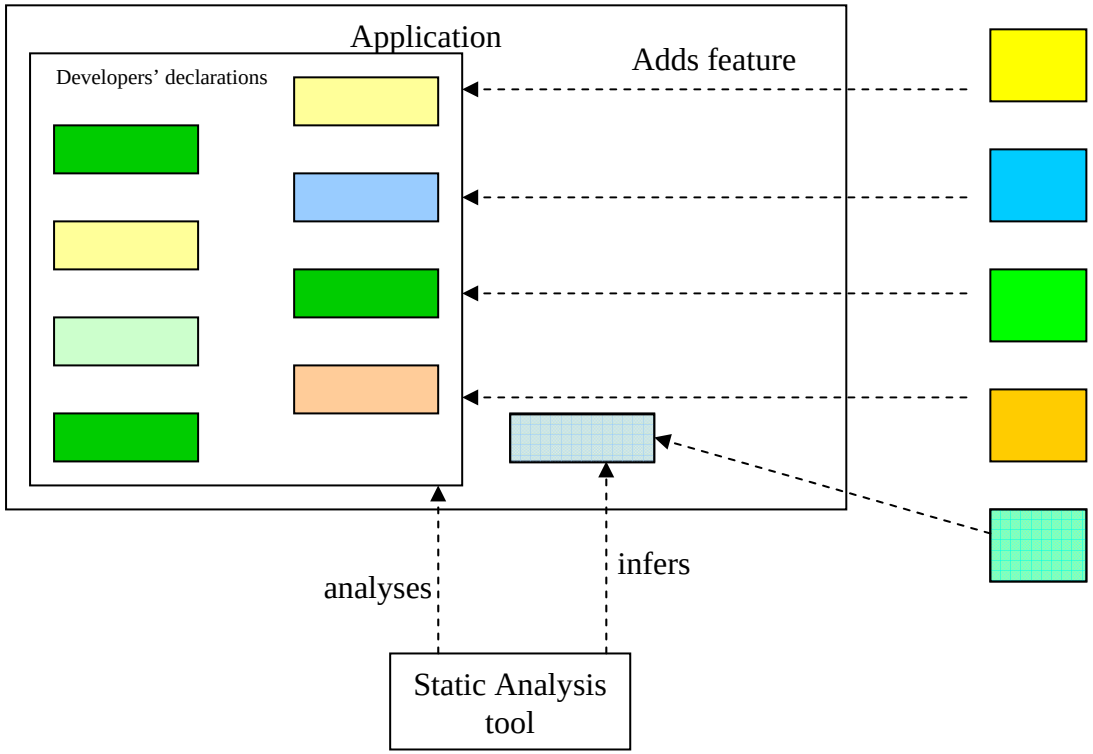
```
/* Associates an author notice with the annotated API element.
*/ public @interface Author { String value(); }

@Author("Maarten Rits")
public class SomeClass { ... }
```

- AspectJ allows the weaving of metadata into bytecode:
  - Declare @method: * *.someMethod(..) : @Audit/@Authentication/ etc.

- JML annotations …

  “you won’t be able to use the annotations/metadata syntax in Java 1.5 to write JML specifications. It’s simply not powerful enough for that.”

- We would like to express security properties like “authentication needs to occur before authorization” etc.



Primary Annotations are Java annotations manually written within the code of an application, or explicitly woven by a non-conditional aspect.

Secondary Annotations are Java annotations woven by an external program, based on primary or secondary annotations.

1. Write the core of the application
2. Insert primary annotations
3. Infer secondary annotations
4. Write and weave the code for the concerns
5. **Proof the adherence to high level security properties***

*Related work: M. Pavlova, G. Barthe, L. Burdy, M. Huisman, and J.-L. Lanet. Enforcing high-level security properties for applets. In P. Paradinas and J.-J. Quisquater, *Proceedings of CARDIS'04*, Toulouse, France, August 2004. Kluwer Academic Publishers.

Related Work


- Focuses on smart card applications
  - proposes a method to translate high level security properties into JML annotations (propagation of annotations)
  - Whether the applet respects its annotations can be established with the existing tools for JML (JACK, ESC/Java, Loop, etc.)
- Example of JML annotation
  - "No nested transactions"
  - A static ghost variable TRANS is declared that keeps track of whether there is a transaction in progress.

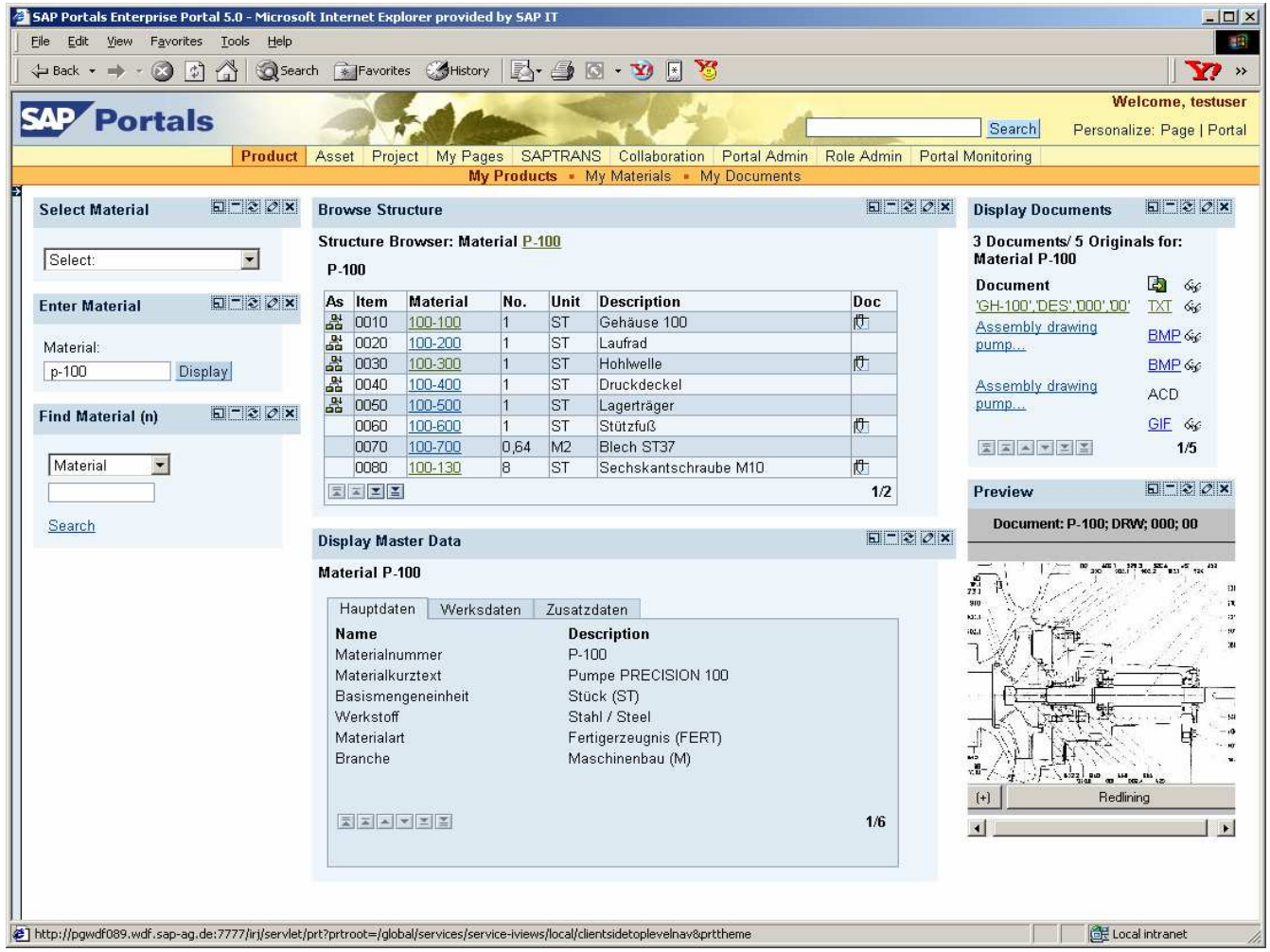
```
/*@ static ghost int TRANS == 0; @*/

The method beginTransaction is annotated as
follows.

/*@ requires TRANS == 0;
@ assignable TRANS;
@ ensures TRANS == 1; @*/
public static native void beginTransaction()
throws TransactionException;
```


- Case Study: A Defense PLM Application

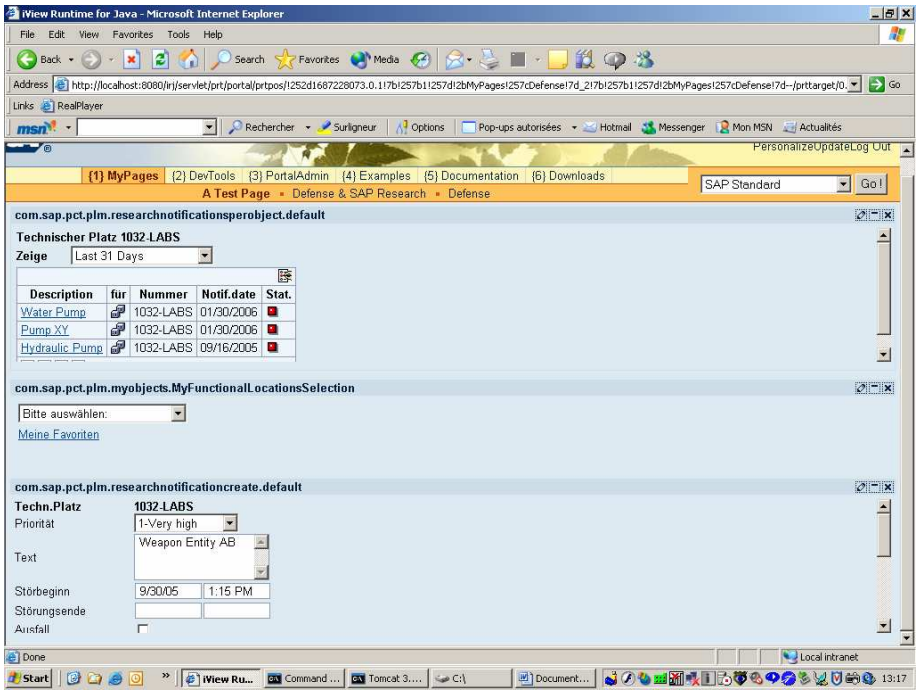
SAP Research and Defense …

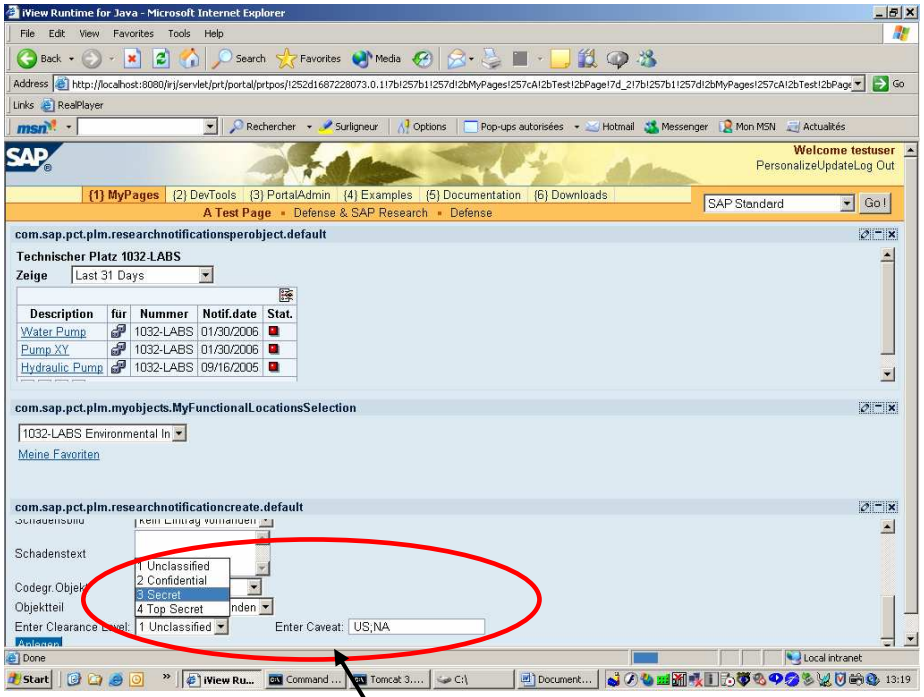
- Specific requirements for screen marking and access control (based on CAVEAT and clearance level) cf. DEIG DSSRS
- SAP Research analyses Enterprise Portal Solution
  - Java-based iViews
  - Aspect Oriented Software Development (AOSD)
- Demonstrator for Enterprise Portal 5.0:
  - Implementation of screen marking and access control in Java
  - Selected iViews are:
    - NotificationsPerObject
    - CreateNotifications
    - FunctionalLocationSelection
- Demonstrator with AOSD in NetWeaver Studio:
  - Reproducing a Web Dynpro structure (aka MVC)
  - Featuring aspects for:
    - Screen marking
    - XACML for a fully declarative and context-based access control
  - Performance overhead compared to hand-coded solution: 20-25 ms (total application execution time in test = 2000 ms)

PLM in Enterprise Portal 5.0



Modified with AOSD …



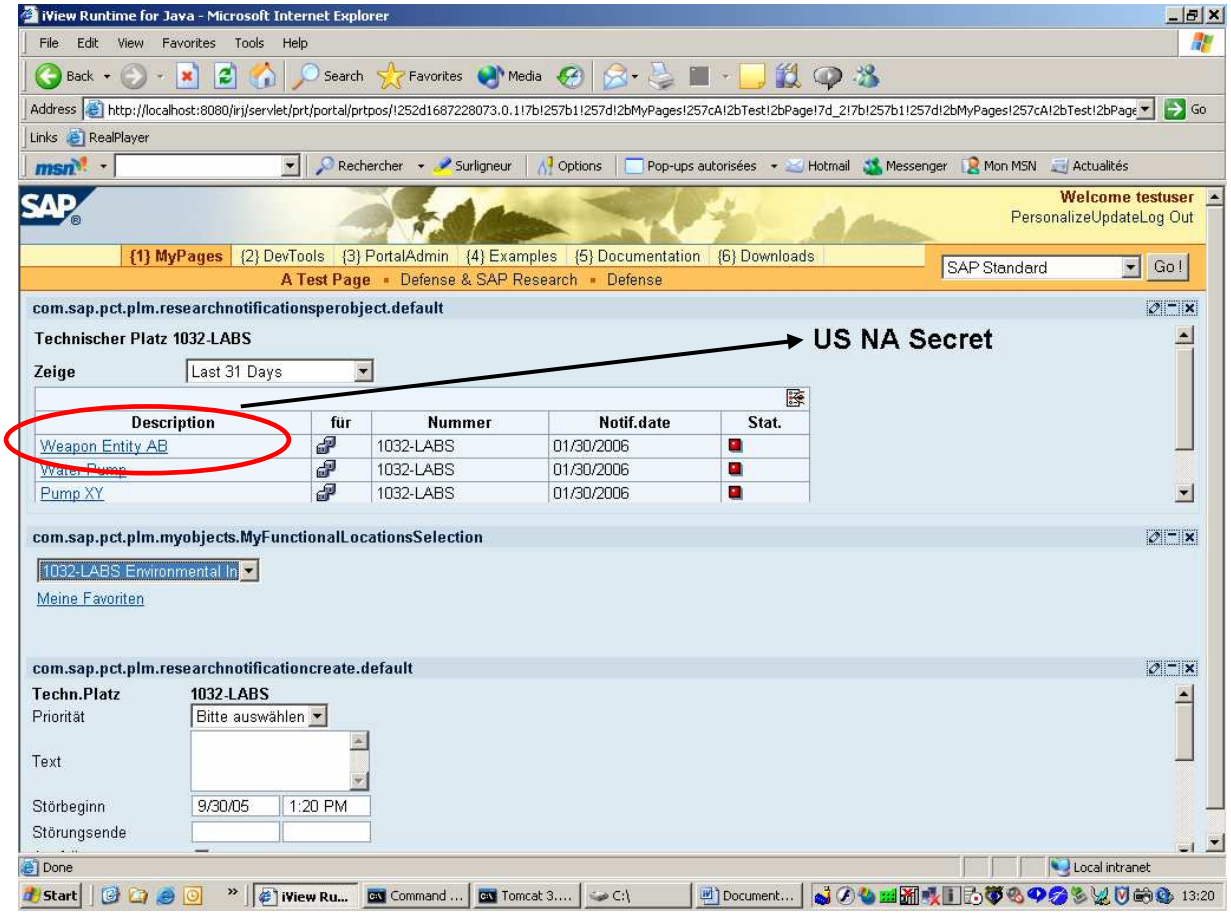


Make notification about defect weapon

Insert clearance level & CAVEAT for:

- **access control**
- **screen marking**

End Result (prototype)




- Questions ?

- Thank you for attending.