# Dependability and Accountability for Context-aware Middleware Systems

Andrew Rice

Computer Laboratory

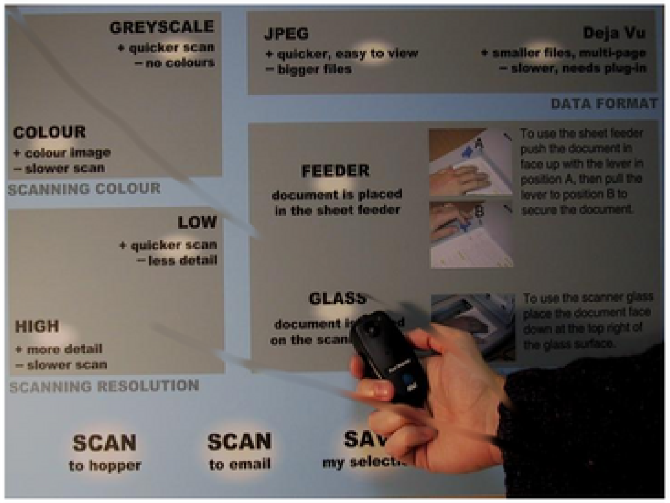
University of Cambridge

Digital Technology Group

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

## Distributed Middleware

- Middleware for sentient computing are inherently distributed
  - Many connected devices with wide-ranging capabilities
  - Multiple sources of context


GREYSCALE
+ quicker scan
– no colours
JPEG
+ quicker, easy to view
– bigger files
Deja Vu
+ smaller files, multi-page
– slower, needs plug-in
DATA FORMAT
COLOUR
+ colour image
– slower scan
SCANNING COLOUR
FEEDER
document is placed in the sheet feeder
To use the sheet feeder push the document in face up with the lever in position A, then pull the lever to position B to secure the document.
LOW
+ quicker scan
– less detail
GLASS
To use the scanner glass place the document face down at the top right of the glass surface.
HIGH
+ more detail
– slower scan
SCANNING RESOLUTION
SCAN
to hopper
SCAN
to email

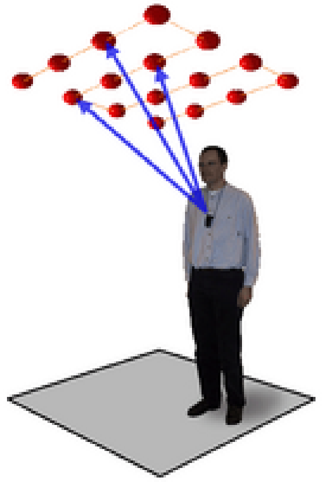
## Complicated Middleware

- Middleware for sentitent computing are inherently complicated
  - Computationally intensive tasks are performed on behalf of low-power devices
  - Shared processing steps are made available to many applications

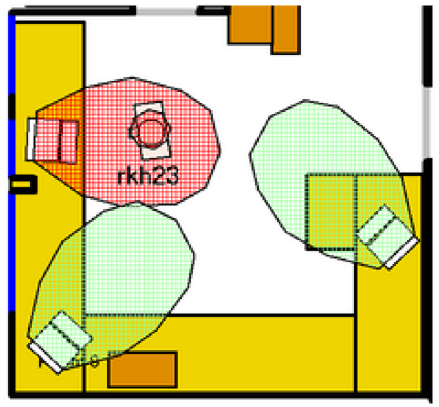
## Examples

- SPIRIT + Active Bat system

Distributed access for applications using CORBA



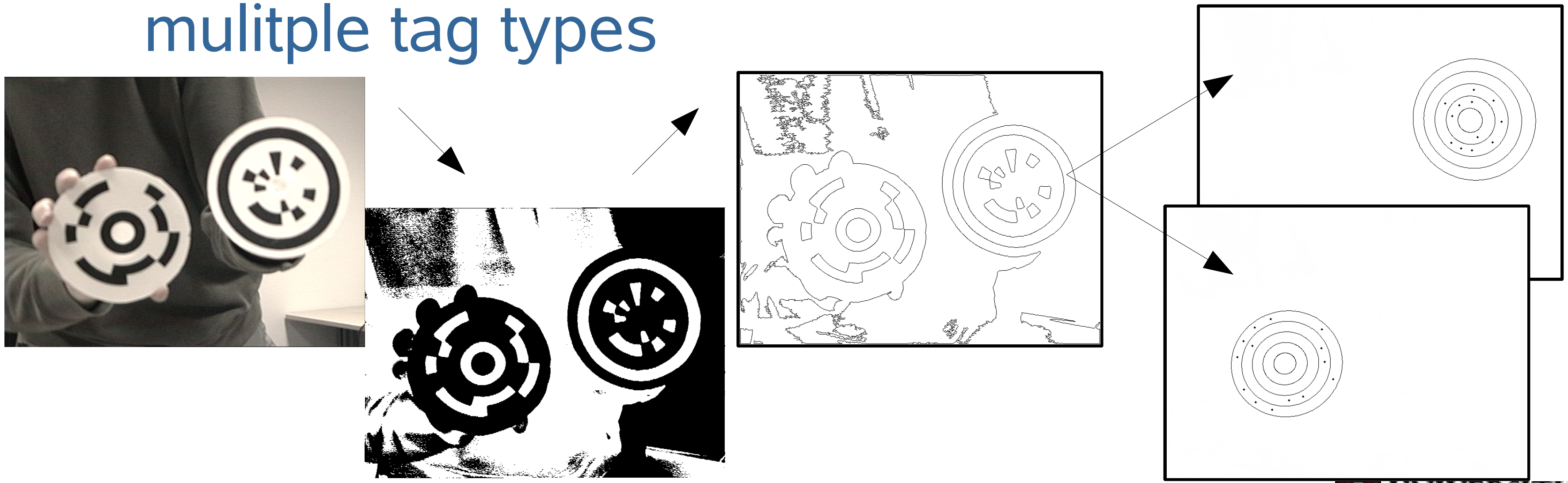
Multi-lateration on raw bat sensor readings to derive a location



Monitors predefined regions for containment events and provide call backs to applications

## Examples

- Cantag (image processing framework)
  - Supports distributed image pipelines
    - increase throughput
    - off-load complicated stages
  - May share computation steps when tracking mulitple tag types

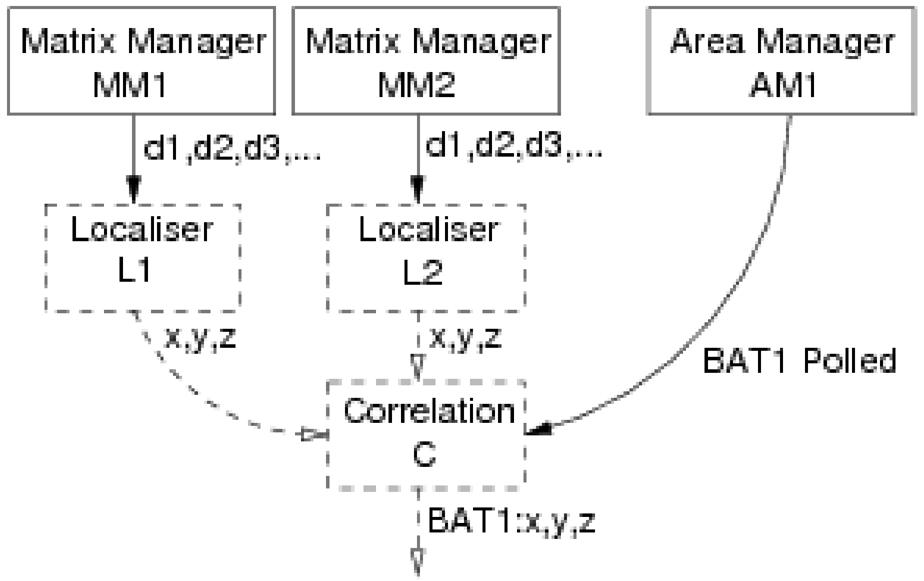


## Validation

- Many operations performed by the middleware are asymmetric
  - difficult to compute but cheap to check
- Multi-lateration, containment monitoring, shape-fitting are all examples of this

## Concept

- Define a set of trusted components



- Decompose remainder of system into untrusted components
  - check the outputs against the input data

## Positive Validation

- Given an item of data from the middleware we can check  that there is supporting evidence
- Example: Whenever the Bat system reports a sighting we check that it is correct

# Positive Validation

## Application: Tell me acr31's position

$c_1 : t,d_1$

$c_2 : t,d_2$

$c_3 : t,d_3$

$c_4 : t,d_4$

$c_5 : t,d_5$

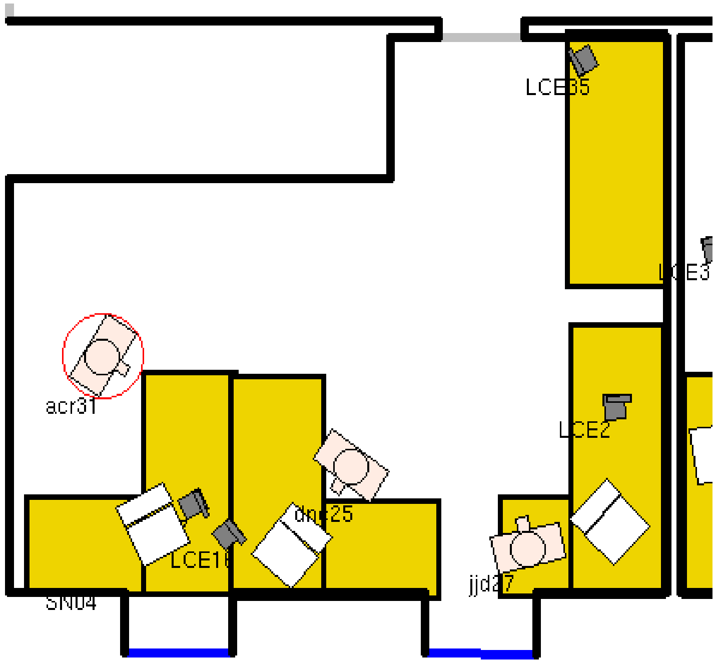
$c_6 : t,d_6$

$\longrightarrow$

$s_1 : t,\text{'acr31'}$

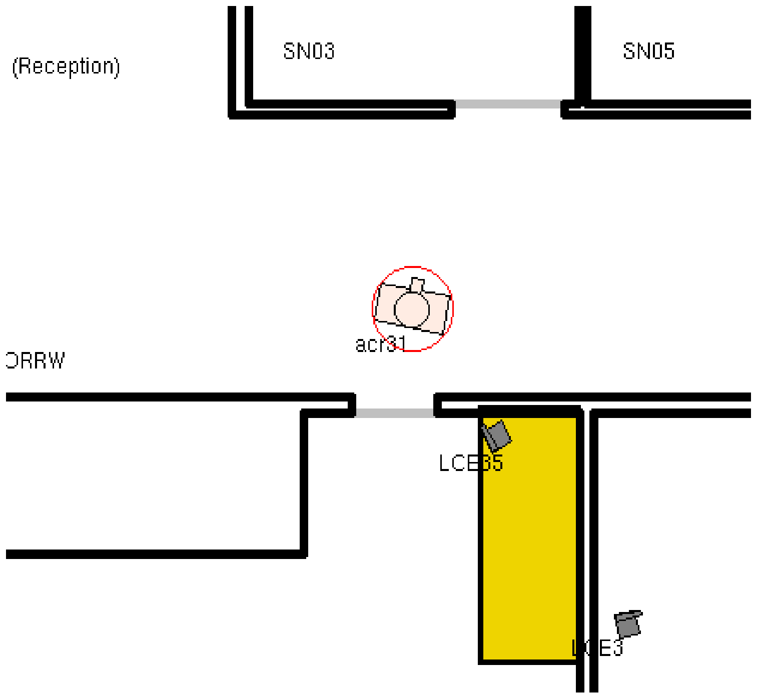
$l : t,x,y,z$

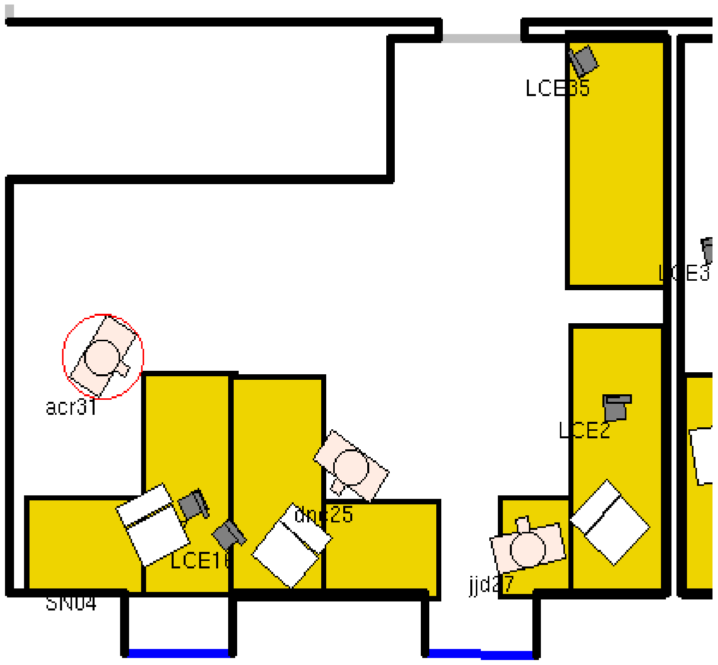
$\longrightarrow$



# Positive Validation Shortcoming

## Application: Tell me when acr31 enters SN04





Prove: acr31 not in SN04 at time $t_1$

Prove: acr31 in SN04 at time $t_2$

Also must show that acr31 not sighted between $t_1$ and $t_2$

## Negative Validation

- Need to demonstrate a lack of sightings
- This can be done by coupling a sequence number when the Bat is polled
  - (we have to trust the polling component)

Application: Tell me when acr31 enters SN04

Prove: acr31 not in SN04 for poll('acr31',n)

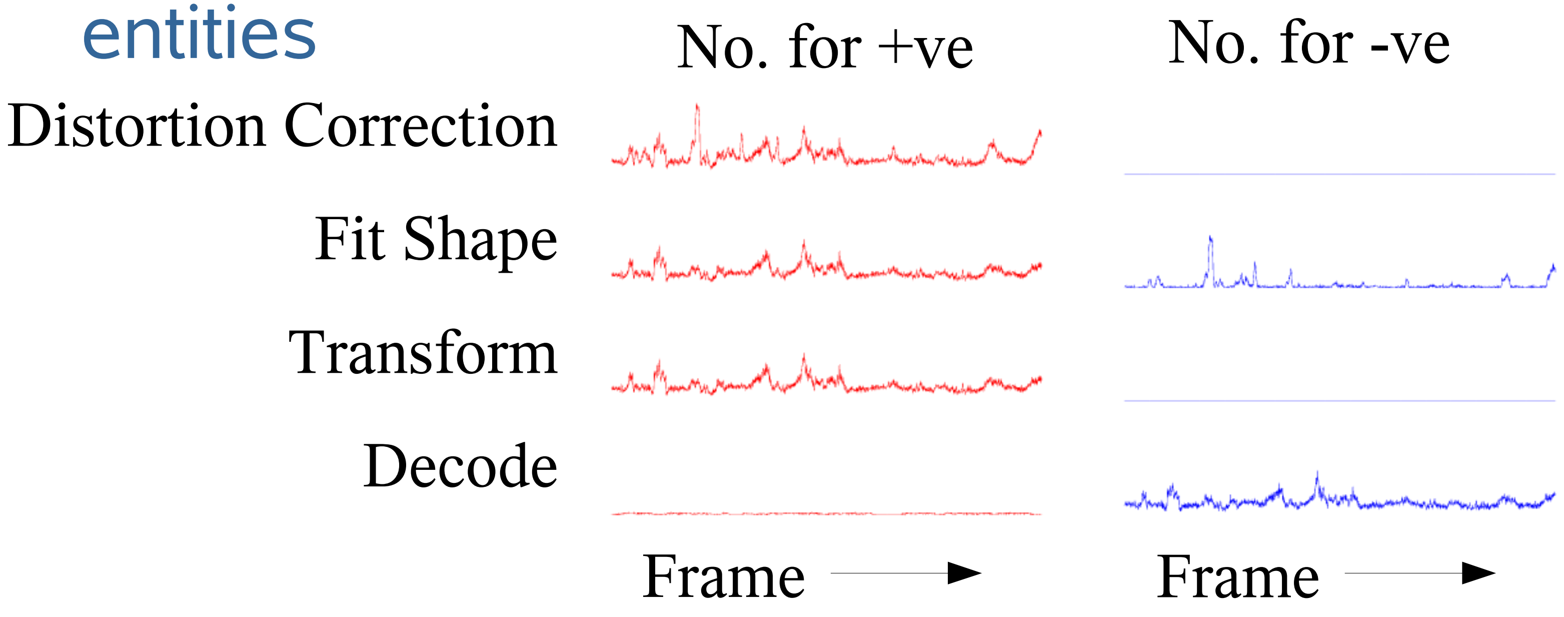
Prove: acr31 in SN04 for poll('acr31',n+1)

## Negative Validation

- This approach doesn't work for unpolled systems
- Example: Cantag may sight many tags in a single frame (or none at all)
- Negative validation in this scenario requires checking discarded data
- Example:  Shape fitting
  - Fitted shapes close to contours? -  +ve validation
  - Contours for unfitted shapes should have been discarded? - -ve validation

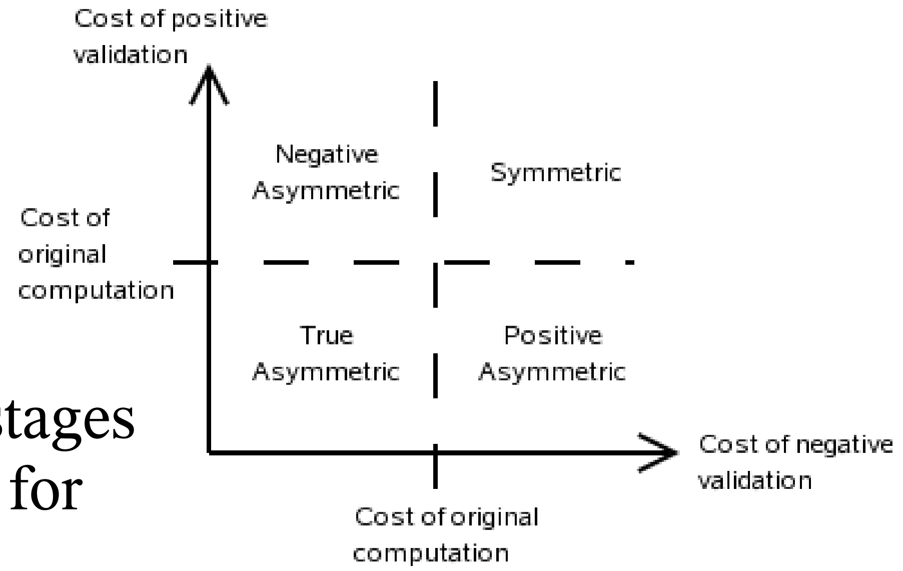
# Costs of Validation

- Implemented validation for Cantag
- Huge variation in the runtime cost for validation
  - this is due to large variation in the number of entities



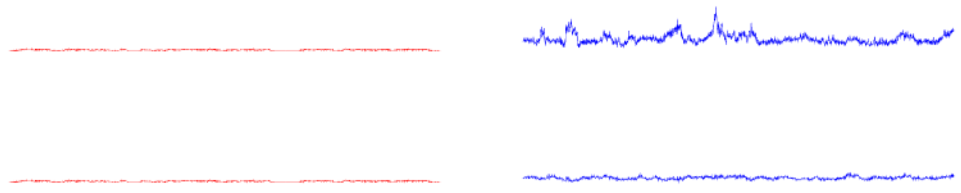
# Costs of Validation

- Cost of checking each stage can be used to classify each function


Cost of positive validation
Negative Asymmetric
Symmetric
Cost of original computation
True Asymmetric
Positive Asymmetric
Cost of negative validation
Cost of original computation

Positive asymmetric functions are common.

The addition of heuristics in early stages can reduce the number of elements for checking in expensive stages.



## Evaluation

- Validation is a mechanism for exposing the state of the system to allow applications to validate behaviour
- Checking each stage requires a application specific check – no sensor system abstraction
- Every piece of data might not need checking
  - infrastructure required to support this
- Bugs were caught in Cantag – it does work!

## Future Work

- Longterm deployment of validation in a real sentient middleware (as opposed to small tests)
- General interfaces for applications to request validation data and acquire validation algorithms