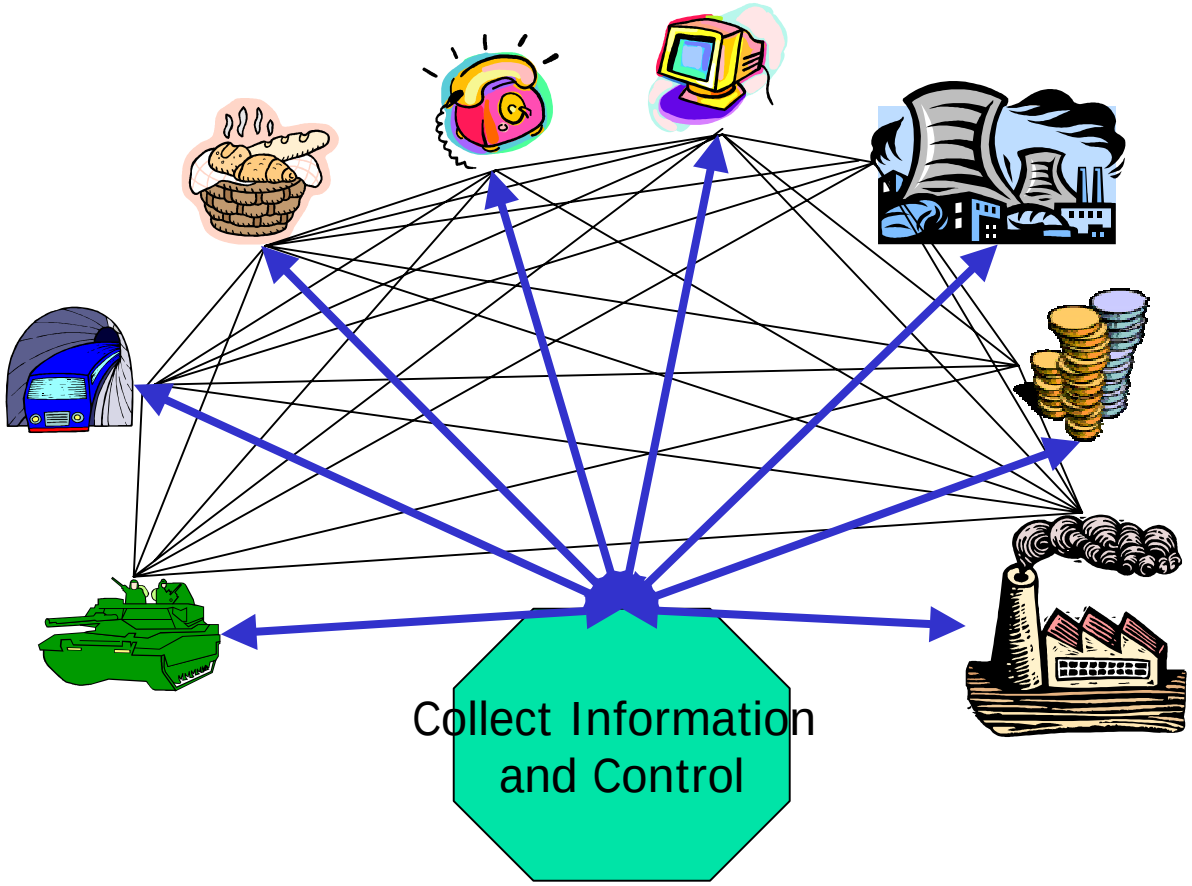
Security and Safety in Large Complex Critical Infrastructures

John Bigham

Department of Electronic Engineering, Queen Mary, University of London, UK

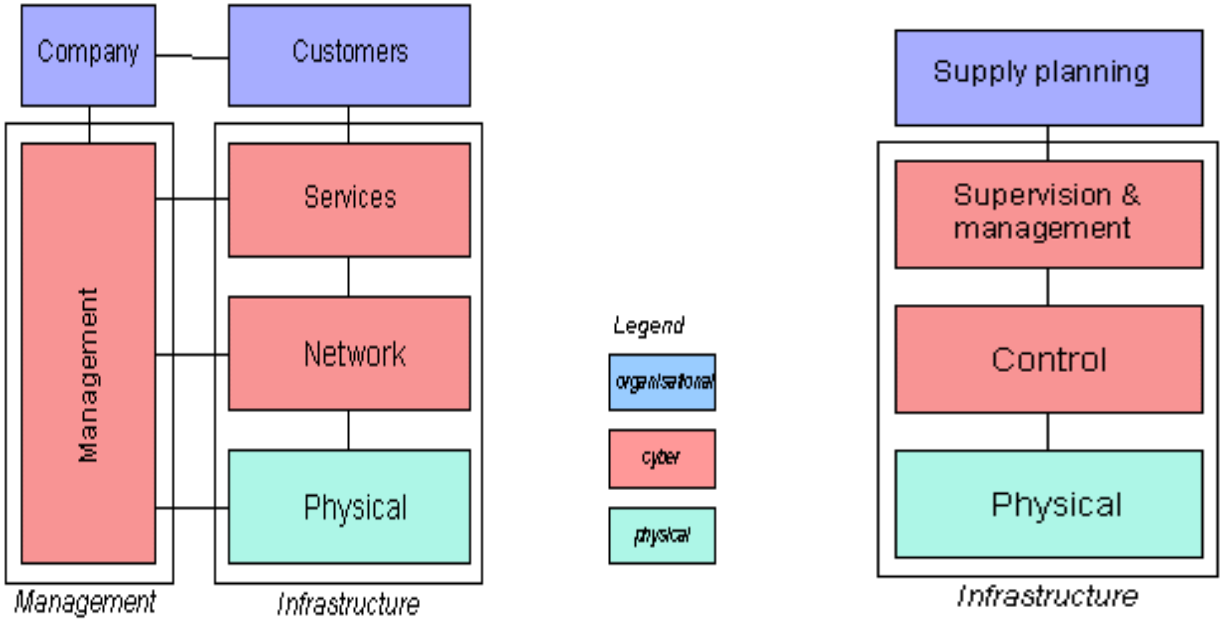
LCCIs



Safeguard

- European project developing an agent-based system to protect the management networks of large complex critical infrastructures, such as the telecommunications and electricity networks, against attacks, failures and accidents.
- Started December 2001, runs until May 2004.
- Safeguard website: www.ist-safeguard.org

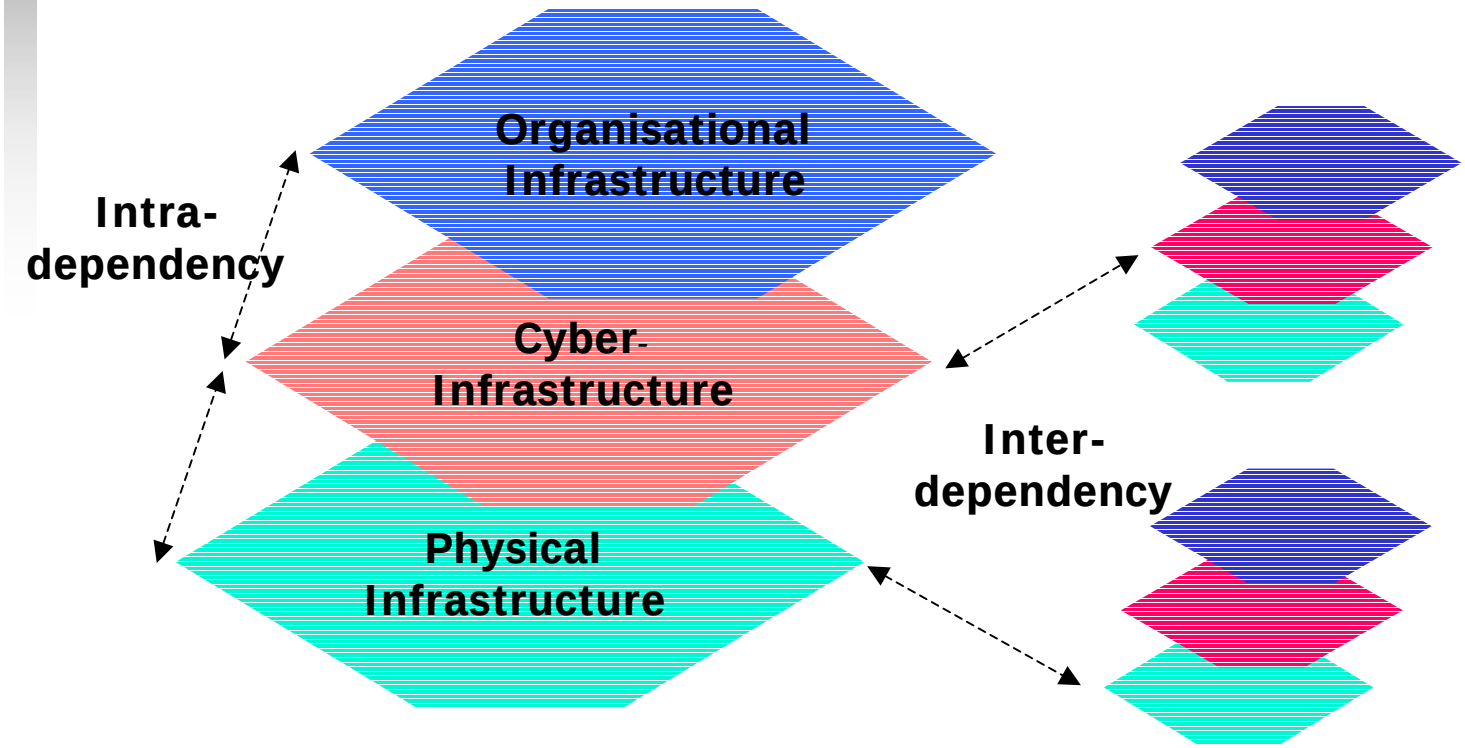
Complex systems are layered



Telecoms

Electricity

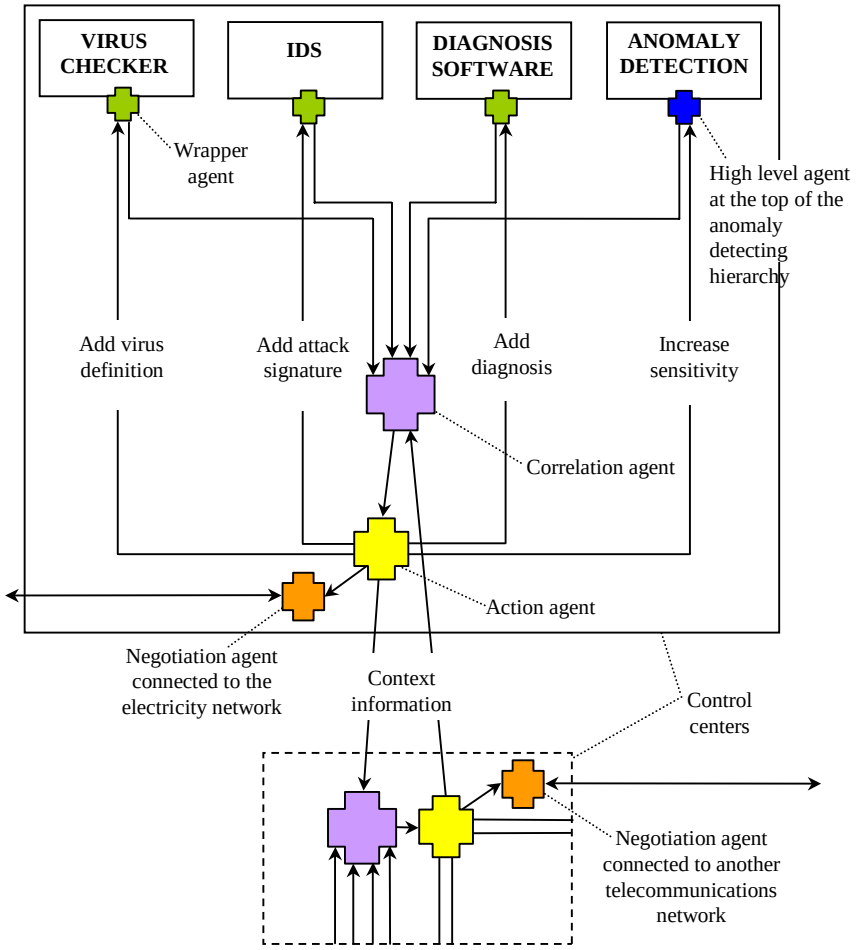
Dependencies between layers and between management centres



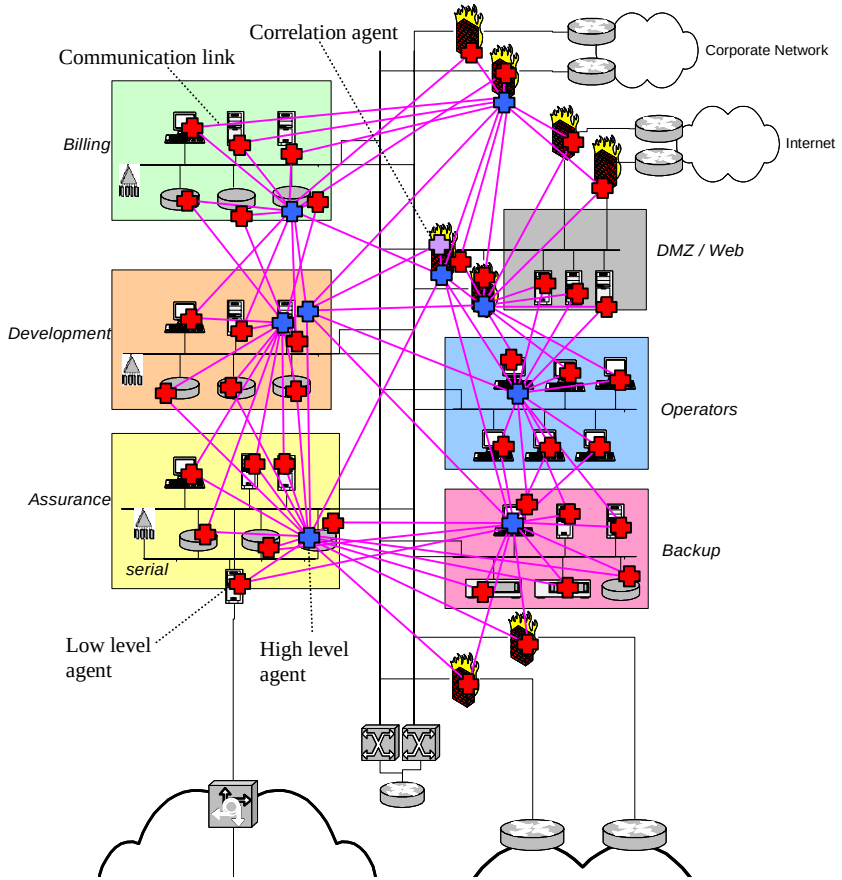
The role of Safeguard agents

- Maintain critical services
- The society of agents could have a hierarchy of roles:
  - Level 1 – identify component failure or attack in progress
  - Level 2 – self-healing to replace functions of failed component
  - Level 3 – if self-healing fails, isolate problem components and suggest reconfiguration strategy
- Need to be able to recognise dynamically changing
  - *Normal* behaviour
  - *Abnormal* but *acceptable* behaviour
  - *Abnormal* and *unacceptable* behaviour

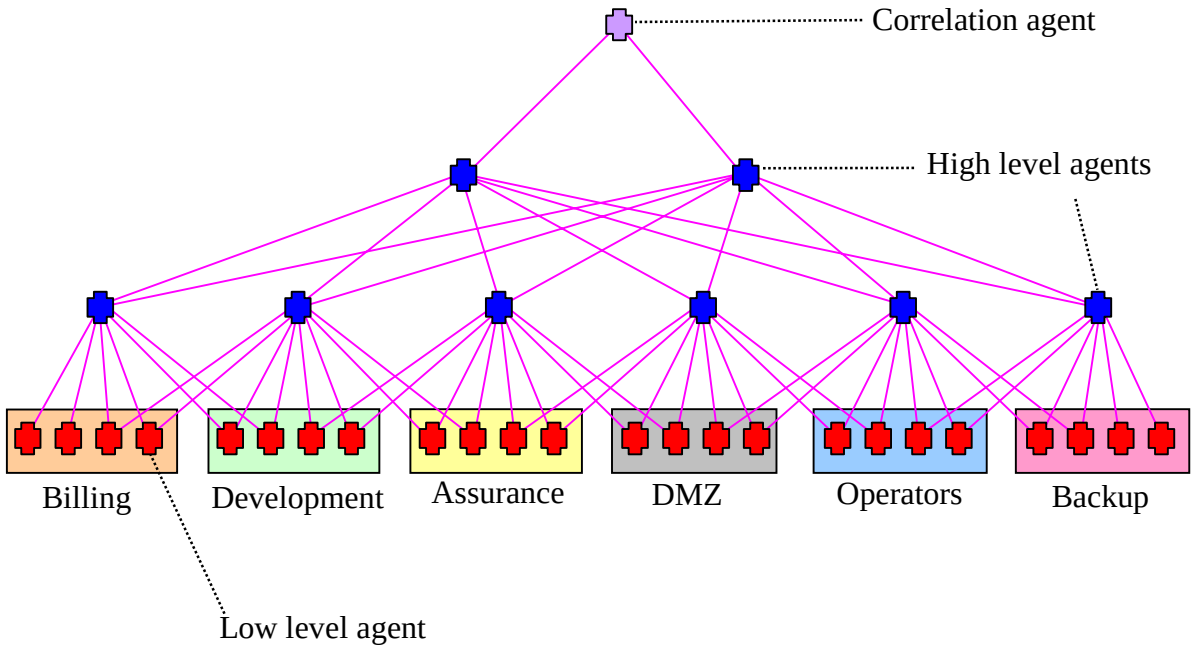
Agent architecture



Example: Anomaly detecting agents



Anomaly detecting agents (2)



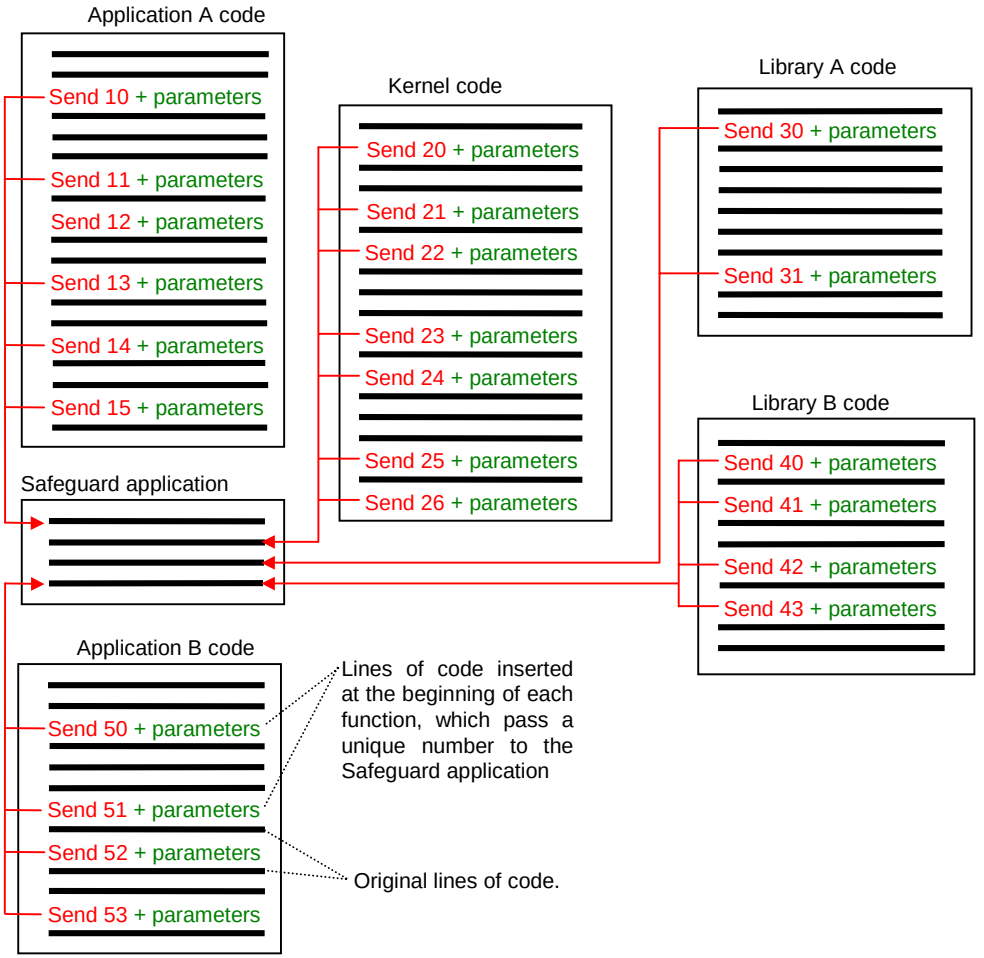
Not a fortress architecture

Examples of anomaly detection

- Safeguard is MUCH broader than anomaly detection
- But will illustrate some of intended use the novel approaches around anomaly detection

- Alphabet analysis will look at *which* functions are called in the normal operation of the system.
- Time sequence analysis will look for *relationships between* the functions that are called.
  - Special case Event Course Analysis
- Analysis of the *values* of the parameters passed to discover invariants

Software instrumentation



Example function call data

System events	Example data
Sequence of function calls within application A.	10 12 10 14 13 15 12
Application A calls a function in the operating system kernel.	10 12 22 23 13 15 10
Switch between application A and application B caused by time slicing in the kernel (25 and 26 are the kernel functions responsible for time slicing).	10 12 25 26 51 53 53
Switch between application A and application B caused by a user action (21 and 20 are the kernel functions responsible for context switching).	10 12 21 20 51 53 53

Alphabet analysis

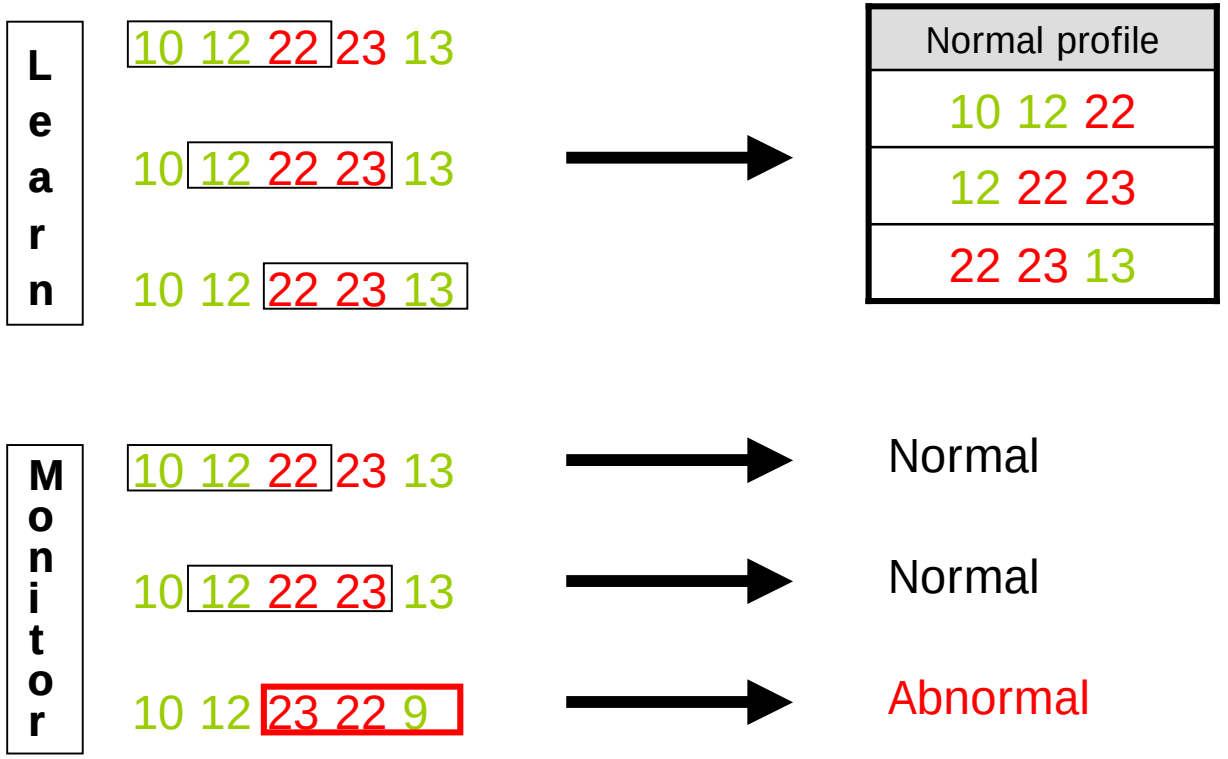
- Simply look at which functions are being called
- Intruders often use unusual functions, e.g. Telnet, compilers, etc.
- Redundant functionality in modern complex software
- By querying unusual functions, a bloated operating system can be hardened down into a thinner more survivable operating system

Time sequence analysis

- Build up a profile of the system's behaviour by monitoring the sequence of function calls
- So identify anomalous use of the program and unusual interactions between the program and the operating system
- The way in which a program is used will vary from operator to operator

Time sequence analysis (2)

- Stephanie Forrest - pH



Analysis of values

- Program invariants are properties that are true at a particular program point
- Invariants can help us know if the system is being misused (e.g. by insiders) or data is erroneous
- Trivial examples:
  - length of string p is < 12 characters (buffer overflow!)
  - x > 0
  - x + y = 5
  - Can be application dependent
    - E.g. Kirkchoff's rule will apply in electricity networks
- Good idea but how to find them!
- Learn context dependent invariants as the program is running as it should
  - Extending ideas of Michael Ernst's Daikon system
- Monitor them at run time
  - Send deviations from normality to the anomaly detection

after adding learned assertion

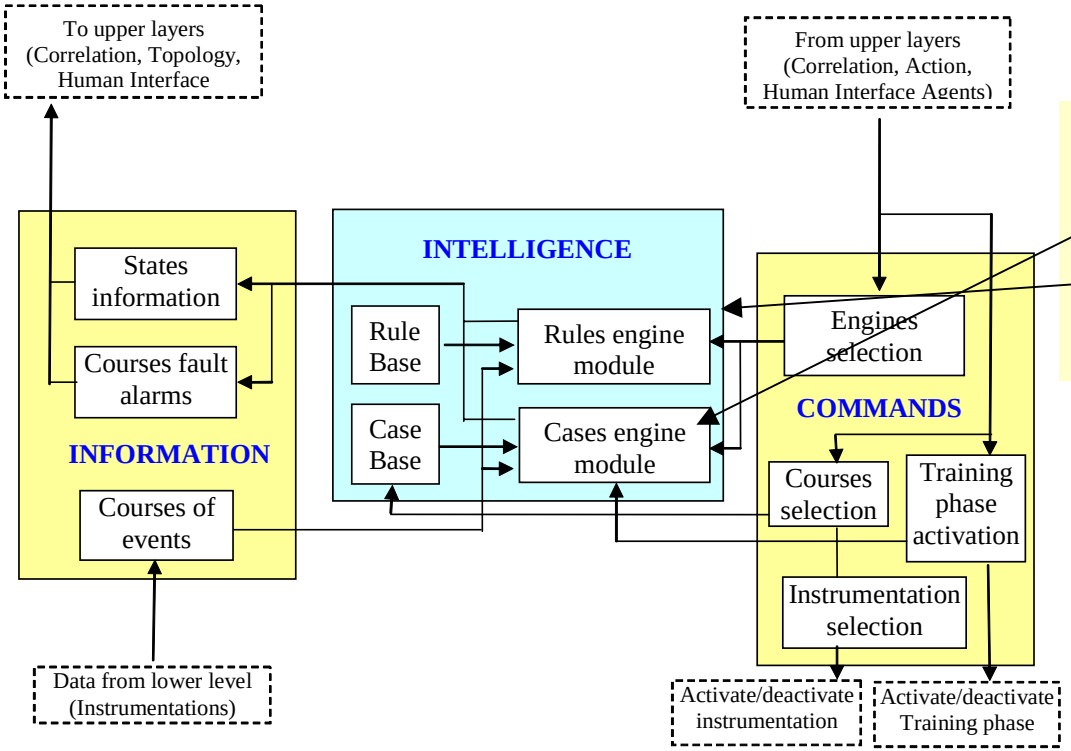
```
boolean compare(int n, float f){
assert f>0;     i.e send message to listener if violated
    boolean b;

    if (n >= fa.length){   n = fa.length - 1;}
    if (fa[n] > f){  b = true;  }
                    else{ b = false; }

assert f>0;
    return b;
  }
```

Pre-conditions & post conditions

Architecture of an anomaly component



Dual components give

completeness + speed

Simple instrumentation – monitoring courses of events

What does this approach promise

- .....that the existing approaches do not achieve?

- Trustworthiness
  - Attempts to look at many of the factors and correlate these
    - Not just intrusions, but also diagnosis, anomalies, current state estimates
  - Not orthogonal to the state estimation which is a key part of any controlling system
    - many IDS systems don't look at data

Promise

- Scalability
  - Normality is inherently easier to define
    - but still difficult
  - Software sensors give more information that allow the decision making to be simpler
    - Intelligence without data just gives a huge search space and myriad of possible causes
  - The “learning” approaches are intended to be simple based on techniques such as CBR and elimination (invariants)
    - Also learning is local which reduces the complexity

Promises

- Adaptability
  - The emphasis is on learning normality (and in an incomplete way cases of abnormality)
  - Potential to recognising novel anomalies
  - When structure of system changes just train the system again (ideally)

Can these promises be achieved?

- too early to tell!

Cost

- The instrumentation
  - Access to the source code is problematic
    - Looking at modification of binaries
    - Looking at approached like wrapping of classes with decorator classes in Java
  - But for much we only need module to module communication which may well be obtainable
    - Need to establish how to instrument to get adequate data
- The performance degradation
  - Of the O/S : this can be expensive in operational time but we believe this can be limited by
    - selection of locations to apply and tuning
  - Of the application : this is not such a problem as very little code need be inserted and checks are fast and performed in parallel – even elsewhere

Challenges

- To be usable the system itself must be robust and not need perpetual readjustment of deployment descriptors.
  - In large systems there is always something changing
  - Keep the dependencies clear
- and secure
  - E.g. Not build on “agent platform” but on more secure and scaleable enterprise middleware
    - Can emulate ACL and agent communication with message passing EJBs
- Really too big an objective
  - So all help and suggestions welcome

Questions?