



**Imperial College
London**

Distributed Software Engineering (DSE) Group
Department of Computing

Access Control in Publish/Subscribe Systems

Jean Bacon
David Eyers
Jatinder Singh

{firstname.lastname}@cl.cam.ac.uk

Opera Group
Computer Laboratory
University of Cambridge

Peter Pietzuch

prp@doc.ic.ac.uk

Distributed Software Engineering
(DSE) Group
Department of Computing
Imperial College London

1. Introduction

Jean Bacon

Jean.Bacon@cl.cam.ac.uk

Opera Group
Computer Laboratory
University of Cambridge

Introductions and Biographies

Jean Bacon

- Leads Opera Research Group (University of Cambridge)

Peter Pietzuch

- Past PhD at Opera, post-doc at Harvard
- Lecturer at Imperial College London

Dave Eyers

- Past PhD at Opera
- Post-doc at Opera

Jatinder Singh

- PhD at Opera

Overview of Tutorial

1. Introduction

- Research background

2. Client access control using RBAC

3. Event type naming and version control

Intermission

4. Broker network security

5. Application scenarios and interaction control

Open discussion

Setting the Scene

Large-scale distributed systems over large geographic areas

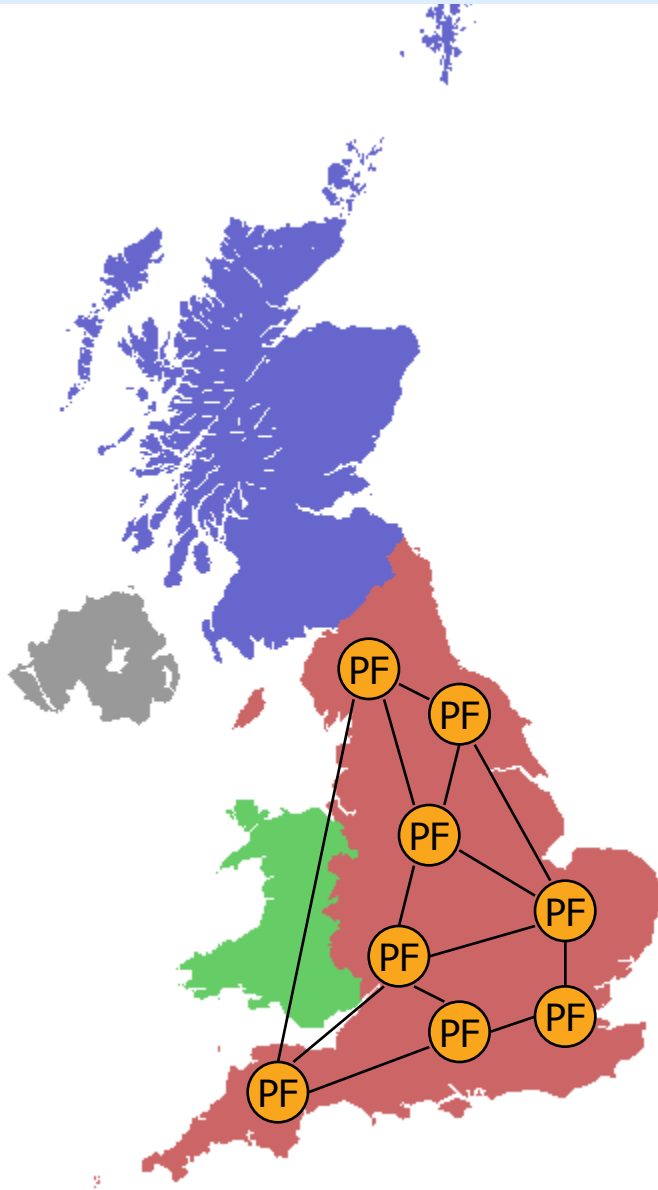
- Multiple independent administrative domains
- Varying levels of trust between domains

Both private and shared applications

Access to the shared **infrastructure** and private and shared **applications** needs to be controlled

- who can advertise and publish (no spam!)
- who can subscribe (and see event content)

Motivation - Environment



Police IT Organisation

- 40+ autonomous domains: county police forces

Domain-specific name spaces and applications

Use court-case system and DVLA (drivers/vehicles licensing)

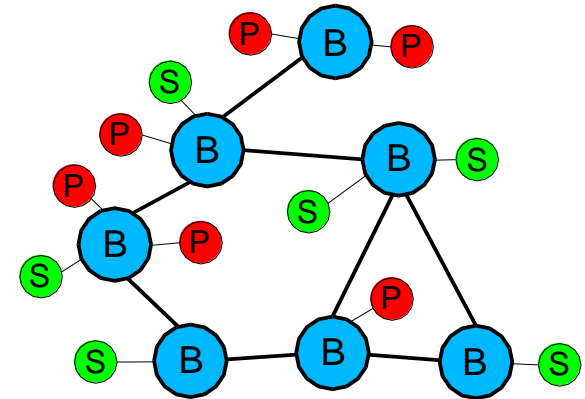
Hermes Publish/Subscribe

Developed in the Opera group (Peter Pietzuch)

- Content-based P/S middleware
- Supports strong event typing

System consists of event clients (publishers or/and subscribers) and event brokers

- Clients connect to broker (PHB, SHB) to access P/S system
- Broker network built on distributed hash table (DHT)



The dynamic, unlocalised nature of DHTs makes access control difficult

Access Control – Previous Work

Application developers use OASIS RBAC to express authorisation policy for service use, including the P/S service, within and between domains

Assuming RBAC authorisation policy is satisfied, event clients, and the event brokers they connect to, are issued credentials that authorise them to access event types in order to publish and/or subscribe

This is transparent to the end users – event clients act on their behalf

OASIS ref: ACM TISSEC Nov 2002, ref 1 for secure P/S

2. Client Access Control using RBAC

Peter Pietzuch

prp@doc.ic.ac.uk

Distributed Software Engineering
(DSE) Group
Department of Computing
Imperial College London

Access Control for Pub/Sub

Goals of pub/sub and distributed access control not entirely complementary

- Access control wants detailed information about users
- Pub/sub optimises operation by not knowing about users

Want architecture to provide security with minimal overhead

- Ideally pub/sub clients might be access control agnostic

Policy-based access control over advertisements and subscriptions rather than event publications themselves

- Advertisements and subscriptions relatively infrequent
- Events exist within type system
- We can trust event brokers (for the moment...)

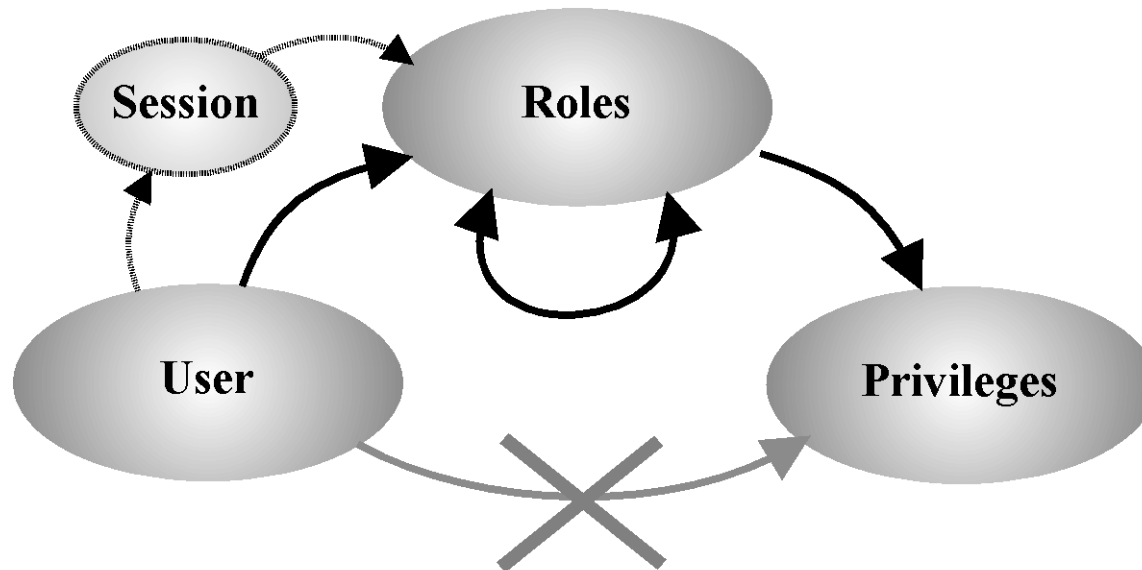
Role-Based Access Control (RBAC)

Abstraction between principals and privileges

- Eases administration by providing grouping

Some RBAC models are session-based

- NIST RBAC requires role activation within session



OASIS

Session-based, parameterised, distributed RBAC

- Architecture developed at CL
- First appeared around 1995 (before other OASIS in 1998)
- J2EE health record demonstrator

Horn-clause rule structure

Role activation rules:

$$r_1, r_2, \dots, r_{n_r}, ac_1, \dots, ac_{n_{ac}}, e_1, \dots, e_{n_e} \vdash r$$

Privilege authorisation rules:

$$r, e_1, \dots, e_{n_e} \vdash p$$

Need to evaluate constraints on role activation

- OASIS provides inference mechanism

Support for event-driven security

- Prerequisite roles can be revoked using event channels

Edge Control in Broker Network

Security with minimal overhead to pub/sub efficiency and scalability

- But need to provide necessary flexibility

Perform access control checks at

- | | |
|------------------------|-----------------------------------|
| Client connection time | → Client connection policy |
| Advertisement time | → Advertisement policy |
| Subscription time | → Subscription policy |
| Event type management | → Type management policy |

Use RBAC to achieve scalability and flexibility

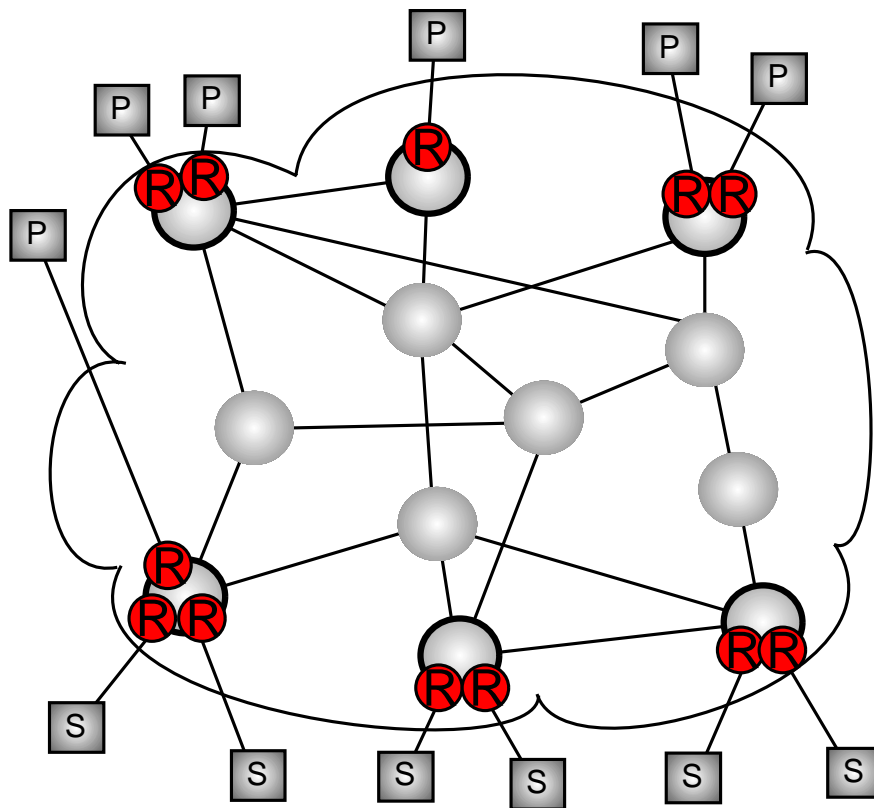
- Investigated tight role-type integration but doesn't satisfy real-world requirements

General Architecture

Only edge brokers need to check policy

- No overhead at publication time

Introduce **restrictions** for advertisements and subscriptions



Submitted subscription:
AmbulanceEvent(loc = any)

Added restriction:
AmbulanceEvent(loc = London)

Restrictions

For type-based restrictions, we get access control for free from the pub/sub system

- Otherwise predicate may need to be verified per event

1. Generic non-optimised restriction predicate

- Use any black box predicate
- May be expensive to evaluate

2. Pub/Sub restriction predicate

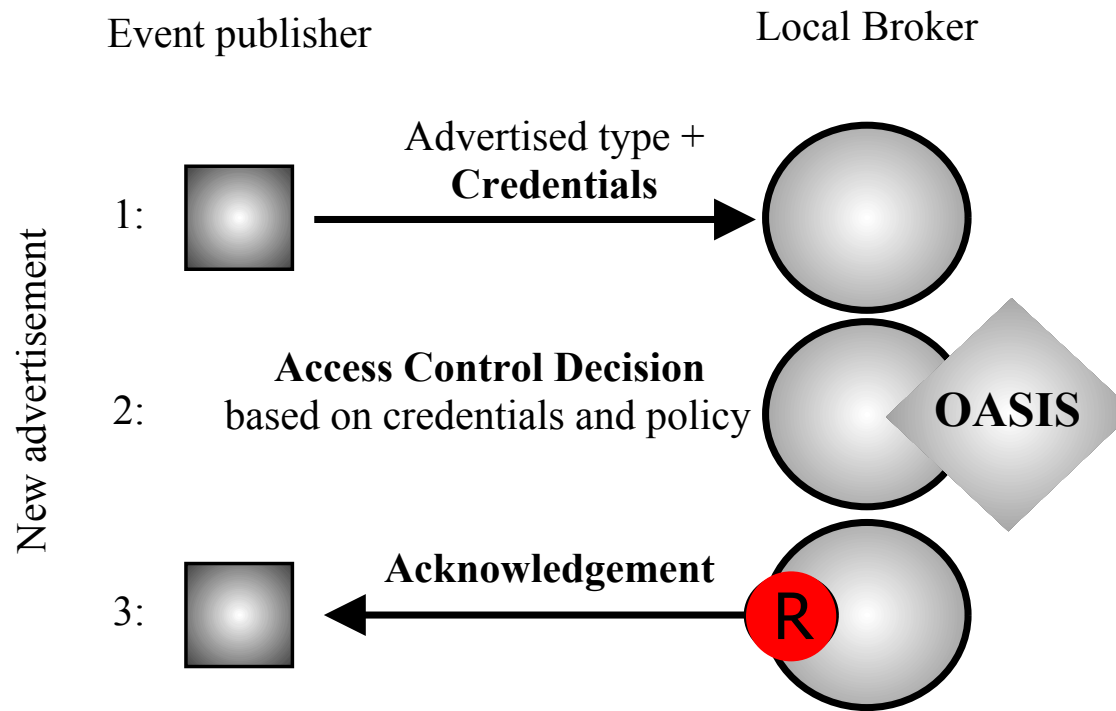
- Use filters available by pub/sub system
(e.g. `AEvent[loc=any]` AND `AEvent[loc=London]`)
- Get access control for “free”

3. Hybrid schemes

- Combination of both

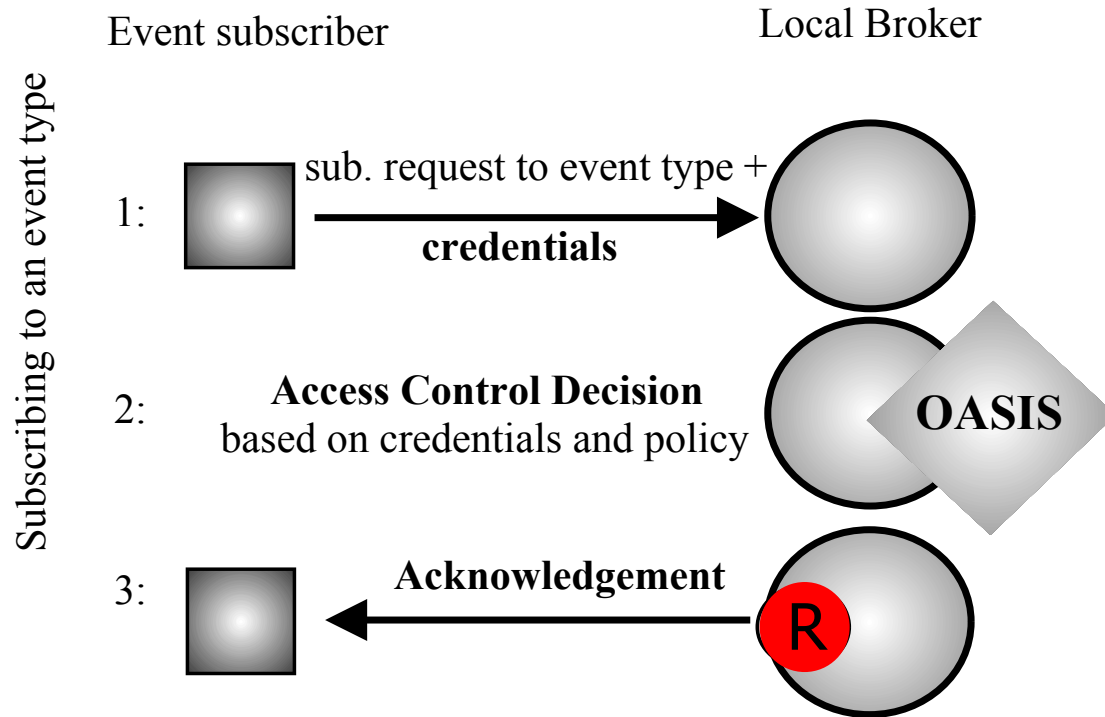
Advertisements with Access Control

The steps for advertisement are as follows:



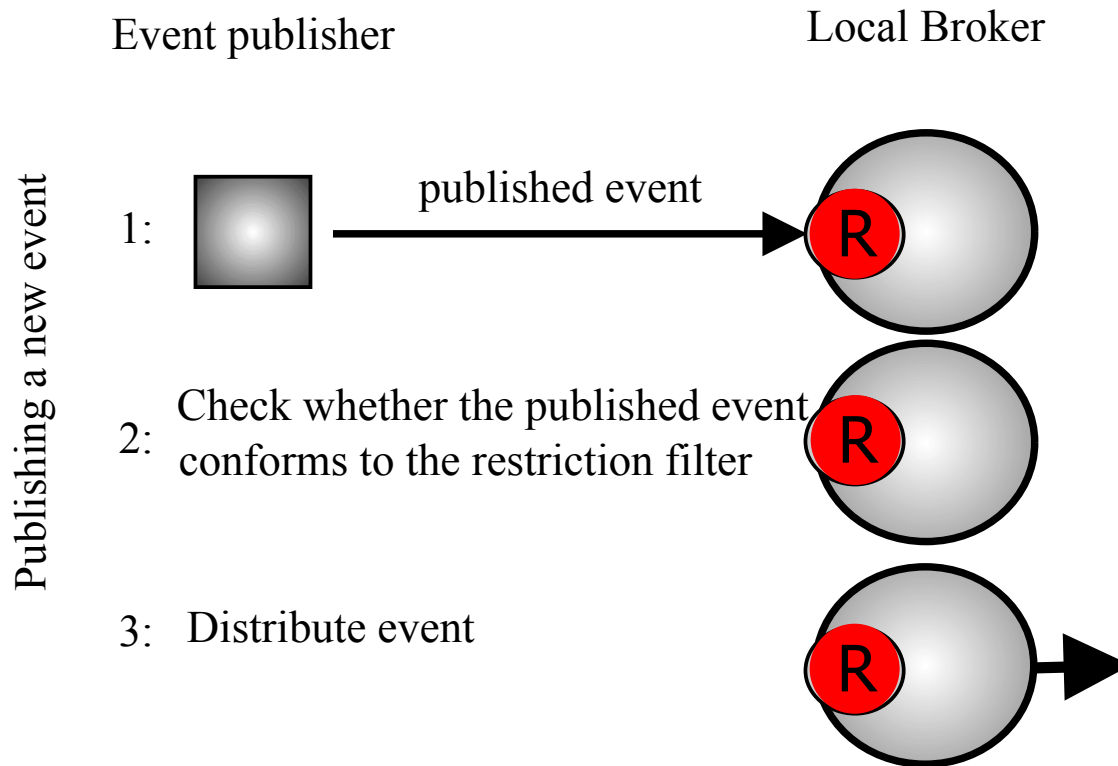
Subscriptions with Access Control

Likewise for subscriptions:



Publishing Events

Subscriptions are passive, but advertisements require us to look at what actual events are published:



API of an Oasis-Aware Hermes Broker

<code>connectPublisherToBroker</code>	<code>(publisher, creds)</code>
<code>disconnectPublisherFromBroker</code>	<code>(publisher, creds)</code>
<code>connectSubscriberToBroker</code>	<code>(subscriber, creds)</code>
<code>disconnectSubscriberFromBroker</code>	<code>(publisher, creds)</code>
<code>addEventType</code>	<code>(typeowner, creds, eventType)</code>
<code>removeEventType</code>	<code>(typeowner, creds, eventType)</code>
<code>subscribeType</code>	<code>(sub, creds, eventType, callb)</code>
<code>unsubscribeType</code>	<code>(sub, creds, eventType)</code>
<code>subscribeTypeAttr</code>	<code>(sub, creds, eventType, filter, callb)</code>
<code>unsubscribeTypeAttr</code>	<code>(sub, creds, eventType, filter)</code>
<code>advertise</code>	<code>(publisher, creds, eventType)</code>
<code>unadvertise</code>	<code>(publisher, creds, eventType)</code>
<code>publish</code>	<code>(publisher, creds, event)</code>

Conclusions

Need suitable access control model for pub/sub

- RBAC provides expressiveness and flexibility in distributed setting without centralised control

Restrictions: leverage pub/sub system for a/c enforcement

- Keeps publication overhead down

Edge access control efficient and flexible, but:

- Assumes trusted broker network
- Doesn't provide fine-grained access control to event attributes

➡ Need better model of event type in pub/sub system

3. Event type naming and version control

David Eyers

David.Eyers@cl.cam.ac.uk

Opera Group
Computer Laboratory
University of Cambridge

Event type naming and version control

Wide area pub/sub requires a distributed naming system

- No single point of administration (i.e. multiple domains)
- Parallels X.509 certificates in HTTPS (i.e. multiple CAs)
- Protect type definition and version, not just type name

Each domain provides a policy management interface

- Local scope for policy (assume domain names are unique)
- Role activation policies
- Service authorisation policies

Event type evolution

- Facilitate modification of type definitions post-deployment

Typed Publish/Subscribe

Building on Hermes

- Strongly typed events

Much of this work stems from Lauri Pesonen's Ph.D. research

Event type definitions: type name and a set of attributes

- Each attribute consist of an **name** and a **type** (e.g. **integer**, **string**)
- Local brokers do type-checking

Types are stored in a type registry

- At the *rendezvous* node in Hermes

Type issuers issue new types

Type managers manage existing types

Evolution from previous pub/sub research

Type definition and type name bound together securely

- Public key cryptography verifies authenticity and integrity
- Protection against forgery / tampering
- Reduce name collision risks

Components of type definitions uniquely identifiable

- Provide handle for policy reference
- Securely name individual type attributes

Type management decoupled from pub/sub system

- Allows for delegated event type management
- PKI certificate chains can operate out of band

Secure event type definitions

Event type issuers must generate a key pair

- Public key is included in type name (removes collisions)
- Private key signs type definition
- Forms an ownership/naming scope

User-friendly name also maintained

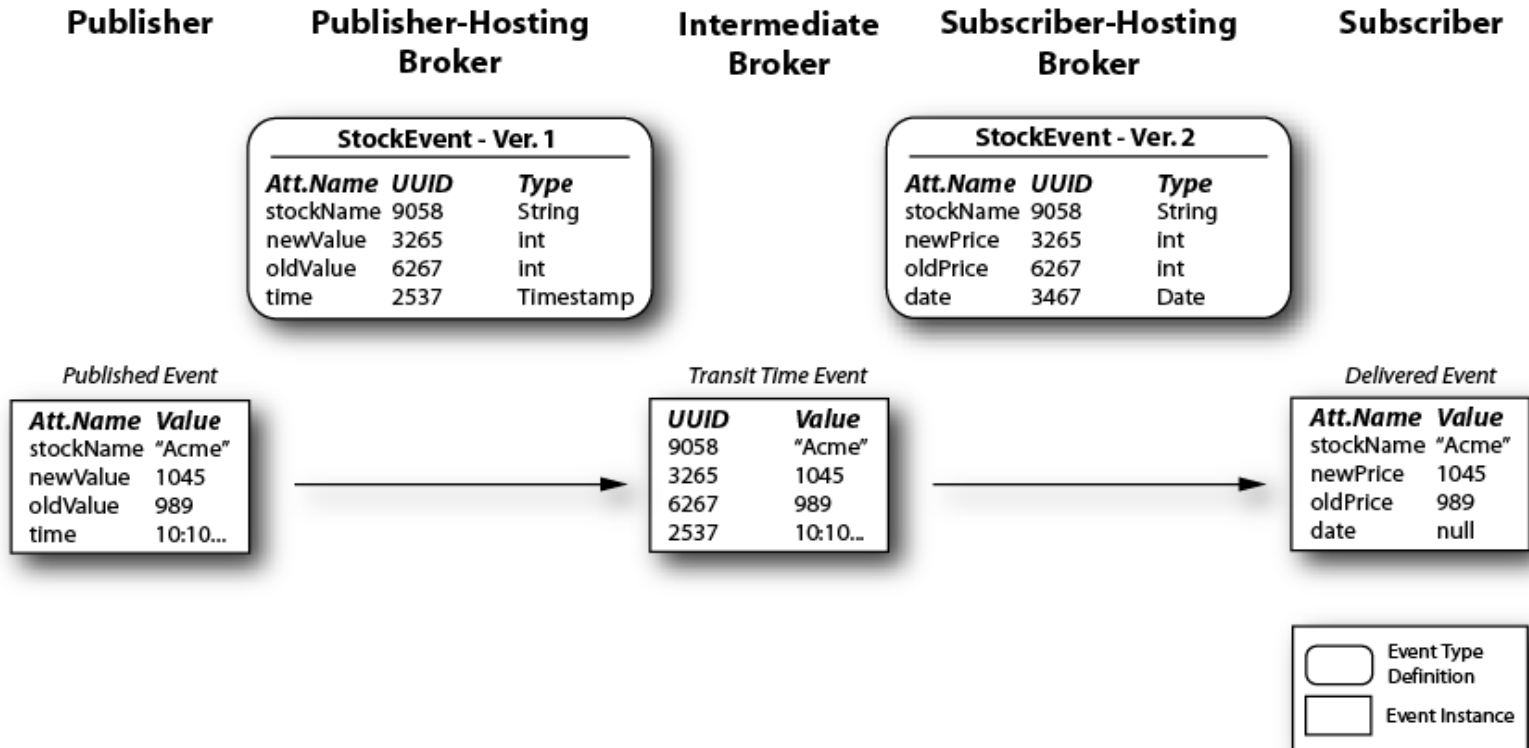
- Easier to manage by humans (e.g. in source code)
- Can encode hierarchical naming schemes

Version number included

- Avoids race conditions by using UUIDs

Event type versioning

- Version numbers allow for dynamic event type evolution
- Potential for automatic attribute transformation



Secure event type definitions (cont.)

Name tuple:	1 Type issuer's public key
	2 User-friendly name
	3 Version number
Body:	4 Event type's attributes
Digital Signature:	5 Delegation certificates
	6 Digital signature

“Body” defines each event type's attributes including:

- Friendly name (used by clients)
- UUID (used by intermediate brokers)
- Attribute type (depends on filter language)

Signature calculated over all name and body elements

Domain hierarchies

Domains may be grouped

- Parent domain provides initial role names and policies
- Domains share trusted root certificates

Many deployments must balance local and global policy

- Domains can augment event types
- Parameterised roles can incorporate domain-specific data

Owners issue type modification rights, e.g.

- Add or remove attributes
- Edit attribute names
- Edit attribute type

4. Broker network security

David Evers

David.Evers@cl.cam.ac.uk

Opera Group
Computer Laboratory
University of Cambridge

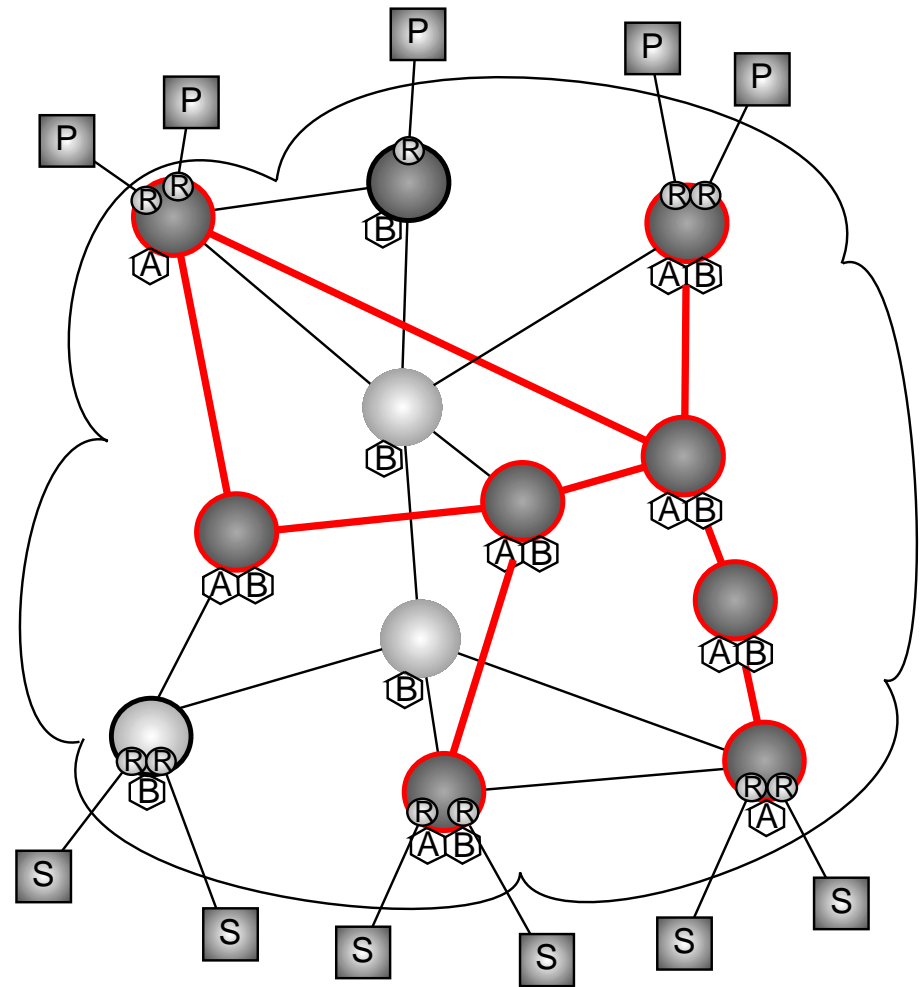
Motivation: multi-domain broker networks

Broker trustworthiness may vary across domains

- Infrastructure owned by different organisations
- Varying security levels

Legal implications of data handling

- Avoid seeing data to avoid responsibility



Motivation: multi-domain broker networks

Combine brokers to increase resilience in the network

- Operating in difficult environments
- Desire to operate is shared
- Data is independent

Implications of trust in federated system

- Fairly untested in the legal system so far...

More on “untrusted” brokers

- Brokers probably need to be trusted to route data

Delegated authority

Resource owners grant delegation certificates to the Access Control Manager (ACM) of target domains.

- Separates inter- and intra-domain policy enforcement

Resource owners must trust the domain ACMs

- However ACMs' certificates can be revoked

Client requests are coupled with ACM-issued certificates

- ACM links resource owner to client

Coordinating domain

- Responsible for managing broker enrolment

Security credentials

Brokers joining pub/sub system

- Coordinating domain issues credentials

New types and topics

- Access right is binary (CD can't check every action)

Extending types and topics

- Type owners rather than CD issue credentials
- Wildcards are supported in names

Accessing the pub/sub API

- Type owners issue delegation to ACS
- Restrictions may be specified (e.g. on attribute values)

Credential Propagation

Client credentials propagate through the broker network

- Credentials carried alongside event data
- Brokers are trusted to pass these events
- Intermediate brokers may verify credentials
- Time-stamp is included
- Credentials and time-stamp signed by client

Use soft-state rather than explicit credential revocation

- Although event channels may be needed in some cases

Encrypting event content

Enforcement achieved using encryption of event data

- Encryption tasks carried out by brokers
- Reduces the number of nodes that have key access

Choice between whole-event and attribute encryption

- Trade off between security and encryption overhead

Encryption keys are changed periodically

- Also changed in response to broker topology changes
- Use one-way function tree (OFT) in prototype $O(\log_2 n)$

Client and local broker independently gain privileges

- Broker's authorisation should encompass client's

Key groups

Client and broker credentials linked to encryption keys

- ... but are decoupled from them
- Support both whole event and attribute encryption

Encrypted data has a UUID

- Selects the cryptographic key used
- Broker “key groups” have access to keys

Key group manager enrolls brokers

- Key group manager must be trusted by type owner
- Usually a member of the type owner’s domain (or TTP)

Event Attribute Encryption

Access control manager
can enforce local policy

- E.g. credentials
required to access
each attribute

Experimented coupling
keys directly

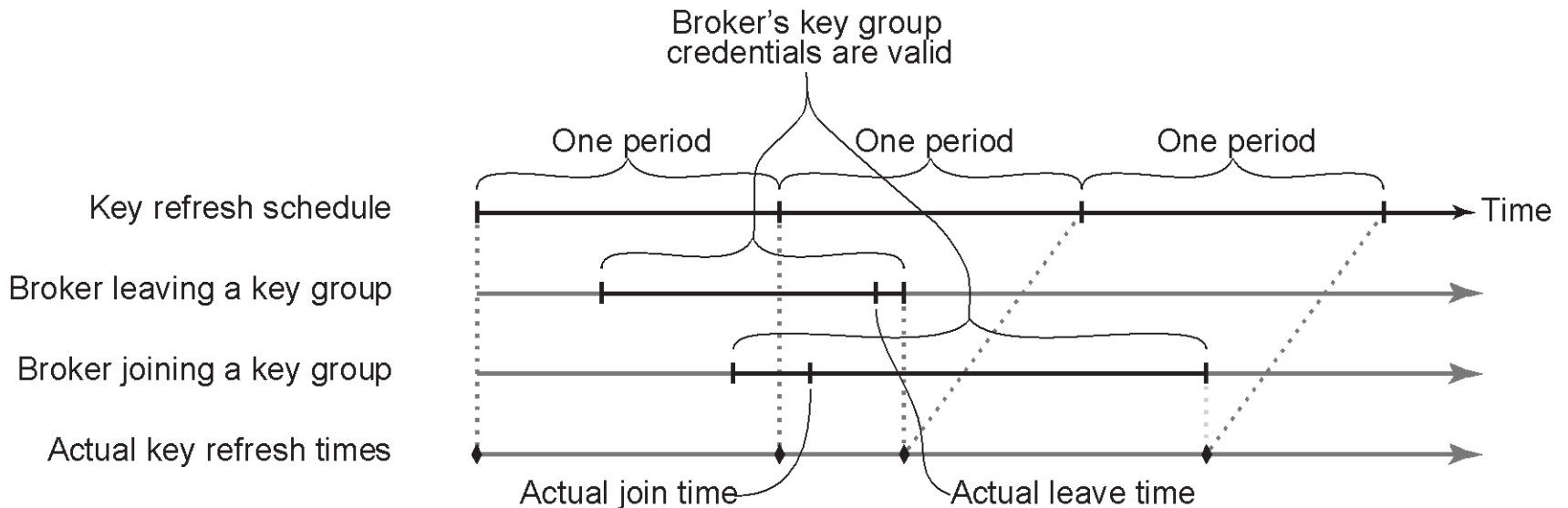
- Becomes messy very
quickly



Optimisations

Reducing the number of key refresh operations

- State of the art secure group communications expensive
- Relate capability validity to refresh cycle



Optimisations (cont)

Reducing number of encryptions

- Encryption costs dominated our experimental results
- Clearly will depend on workload

Plain-text content caches

- Augment events with plain-text attributes
- Cache can be filed on-demand
- Discarded before hop to broker without trust

Cryptographic state

- Avoid re-initialisation of cryptographic state machines

Consequences

Decoupled many components from the pub/sub system:

- Access control policy representations
- Intra- and inter-domain access control decisions
- Type naming, management and certification
- Encryption key groups

Some unintuitive situations can arise ...

- Brokers that cannot perform content-based routing
- Brokers with less privilege than their clients

... but the high flexibility will be necessary in practice

5. Multi-domain Examples

Application of access control mechanisms *and* directions...

Jatinder Singh

Jatinder.Singh@cl.cam.ac.uk

Opera Group

Computer Laboratory

University of Cambridge

Multi-Domain Environments - Recap

Domains

- Partitioned – organisation, region, services, department
- Autonomous (somewhat) - policies, systems, data models

Need to share information

- Asynchronous notification

Trusted environments

- Base-level of control
- Need extra controls due to nature of information
 - Privacy, public interest, national security, legalities...

Consider: Police and Healthcare

Police Information Technology Org. (UK)

British Police Force

- >50 autonomous forces
- Independent decision making
 - E.g. data models, software systems
- Need to communicate, collaborate

PITO

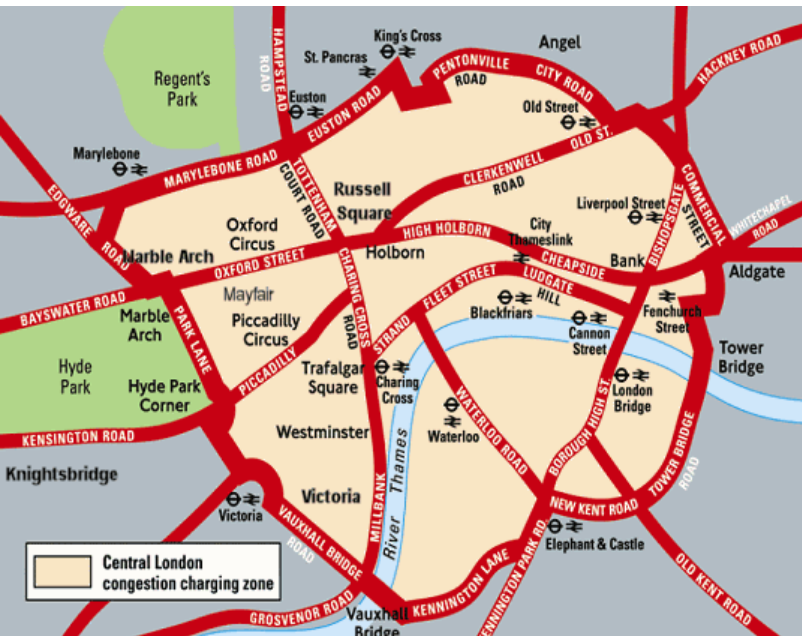
- Improve communications between regional forces
- Responsible for Police data standards
- As a separate domain
 - cf. overarching structure



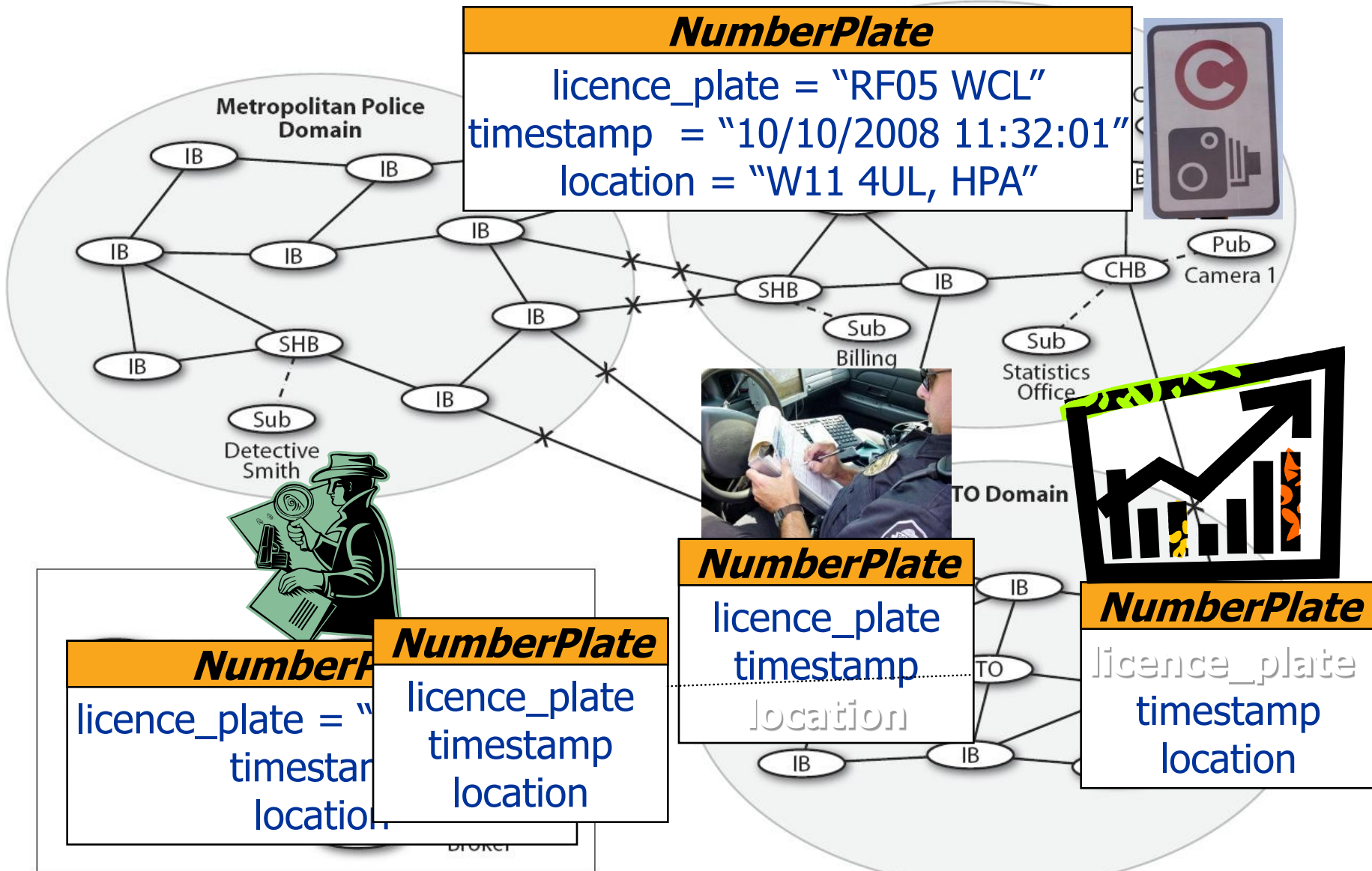
Congestion Charge Scenario

Congestion Charge

- Attempt to reduce traffic through penalties
- Cameras used to photograph cars
 - Numberplate recognition
- Exists in London, other regions interested



Congestion Charge Scenario



Congestion Charge Scenario

CCD cameras can publish *NumberPlate* events

MET domain can subscribe to events in the CCD

MET brokers can receive all information

CCD clients

- Encryption control
 - Billing – can't decrypt *location*
 - Stats – can't decrypt *licence_plate*

Detective Smith

- Capability: *location, timestamp*
- *licence_plate* = RF05 WCL

Information must be shared yet protected

Goal: Share on a **NEED-TO-KNOW** basis

Interaction control:
customising data access to circumstance

Interaction Control

Emerging area

- Policy integration into publish/subscribe frameworks

Typical control:

- Binary access control
- Restrictions concern message instances
- Issues with long-lived, highly-sensitive data

Interaction control

- Customise the data disclosed according to circumstance
 - Context-sensitive
 - *Tailoring* data to the situation
 - *Need-to-know*
- Implemented through data-control policies...

Restrictions

Subscription authorisation

- Authorise event channel
- A **DOCTOR** can subscribe to a **TREATMENT** event where `treats(subscriber_id, sub.patient_id)`

Event restrictions

- Restrict an event instance
- **Dr X** only receives **TREATMENT** events for **Jill** where `ev.condition≠HIV`

Sensitive to context

- Defined on:
 - messaging substrate information,
 - user credentials, and
 - environmental state.
- Restrictions reference state, relationships, qualifications, etc...
- Reactive to change

Transformations

Customising an event to circumstance

Transformation functions

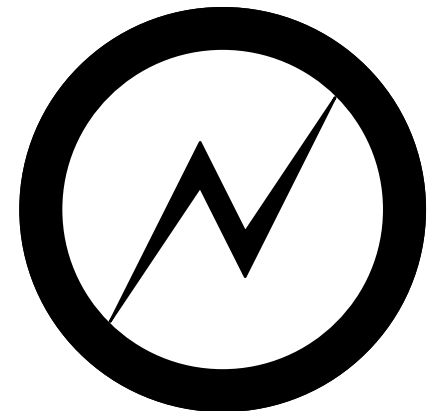
- Altering attribute values of the instance
 - Change granularity, remove/mask
- Produce another event type
 - Enrich, degrade or unrelated event
 - May consume the original

In a broker

- On *publication (receipt)*
- On *notification*

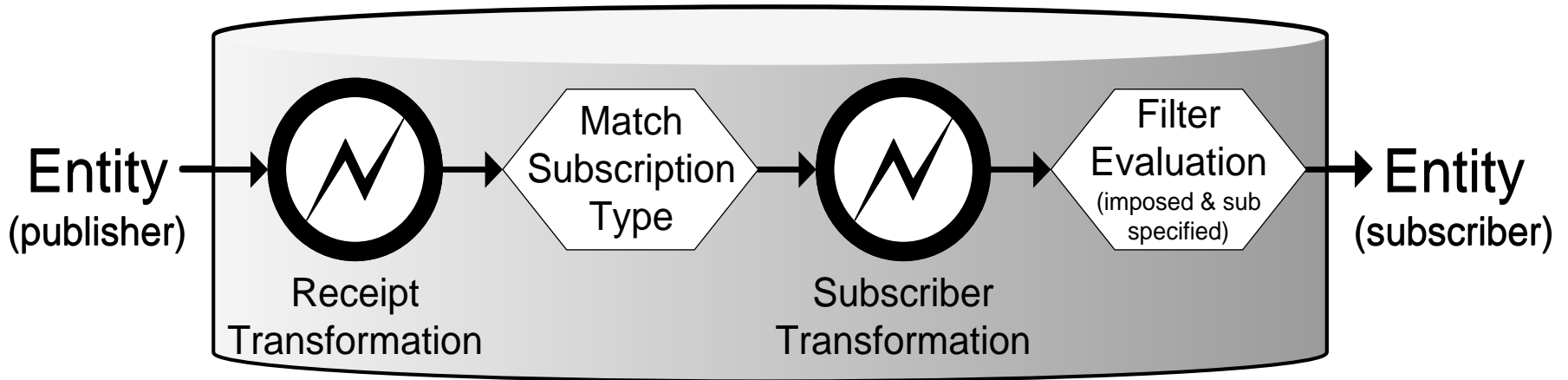
Customisation to circumstance

- Control in line with responsibility



More than binary access control....

Process



Event-Driven Healthcare

Healthcare is a highly collaborative, known environment

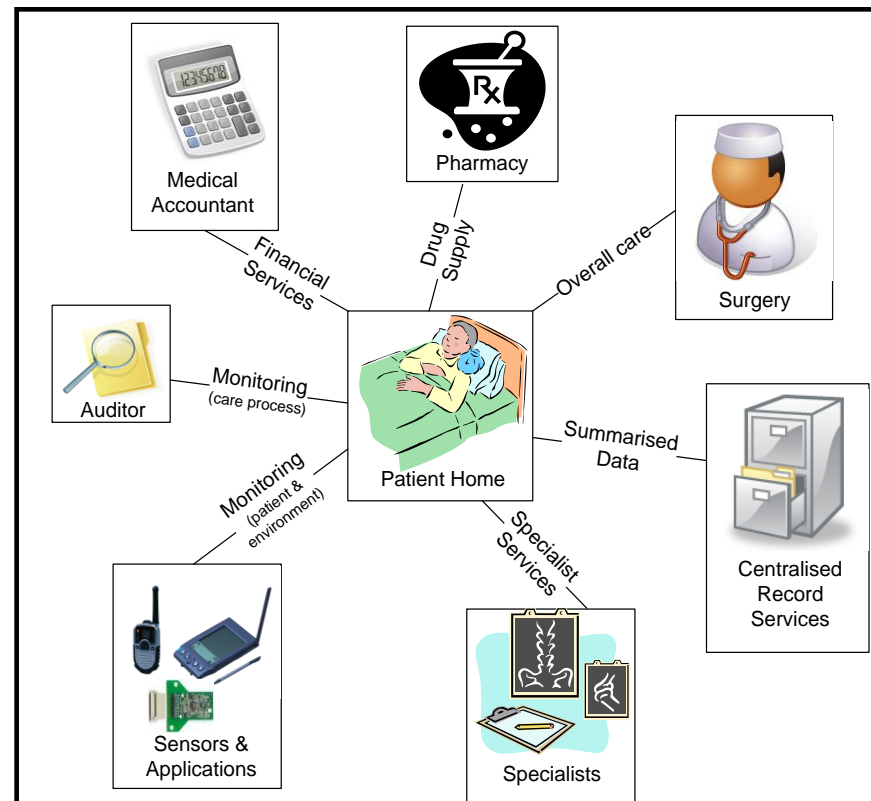
Home healthcare

- Outside of centrally controlled environment
- Favours inter-domain interaction
- Require notification of incidents
 - Actions, observations, state-transitions

Domains and principals

- Share information
- Legally responsible for protecting confidentiality

SHARED and PROTECTED



Home healthcare involves interactions between entities, managed in different administrative domains, delivering specific services as part of the care process

Enterprise Flows: Drug Control

Drug allocation in the UK

Nurses may prescribe drugs

- Restricted drugs in certain situations

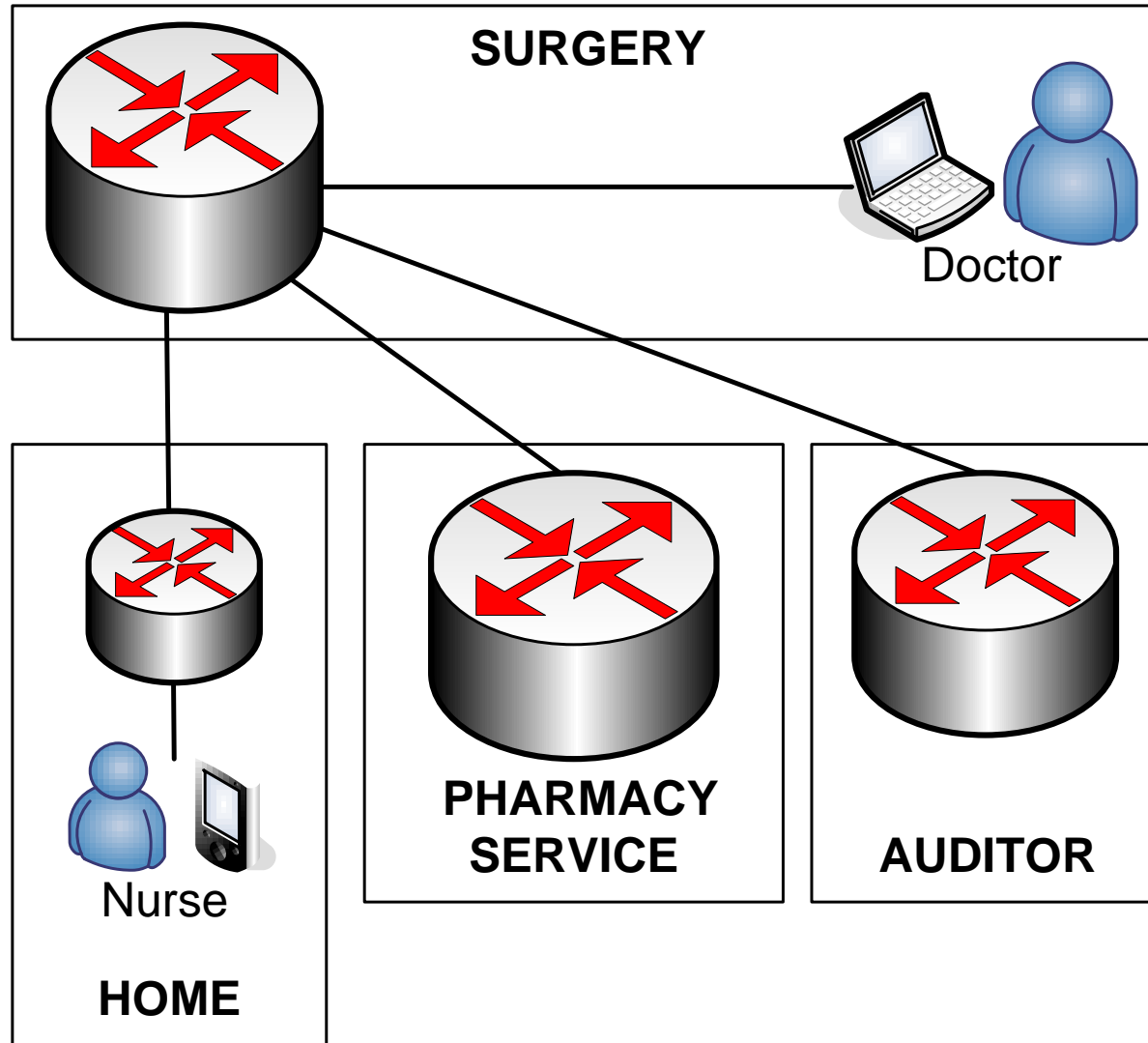
Pharmacy service fills prescriptions

- Prescriptions have a specific form

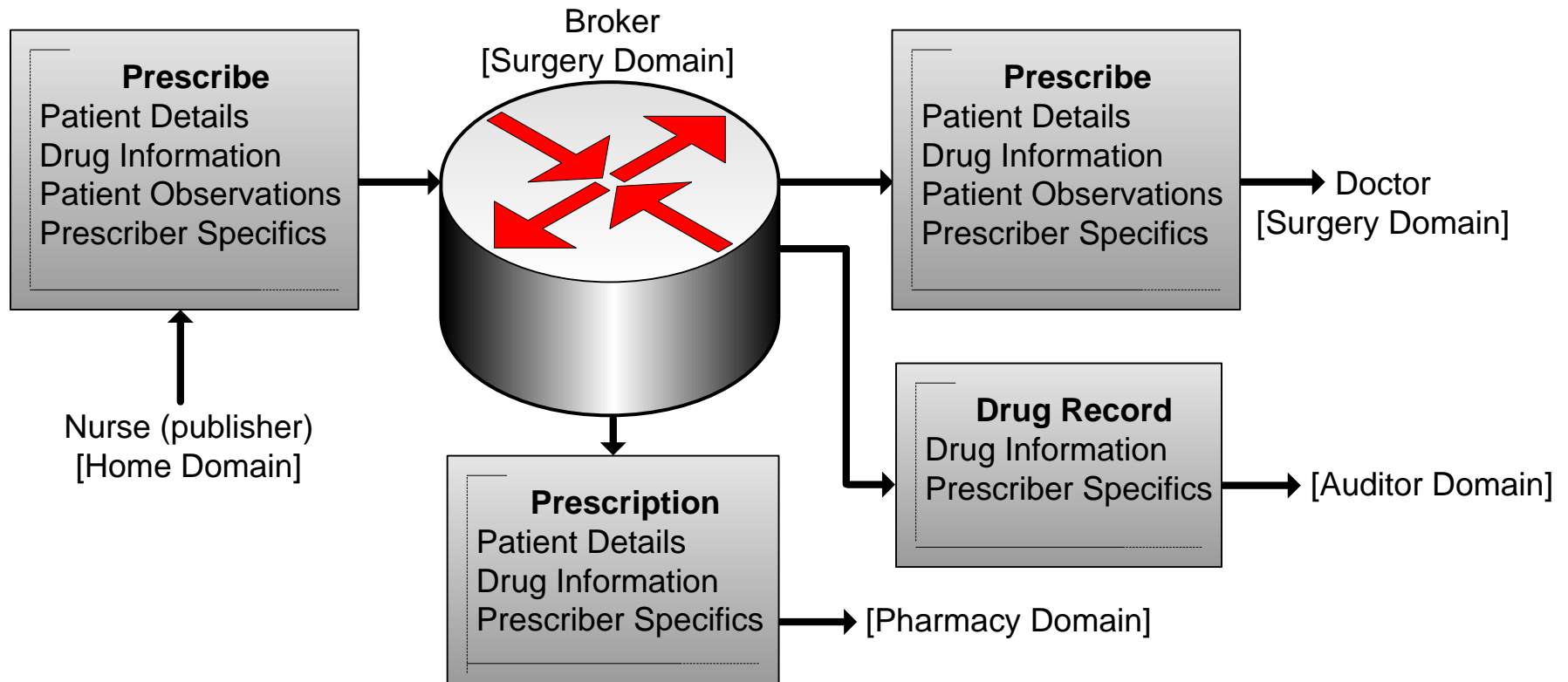
Restricted drugs must be audited



Enterprise Flows: Drug Control



Enterprise Flows: Drug Control



Enterprise Flows: Drug Control

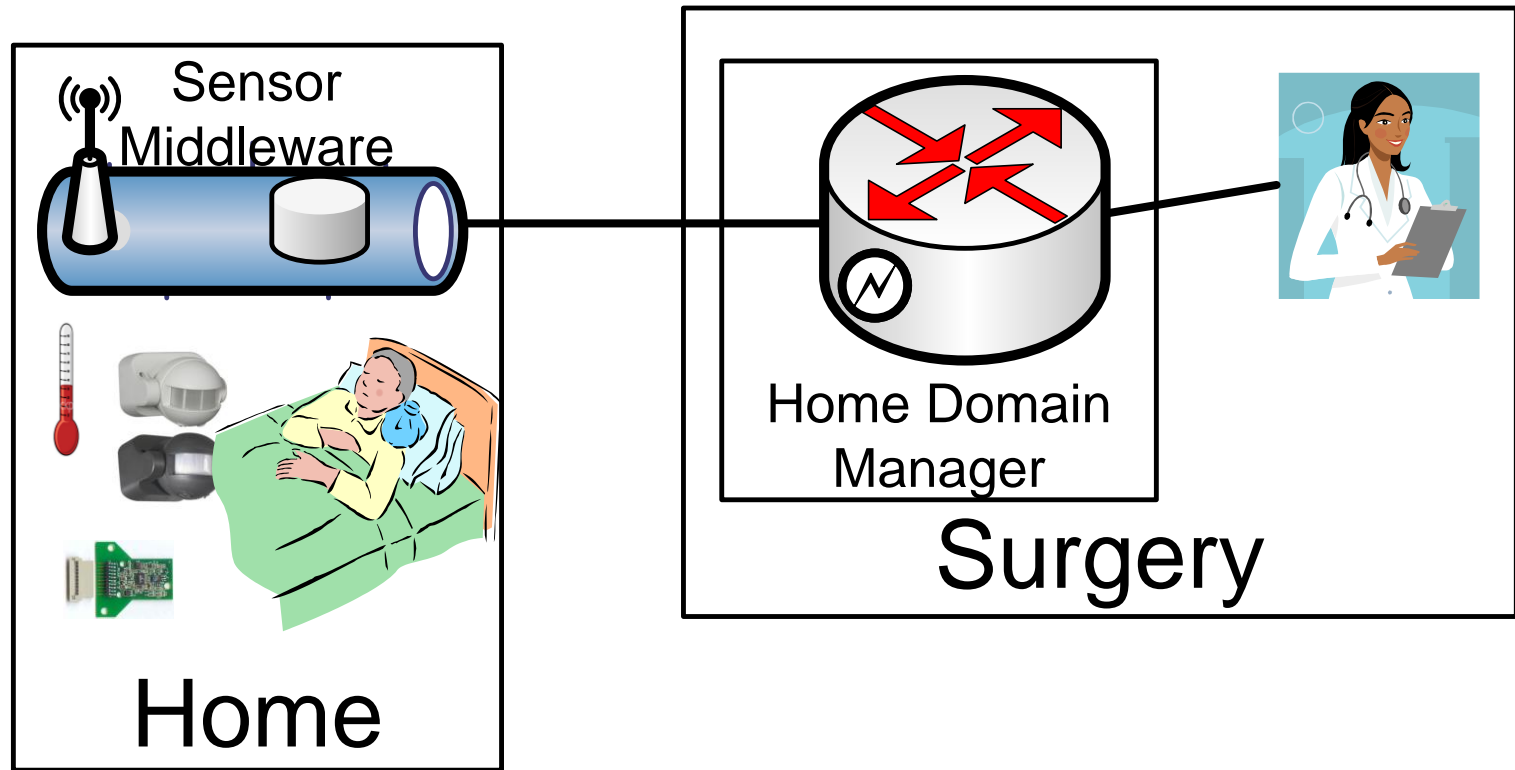
One incident relevant to many

- Depends on their *role in the care process*

Methodology allows a single event to be customised to recipients

- In line with notions of responsibility

Data Detail: Sensor Reading Fuzzification



Transform function: Replace GPS/Room coordinates with fuzzified equivalents (home or not home)

Condition: Not Emergency(patient_id)

Some Issues...

Conflict resolution

- Functions to detect possible conflicts
- Allow administrators to define resolution strategies
- Smarts?
 - Automation can be dangerous

Distribution

- Trusted brokers
 - Local vs centralised policies...
- Combination of encryption, content based routing

Audit management

- Single event produces multiple outputs
- Context change – what is relevant?

Related Work

Post-Matching Policy Model (PADRES)

- General policy enforcement model
- Execute an action after a matching operation

Scoping (Fiege)

- Grouping structures for visibility control
 - Overlay networks
 - Routing, interoperability, security

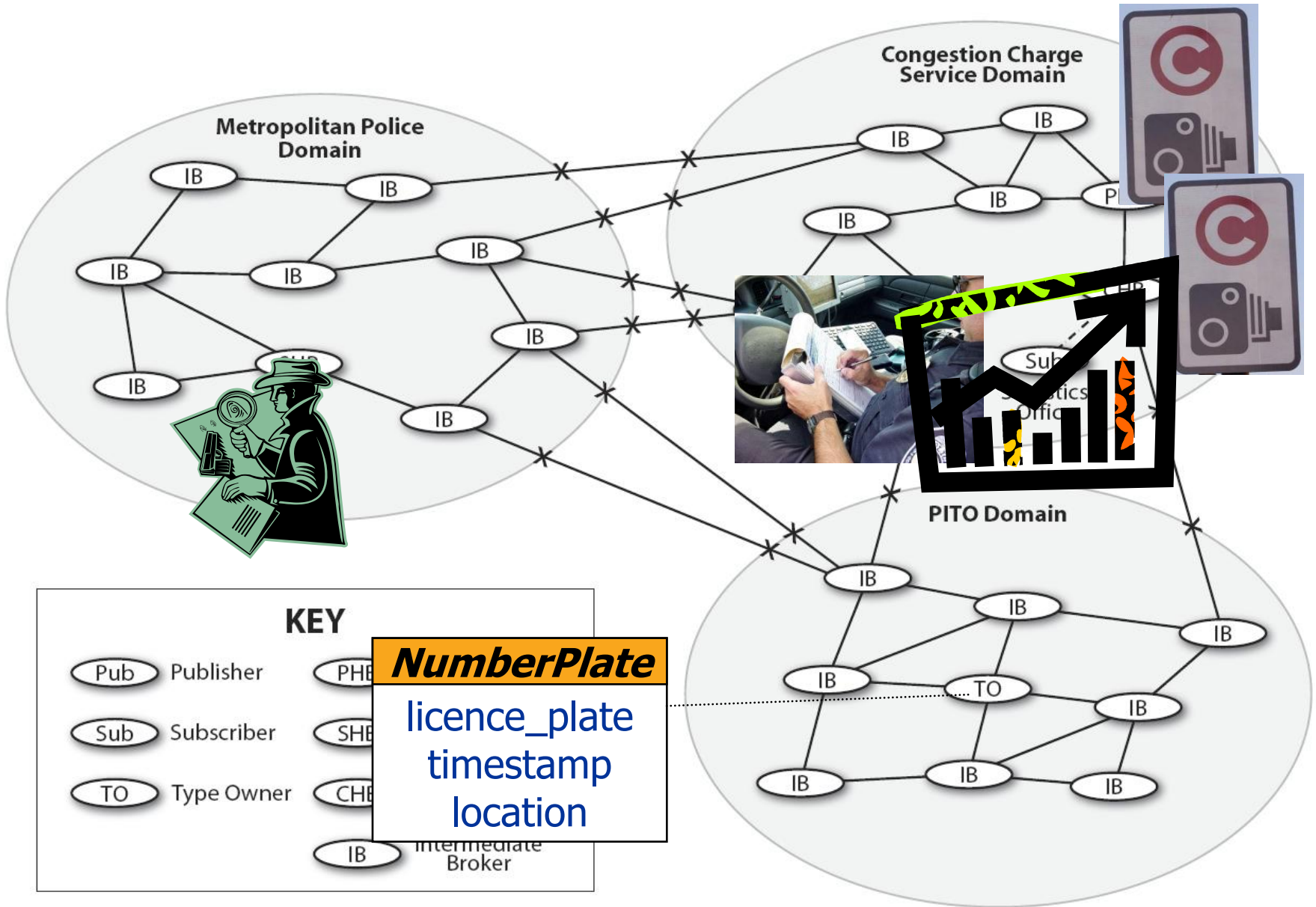
Symmetric Publish/Subscribe (CMU)

- Publishers specify filters to which subscriber filters must match

Location Privacy (CPOL)

- Event owners (instances) licence publications to others
- Research in this area is complementary...*

Congestion Charge Scenario



Interaction Control Policies

Control data released by a broker

Context-sensitive policy rules

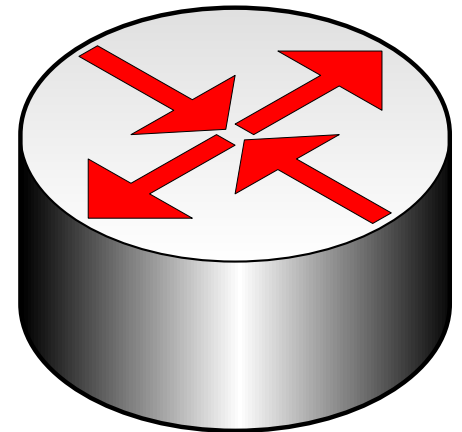
- Messaging information (event instance, principal)
- Credentials of the principals (e.g. Dr, Surgeon, Nurse)
- Environmental state
 - Stored data, external functions

Enforced in local brokers

- Reactive to events
 - At stages of the messaging process
- Produce and consume events

Database Publish/Subscribe

- Brings rich representation of state & data handling capabilities to the messaging system



Database Publish/Subscribe

Enterprise applications concern data

Build publish/subscribe substrate into DB

- Common type interface
- Replication
- Transactional semantics
- Relational model, query languages
- Performance, maintenance, etc...

Broker = Pub/Sub DB instance

- Routes messages
- Stores/consumes data
- Act as a publisher and/or subscriber
 - Produce & consume events

Rich representations of context

