Technical Report

UCAM-CL-TR-770
ISSN 1476-2986

Number 770

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Controlling the dissemination and disclosure of healthcare events

Jatinder Singh

February 2010

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

© 2010 Jatinder Singh

This technical report is based on a dissertation submitted September 2009 by the author for the degree of Doctor of Philosophy to the University of Cambridge, St. John's College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Abstract

Information is central to healthcare: for proper care, information must be shared. Modern healthcare is highly collaborative, involving interactions between users from a range of institutions, including primary and secondary care providers, researchers, government and private organisations. Each has specific data requirements relating to the service they provide, and must be informed of relevant information as it occurs.

Personal health information is highly sensitive. Those who collect/hold data as part of the care process are *responsible* for protecting its confidentiality, in line with patient consent, codes of practice and legislation. Ideally, one should receive only that information necessary for the tasks they perform—on a *need-to-know* basis.

Healthcare requires mechanisms to strictly control information dissemination.Many solutions fail to account for the scale and heterogeneity of the environment. Centrally managed data services impede the local autonomy of health institutions, impacting security by diminishing accountability and increasing the risks/impacts of incorrect disclosures. Direct, synchronous (request-response) communication requires an enumeration of every potential information source/sink. This is impractical when considering health services at a national level. Healthcare presents a data-driven environment highly amenable to an event-based infrastructure, which can inform, update and alert relevant parties of incidents as they occur. Event-based data dissemination paradigms, while efficient and scalable, generally lack the rigorous access control mechanisms required for health infrastructure.

This dissertation describes how *publish/subscribe*, an asynchronous, push-based, many-to-many middleware communication paradigm, is extended to include mechanisms for actively controlling information disclosure. We present *Interaction Control*: a data-control layer above a publish/subscribe service allowing the definition of context-aware policy rules to authorise information channels, transform information and restrict data propagation according to the circumstances. As dissemination policy is defined at the broker-level and enforced by the middleware, client compliance is ensured. Although policy enforcement involves extra processing, we show that in some cases the control mechanisms can actually improve performance over a general publish/subscribe implementation. We build Interaction Control mechanisms into integrated database-brokers to provide a rich representation of state; while facilitating audit, which is essential for accountability.

Healthcare requires the sharing of sensitive information across federated domains of administrative control. Interaction Control provides the means for balancing the competing concerns of information sharing and protection. It enables those responsible for information to meet their data management obligations, through specification of fine-grained disclosure policy.

To Amar, Charlene, Richelle and Jessica

Acknowledgements

I thank my supervisor, Prof. Jean Bacon for her support, guidance and generally making this work possible, and also Dr. Ken Moody for his useful criticisms and comments.

I also thank past and present members of the Opera group for their discussions and friendship, particularly Dave Eyers, who took almost a supervisory role, Luis Vargas, whose work influenced mine, and Pedro Brandão for his assistance with the sensor aspect. I'd also like to thank my friends for putting up with me and my bad jokes, especially during those stressful times.

And most of all, thanks to my family for their love and invaluable support.
This dissertation is dedicated to them.

Contents

1 Introduction **13**

1.1 Event-Based Middleware . . . 14

1.1.1 Publish/Subscribe . . . 14

1.1.2 Health Communication Infrastructure . . . 14

1.1.3 Research Statement . . . 15

1.1.4 Dissertation Outline . . . 17

2 Healthcare Primer **18**

2.1 Healthcare Organisation and Directions . . . 18

2.1.1 Health Information . . . 19

2.1.2 Healthcare Directions . . . 20

2.1.3 Technology-Assisted Homecare . . . 20

2.2 Information Sharing . . . 21

2.3 Information Sensitivity . . . 22

2.3.1 Electronic Data Concerns . . . 23

2.3.2 Responsibility . . . 23

2.4 Data-Driven Healthcare . . . 25

2.4.1 Data Relevance . . . 26

2.4.2 Contextual Control . . . 26

2.5 National Program for IT (NPfIT) . . . 26

2.6 Summary . . . 28

3 Systems Background **29**

3.1 Event-Based Systems . . . 29

3.1.1 Events and Healthcare . . . 30

3.2 Messaging Middleware . . . 30

3.2.1 Message Passing . . . 30

3.2.2 Message Queues . . . 31

3.2.3 Publish/Subscribe . . . 31

3.2.4 Distributed Publish/Subscribe . . . 33

3.2.5 Request-Response Interaction . . . 35
3.2.6 Healthcare Communication . . . 35
3.3 Access Control . . . 36
3.3.1 Role-Based Access Control . . . 36
3.3.2 RBAC and Beyond . . . 37
3.3.3 Healthcare Access Control . . . 38
3.4 Publish/Subscribe Security . . . 39
3.4.1 Policy-Based Control . . . 39
3.4.2 Encryption-Based Control . . . 41
3.4.3 Secure Event Types . . . 42
3.4.4 Scoping . . . 42
3.4.5 Applicability to Healthcare . . . 43
3.5 Summary . . . 45

4 Publish/Subscribe and Databases **46**
4.1 Databases . . . 46
4.1.1 Active Databases . . . 46
4.2 Database Publish/Subscribe . . . 47
4.2.1 Continuous Queries . . . 48
4.2.2 Database-Publish/Subscribe and Healthcare . . . 48
4.3 PostgreSQL-PS . . . 49
4.3.1 Event Model . . . 49
4.3.2 Advertisements . . . 52
4.3.3 Event Delivery . . . 52
4.3.4 Access Control . . . 55
4.4 Summary . . . 55

5 Interaction Control **56**
5.1 Motivation . . . 56
5.1.1 Middleware Control . . . 57
5.2 Assumptions . . . 58
5.2.1 Trust . . . 58
5.2.2 Other Assumptions . . . 58
5.3 Broker Context . . . 59
5.3.1 Permission Attributes . . . 59
5.4 Interaction Control . . . 60
5.4.1 Request Authorisations . . . 60
5.4.2 Imposed Conditions . . . 61

5.4.3 Transformations . . . 62
5.5 Policy Enforcement . . . 63
5.5.1 Request Validation . . . 63
5.5.2 Publication Enforcement . . . 65
5.6 IC in Distributed Broker Networks . . . 66
5.6.1 Link Authorisation Rules . . . 66
5.6.2 Request Forwarding and Processing . . . 67
5.6.3 Request Forwarding . . . 67
5.7 Relationships Between Rules . . . 70
5.7.1 Rule Fragmentation . . . 70
5.7.2 Application of Multiple Rules . . . 72
5.8 Related Work . . . 73
5.9 Summary . . . 75

6 Context and Conflict **76**
6.1 Representing State . . . 76
6.1.1 Fluents . . . 77
6.1.2 Fluent Discussion . . . 79
6.2 Credentials . . . 80
6.2.1 Credentials in Healthcare . . . 81
6.3 Context Summary . . . 81
6.4 Policy Conflict . . . 82
6.4.1 Static Conflict Detection . . . 82
6.4.2 Selection Summary . . . 84
6.4.3 Resolution Strategies . . . 85
6.4.4 Runtime Application . . . 87
6.4.5 Detection and Resolution . . . 88
6.4.6 Resolution Usage . . . 88
6.5 Summary . . . 88

7 Integration into PostgreSQL-PS **90**
7.1 Data Control Layer . . . 90
7.2 Client Operations . . . 91
7.2.1 Client-Specified Properties . . . 91
7.3 Context . . . 92
7.3.1 Permission Attributes . . . 93
7.3.2 Representation: Fluents . . . 93
7.3.3 Monitored Conditions . . . 93

7.4 Policy Definitions . . . 95
7.4.1 Representation . . . 95
7.4.2 Policy Storage . . . 95
7.4.3 Conflict Definition . . . 96
7.5 Layer Interactions: Hook Rules . . . 96
7.6 Request Processing . . . 98
7.6.1 Deserialisation . . . 98
7.6.2 Validation . . . 98
7.7 Request Propagation . . . 100
7.7.1 Routing Table Extensions . . . 100
7.7.2 Link Establishment . . . 101
7.7.3 Loading Forwarding Restrictions . . . 101
7.7.4 Request Forwarding . . . 102
7.7.5 Re-Evaluation . . . 104
7.8 Event Processing . . . 105
7.8.1 Transactional Event Processing . . . 105
7.8.2 Publication Processing . . . 107
7.8.3 Notification Processing . . . 108
7.9 Summary . . . 109

8 Healthcare Integration Summary 110
8.1 Healthcare Collaboration . . . 110
8.2 Data Sensitivity . . . 111
8.3 Broker Interactions . . . 112
8.4 Domains and Responsibility . . . 113
8.4.1 Trust . . . 113
8.4.2 Realising Domain Policy . . . 114
8.5 Federation . . . 114
8.5.1 Point-to-Point Control . . . 114
8.5.2 Central Services . . . 116
8.5.3 Shared Policy . . . 117
8.6 Summary . . . 117

9 Case Studies 118
9.1 Prescribing Scenario . . . 118
9.1.1 Adaptation to Homecare . . . 119
9.1.2 Data Flows and Policy . . . 119
9.1.3 Data Control Policies . . . 120

9.1.4 Single Broker Implementation . . . 121
9.1.5 Controlled Drug Implementation . . . 123
9.1.6 Comparison to Vanilla Pub/Sub . . . 123
9.1.7 Distributed Environment . . . 126
9.1.8 Scenario Discussion . . . 128
9.2 Regulation Change . . . 129
9.3 Sensor-Based Remote Care . . . 130
9.3.1 Sensor Middleware . . . 130
9.3.2 Data Control Rules . . . 130
9.3.3 Experimentation . . . 132
9.3.4 Scenario Discussion . . . 134
9.4 Contextual Complexity . . . 135
9.5 Policy Interface . . . 136
9.6 Summary . . . 137

10 Audit and Event Replay **139**
10.1 Recording Information . . . 140
10.1.1 Message Receipt/Delivery . . . 141
10.1.2 Event Auditing . . . 141
10.1.3 Exception Queues . . . 142
10.1.4 Conflict Resolution . . . 143
10.1.5 Request Auditing . . . 143
10.1.6 Policies . . . 145
10.2 Contextual History . . . 145
10.3 Accessing the Log . . . 146
10.4 Active Audit . . . 147
10.5 Event Replay . . . 148
10.5.1 Replay Requests . . . 148
10.5.2 Replay Request Validation . . . 150
10.5.3 Replay Request Processing . . . 150
10.5.4 Automatic Replay . . . 151
10.5.5 Application Examples . . . 152
10.5.6 Further Work . . . 154
10.6 Summary . . . 154

11 Conclusions and Further Work **155**
11.1 Further Work . . . 156
11.2 Concluding Remarks . . . 159

Bibliography **161**

Appendices **178**

A Transformations and Interaction Points **179**

B Fluent Representation **181**

C Example XML Rule Definitions **183**

C.1 Link Authorisation Rules 183
C.2 Request Authorisation Rules 183
C.3 Imposed Conditions 184
C.4 Transformations 184
C.5 Conflict Resolution 185

D Hook Rule Specifics **187**

D.1 Event Transformations 187
D.1.1 Resolving Event Transformations 188
D.2 Request Validators 188
D.3 Connection of Links 188
D.4 Request Transformations 189
D.5 Advertisement Processor 189
D.6 Bootstrapping 189

E Prescribing Policy Rules **191**

E.1 Event Authorisation Rules 191
E.2 Imposed Condition for the Auditor 192
E.3 The `prescribe–prescription` Transformation 192
E.4 The `prescribe–audit` Transformation 193

1 Introduction

Healthcare is a fundamental form of welfare. Societies are judged by the standard and accessibility of healthcare services [WHO00], as this directly affects quality of life.

Directions in healthcare are towards a *preventative* care model, to avoid the occurrence and/or severity of *acute* episodes, incidents requiring immediate attention. As the population ages, this push is to improve the quality of care and life, while reducing the burden on health resources. Technological advancement facilitates preventative care. Sensors can monitor physiological state, providing information for the detection, and perhaps prediction, of situations of concern. Communication technologies allow the transmission of alerts and observations. This enables the ongoing, remote administration of care services.

Information underpins modern health processes. Proper diagnosis and treatment regimes depend on carers having access to patient information. Such information concerns ongoing conditions and diseases, previous observations, test results, prior medications, allergies, family history, and so forth. Non-clinical processes also require information to support and manage care services, e.g. for billing/payment, audit, executive planning and research.

Healthcare is a collaborative environment. Many different providers are involved in a national-level care service, including surgeries, hospitals, specialists, governmental bodies (e.g. billing services), pharmacies, insurance companies, research institutions and auditors. Each operate with a degree of autonomy, providing specific services as part of the care process. Interactions occur through the *sharing* of information, which is necessary for the delivery of health services.

Enshrined in the Hippocratic Oath is the notion of confidentiality concerning personal health information. Those who access health data as part of the care process have not only a moral, but also a legal **responsibility** to respect its confidentiality. Modern healthcare involves a number of organisations that deliver specific care services, provisioned through members of staff. While individuals are responsible for information, so too are health organisations, as they provide the infrastructure to support the services on offer. It follows that health organisations require policies to meet their data management obligations [§2.3.2].

Thus, there is tension between sharing and protecting health information. This is managed by sharing sensitive information on a *need-to-know* basis [NHS02]: sharing only that information required in the circumstances. Such a standard is typically met by obtaining informed patient consent, though there are exceptions [DoH03a]. The infrastructure supporting health services must enable those responsible for information to control its transmission and disclosure.

1.1 Event-Based Middleware

Healthcare is a data-driven environment. Health processes react and respond to *incidents*, be they observations, diagnoses, results or treatments. A health record essentially forms a collection of incidents. As incidents reflect information, they must be communicated to the relevant care (and related) service providers, and thus are persisted in various data stores.

Such an environment is amenable to an *event-based middleware*. An *event* encapsulates information surrounding an incident. The role of an event-based middleware is to serve as a layer of indirection between applications and the network, to manage the distribution of events between *clients*: from *producers*, applications that generate information, to *consumers*, applications that receive and process such information. In contrast to the common request-response approach to communication, event-based middleware concerns the distribution of information as it occurs within the environment.

Event-based dissemination infrastructure is becoming increasingly relevant to healthcare, to inform of health incidents as they happen, e.g. alerting of a potential emergency situation, while ensuring that service providers operate with the latest information.

1.1.1 Publish/Subscribe

Publish/Subscribe (pub/sub) [EFGK03] is a scalable, asynchronous, many-to-many communication paradigm useful for event distribution. Pub/sub takes an information-centric approach to event delivery, where consumers (*subscribers*) specify their interest in receiving particular information, and producers (*publishers*) publish information independently of consumers. Clients communicate through *brokers*, which interconnect to provide the pub/sub service. Brokers co-operate to deliver events to the subscribers with matching interests.

In healthcare, a particular incident can be relevant to a number of providers at various locations. Pub/sub has been shown to be an efficient method for data distribution suitable for such environments [CRW01]. The paradigm provides many-to-many communication, decoupling producers from consumers, where routing concerns are left to the pub/sub service. This allows clients to interact without knowledge of every potential information source/sink. Subscriptions enable a client to specify an interest in receiving particular information, which may flow from a number of sources.

1.1.2 Health Communication Infrastructure

Healthcare is a heterogeneous environment. Health providers act with a degree of autonomy, not only with respect to the services they provide, but also regarding their information concerns. Local control is desirable: with control comes responsibility, and with

responsibility comes accountability. Rather than a complete, centralised definition of disclosure policy, it is preferred that providers maintain policy suitable for the requirements of the local environment. Similarly, decentralised data stores are preferable from a confidentiality perspective, where providers hold information relevant to their service, sharing it when appropriate. The technical infrastructure should account for the federated nature of health services.

Middleware provides a suitable place for implementing information disclosure controls, as it abstracts and isolates data control concerns away from client applications. This is important in a highly-collaborative environment such as healthcare, as it ensures client compliance with disclosure policy, and simplifies policy management through fewer definition and enforcement points. While pub/sub is suitable for wide-scale information distribution, its loosely-coupled nature complicates security [MFP06]. Pub/sub takes an information-centric approach to routing, raising issues of confidentiality concerning subscribers, and the brokers providing the pub/sub infrastructure. Data must be protected while in transit and on delivery. Health information remains sensitive, even after the death of patient [DoH03a]. It is therefore inappropriate to allow all interested parties access to information, and to liberally distribute information, even if encrypted, as part of the routing process.

Healthcare information is subject to strict governance procedures [DoH07a]. Given that information is best disclosed on a need-to-know basis, dissemination controls must be fine-grained. Privileges will depend on circumstances, e.g. restrictions on accessing patient data might be relaxed in an emergency. Thus, information disclosure infrastructure must allow for *context-aware* data controls. To monitor information flows, dissemination infrastructure must have the ability to record the details of the data transmitted and received [HIA06], including the specifics of those accessing patient information [NHS07b]. Audit facilitates accountability. Aside from audit, health incidents typically require persistence, e.g. in patient records, where providers maintain various information stores to manage relevant health information.

1.1.3 Research Statement

This dissertation considers the use of pub/sub as an infrastructure for supporting health services. We explore the issues concerning healthcare information and the associated implications for pub/sub, proposing mechanisms to enable the control of sensitive data in a distributed dissemination infrastructure.

Health services are information driven. We describe the (increasingly) collaborative nature of healthcare, arguing that a pub/sub event-based middleware provides an appropriate infrastructure for supporting care processes. As health information is perpetually sensitive, its disclosure must be controlled. This presents a paradox, in that information must be shared, yet protected. Communication infrastructure supporting healthcare must allow those responsible for information to meet their data management obligations. The ideal is to disclose information on a need-to-know basis.

Generally, pub/sub security models impose controls (only) against clients, assume a single administrative domain for the entire network, and/or distribute encrypted events, managing access through key distribution.[1] The core contribution of this work is the design

[1] In addition to complexity, this hinders accountability.

and development of *Interaction Control* (IC), a data control model built as a layer above a pub/sub middleware. IC is designed in accordance with the stringent requirements of healthcare, where all information flows must be strictly controlled by those responsible. As such, IC differs from other pub/sub security models by featuring:

Unified Control Mechanisms IC provides highly expressive policy rules to enable control not only over the transfer of data, but also the management of routing specifics. Authorisation rules control the establishment of (typed) information-channels. The information propagating through a channel can be filtered by way of imposed condition rules, and altered via transformation rules. The construction of the broker network is managed by rules that authorise broker interconnections. Routing paths throughout the distributed broker network are controlled by rules that filter and transform advertisements and subscriptions as they propagate to other brokers. Through transformations, IC allows more than binary (permit/deny) access control, as information can be perturbed, degraded or even enhanced according to the circumstances. Controls are unified as policy provides the mechanism to govern *all* transmissions from a pub/sub broker.

Point-to-Point Enforcement Healthcare is an environment of federated control, where providers are responsible for the information they manage. In IC, a broker enforces its policies directly against all connections, *including* other brokers. That is, rules concern the point-to-point (next-hop) transmission of information in a pub/sub network. By enabling local (broker-level) control over the information released, those responsible for information can meet their obligations[2] by tailoring information flows as appropriate to the (next) recipient in the circumstances.

Context-Awareness Access to health information depends on context. Generally pub/sub evaluates conditions (filters) referencing only the event instance. IC rules are *context-sensitive*, able to reference the credentials of the principal, external services and other aspects of environmental state, in addition to the content of the event/advertisement/subscription itself. This brings flexibility, allowing fine-grained definition of the circumstances for transmission. Further, the model is *context-aware*, in the sense that a change in circumstances might alter privilege, such as triggering the closure of an information channel.

Given that rules are context sensitive, it is possible for rules to conflict. As such, we describe a process that analyses IC rules to warn policy authors of potential conflicts, for which they can design around or define a run-time resolution strategy.

IC is built into an integrated database and pub/sub system. This gives rules access to a rich representation of state. To manage context, we present a method for representing state that clarifies the relationship between events and context, and that can interoperate with remote services, such as credential authorities, while preserving local control.

Health data is typically persisted and its transmission/disclosure audited. This is simplified in IC, as both storage and distribution functionality are managed under a common interface. Given persistence, there may be situations where historical information, perhaps previously inaccessible, may become relevant. There is some work that considers

[2]This assumes that those responsible for information manage their own local (broker) infrastructure.

issues of historical events in pub/sub; however, these overlook issues of control. As such, we extend IC to account for access to historical events, giving consideration as to the effects of a dynamic environment.

In summary, healthcare requires the sharing of sensitive information in a federated administrative environment. The premise of this work is that a broker should transmit only that information necessary in the particular circumstances. The goal of IC is to allow those responsible for information to meet their data management responsibilities, through the specification of broker-specific policy. Although the model is designed to provide controls to meet the stringent data management requirements of healthcare, it is generally applicable to environments where those dealing with information require fine-grained control over its dissemination.

1.1.4 Dissertation Outline

The remainder of this dissertation is organised as follows:

Chapter 2 presents an overview of the issues concerning the distribution of health information. This provides background as to the placement of IC within the health domain.

Chapter 3 describes related work from the systems perspective, introducing messaging paradigms, pub/sub and associated security efforts.

Chapter 4 provides an overview of PostgreSQL-PS, a coupled database-pub/sub framework, on which we build dissemination control mechanisms.

Chapter 5 introduces IC, our model for controlling pub/sub information flows.

Chapter 6 discusses issues of context and policy conflicts.

Chapter 7 details the specifics of integrating IC into PostgreSQL-PS.

Chapter 8 revisits the information concerns of healthcare, with reference to our data control model.

Chapter 9 applies IC to case studies derived from real-world healthcare requirements.

Chapter 10 describes auditing processes and presents a pub/sub extension that enables access to historical events.

Chapter 11 concludes this dissertation and describes areas for future research.

2

Healthcare Primer

National health services concern the welfare and well-being of the population. Collaboration is central to providing the most appropriate care. Information drives health processes; however, health data is inherently sensitive. Those providing care services must respect the confidentiality of health information. This chapter presents a general overview of healthcare, its structure in the UK, and issues concerning health communication infrastructure.

2.1 Healthcare Organisation and Directions

Healthcare is fundamental to any modern society [WHO00]. In England, the *National Health Service* (NHS) [NHSa] is in charge of providing health services for the country. It is a public health service, overseen by the executive. To give an indication as to the scale of the service, it is the largest employer in the world, with expenditure exceeding £170 million per day [Bre05].

The structure of the NHS is organised around *trusts* [NHS09a]. *Primary care trusts* (PCTs) manage the health services for a defined region. Hospitals and associated special care services are managed by *acute* (hospital) trusts. There also exist *care trusts*, which provide health and social services, *ambulance trusts*, and *mental health trusts*. The number of trusts serving England is presented in Fig. 2.1.

Trust	Number
Acute Trusts	175
Ambulance Trusts	12
Care Trusts	"few"
Foundation Trusts	115
Mental Health Trusts	60
Primary Care Trusts	152

Figure 2.1: A count of the NHS Trusts in England (per [NHS09a]).

A *Strategic Health Authority* (SHA) [SHA] provides the link between the Department of Health (government) and the NHS to oversee the local implementation of health policy. There are 10 SHAs in the UK, which manage the strategic/commissioning aspects of the trusts residing within their regional boundary. Fig. 2.2 presents the SHAs and the PCTs they manage.

The structure of the NHS involves several levels of interaction, where a number of local institutions, operating with a degree of autonomy, are overseen by other administrative bodies. In moving forward, an explicit goal of the NHS is to give local providers more freedom to manage their services [Dar08]. An example is the recently introduced *foundation* trust [DoH05a], which is a hospital with greater freedoms and flexibility of management, allowing local decision-making without the involvement of central government. All trusts, however, are still subject to monitoring to ensure compliance with care standards.

Figure 2.2: Regional map of NHS services adapted from [DoH03b], where bold lines denote SHA boundaries and thin lines PCTs.

2.1.1 Health Information

Healthcare processes are information driven.[1] As stated by Brennan [Bre05], *information is the challenge for the National Health Service.*

[1]Some distinguish the terms *information* and *data*, where information is defined as knowledge derived from data. We use the terms synonymously, presenting a framework to control the disclosure of data, where data-control policy addresses issues of inference.

Care processes depend on information. Highly relevant are *health records*, which contain details of previous conditions, medications, treatments and operations, in addition to other information such as allergies and family history. Actions and health workflows *(pathways)* generate and rely on information: observations and results are recorded, medications prescribed, treatments ordered, tests undertaken and services funded.

Those involved in the care process require access to relevant information to perform their tasks. However, healthcare information flows can be complex. The structure of the NHS, for instance, highlights the propensity for information to flow between care institutions at various levels of the organisational structure.

2.1.2 Healthcare Directions

Health processes are in a constant state of change: treatments evolve and technologies develop. A goal of health services worldwide is to improve the quality of care and increase the standard of living, in a sustainable manner.

Current healthcare is organised around *acute* care, where care is reactive [SE05], catering for the urgent requirements of patients [WHO02]. However, *chronic* conditions, health problems that require ongoing management over a period of time, consume a large proportion of healthcare resources. The UK Department of Health states that chronic diseases and related conditions form about 80% of GP consultations, 60% of hospital bed days and two-thirds of emergency admissions [DoH04b]. The number of people suffering from chronic diseases is increasing, with such conditions particularly prevalent in the elderly [DoH04a].

As the population ages, there is a world-wide push to better manage health services. The way forward is through innovation, improved use of information and patient empowerment (self care)—to bring about *preventative* care [DoH05b]. Acute care is costly, both financially and in terms of patient well-being. Preventative care involves the provision of ongoing care, to limit the occurrence and severity of acute incidents. The aim is to improve people's quality of life and avoid the exacerbation of conditions (where possible), while improving the allocation of health resources.

Information management is seen as crucial to healthcare evolution. The goal is to improve access to information and to increase efficiency and the standard of care, for instance by reducing the occurrence of *adverse events* (e.g. accidental mistreatments), while providing the foundation to support future care services. Many countries, including the UK (§2.5,[2] USA [DoHHS], Australia [NEHTA], Canada [CHI] and France [GIP] are in the process of developing technical infrastructure to modernise their health services.

2.1.3 Technology-Assisted Homecare

Homecare,[3] or *assisted living*, involves providing care services for patients outside of traditional care institutions (e.g. a surgery or hospital). Technological developments, such as sensors that monitor patient and/or environmental state, enable the ongoing, remote provision and automation of care services.[4]

[2]The programmes vary for the different home countries.

[3]The term *homecare* is used as it is expected that the home environment will be the primary domain for patient management. However, care might encompass mobile technologies that monitor or provide feedback while the patient is outside their home.

[4]For examples see `http://www.pervasivehealth.org/`.

Given the push towards preventative care, homecare is an increasing area of interest. Sensor technologies are seen as integral to future care [EC07b], where sensor data can provide a detailed representation of the patient's situation, to alert of particular incidents (i.e. emergencies) [BSNM07] and reduce the need for human intervention. Detailed patient and lifestyle information is useful for treatment and early diagnosis, which can reduce the severity of the condition—improving resource allocation while reducing the burden on acute care resources [EC07a, HPS07, DoHD05]. The patient enjoys greater independence, requiring less institutional time (e.g. hospitals, surgeries for 'check-ups') [DoHD05], and has access to more information to assist in self-care (patient empowerment) [DoH05b].

2.2 Information Sharing

Information underpins the health service. Care processes both generate and rely on the distribution of data. Information sharing is central to healthcare, where consultation and collaboration is common, and often required for patient care [Cic90]. Collaboration in healthcare is well established. A modernised Hippocratic Oath [Las] states that carers must interact with others where required for treatment. The General Medical Council guidelines for medical practitioners explicitly recognise the need for practitioners to interact with others [GMCU06]. Clearly, parties involved in the care process require relevant information to perform their duties.

Healthcare services are specialised. *Entities*, such as a medical professional, organisation, employee or system, interact with others to provide care services. Care-specific entities include patients, GPs, nurses, surgeons, pathologists, specialists, hospitals and so forth. Information is stored in databases, accessible by a range of entities. Health data is also relevant to entities who do not (directly) provide treatment, but are essential to the care process, such as billing services, insurance firms and pharmaceutical companies. Other entities are involved in the *secondary use* of information, for purposes such as medical research [AoMS06], for (governmental) administration (e.g. usage statistics) and to support audit, compliance and clinical governance [Bre05]. Such information is shared in accordance with legislation and may or may not require consent and/or a degree of anonymisation [PIAG08a].

As care moves to a preventative model, a greater number of entities become involved in providing care services. Preventative care regimes often include specialists in addition to non-clinical services, such as dietitians, social services (counsellors), physiotherapists, etc. Homecare environments operate outside of a central administrative domain (e.g. a hospital, or surgery), and thus involve interactions between many entities, often across organisational boundaries. Homecare introduces technology into the interaction-mix, where information flows to/from sensors and monitoring devices, the associated software systems and maintenance staff.

As illustrated in Fig. 2.3, health information is relevant to a number of entities, grounded in various administrative domains, each of which provides a particular service as part of the care process. Healthcare infrastructure must facilitate information sharing between entities.

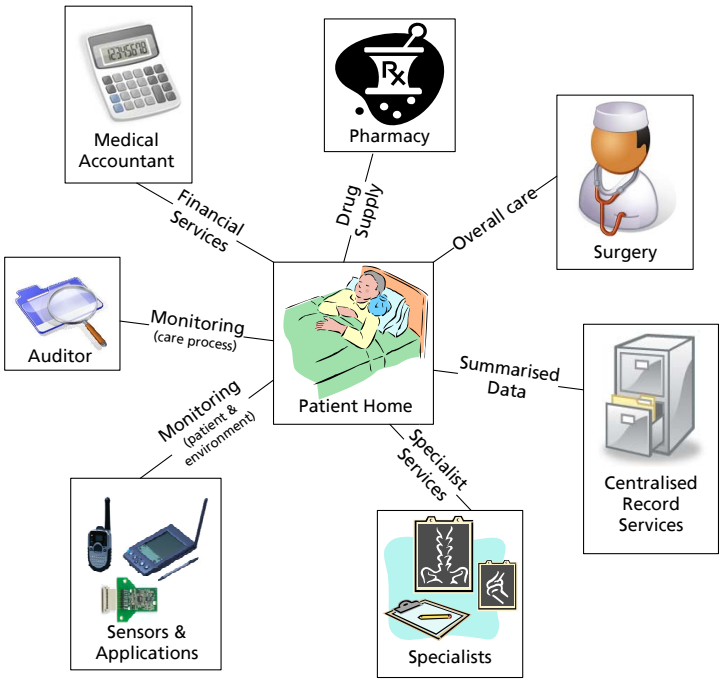


Figure 2.3: Home healthcare involves interactions between entities, managed in different administrative domains, each delivering specific services as part of the care process.

2.3 Information Sensitivity

Healthcare information is sensitive; its confidentiality must be protected. The concerns in managing health information are well established, reflected in the Hippocratic Oath [Las] and professional guidance documents [GMCU06], which state that medical professionals must act in the interests of patients. This includes consulting with other professionals to assist in care (sharing), and respecting the confidentiality of personal information obtained as part of the care process (protection).

Healthcare information brings some interesting considerations. Health data remains sensitive over time, where confidentiality must be maintained, even after death [NHS06c]. Further, while health information concerns a patient, it is typically used by healthcare professionals. That is, in healthcare users operate on the sensitive data of others [SBM07]. In many application domains, issues of privacy concern user anonymity. Healthcare is different in that although data must remain confidential, (generally) users must be accountable for their actions—the actions of medical professionals must be visible.[5]

Notions of confidentiality underpin the carer-patient relationship. Various codes of practices [BMA07a, GMCU06, DoH03a] and legislation (see [DoH03a] and [PIAG08b] for a summary) make it clear that those who use and hold personal information are legally and ethically responsible for maintaining its confidentiality. Generally, personally identifiable medical information may be shared if there is patient consent. There are exceptions, if it is in the public interest, or by reason of a court order or statutory provision, e.g. UK Cancer Registries receive data on cancer patients even without informed consent [UKACR09].

[5]Clearly access to audit information must be controlled (Ch. 10).

2.3.1 Electronic Data Concerns

The electronic representation of sensitive information raises issues of confidentiality. One concern is that electronic data is easily aggregated. Data aggregation poses a privacy risk [Sch04], as it creates a large repository of sensitive information accessible by many users [And96]. Query and data-mining techniques facilitate rapid and complex search, summary and analysis [CHY96] to an extent previously not possible with physical (paper) records. The mining of aggregated data has benefits, an example being for research, but there are issues concerning consent [PIAG08a, CM96]. At a higher-level, aggregation and centralisation works to diminish responsibility for data protection: "who is the 'controller' of data?" [ABD+09].

Confidentiality issues often arise out of user negligence, such as a misplaced storage device containing private information.[6] Such incidents typically involve volumes of information. Despite organisational policies dictating the safe usage of information, consequences are relevant not only for those at fault and those whose data is involved, but also to the organisation responsible for that information. In California, recent legislation mandating the disclosure of confidentiality failures has shown that breaches occur more often than expected [Edi09].

The ease of accessing (volumes of) electronic data, coupled with the difficulty in recovering from the fallout of a privacy failure [Sch03], highlights the importance of mechanisms for controlling access to electronic health data. Given the legislative requirements, and the expectations of the medical profession and the community to maintain data confidentiality, health systems/infrastructure must suitably protect medical information.

The NHS Care Record Guarantee [NHS07b] is a statement aimed at addressing public concerns surrounding health information. The guarantee specifies a *duty* on NHS services to ensure the security and confidentiality of patient records. This is realised through contextual (workgroup and role-based) access control regimes and audit trails.

2.3.2 Responsibility

An explicit goal of the NHS is to give local providers a greater degree of freedom to manage their services [Dar08, DoH09b]. Local control is not only a political ideal, but is something that health institutions demand [HMF04, eHe08a, eHe07a]. This gives a care organisation the ability to manage their own practices and procedures. There are strong arguments for a federated[7] healthcare infrastructure. With control comes **responsibility**.

Federation

National-level healthcare consists of a number of bodies of administrative control, managing their own practices and procedure. There is much support for federated healthcare environments, where organisations hold local data, sharing it only when necessary, in line with consent and legislation [HMF04, ABD+09, LKHT09]. Federation restricts the ability for information to be used for secondary (research, bureaucratic) purposes without 'proper' consent [ABD+09]; data usage is an issue of real concern to patients [RHH+04].

[6] See `http://nhsdatabase.info` for an archive of reported breaches.

[7] Some authors distinguish between *federation* and *decentralisation*, where the former refers to policy and the latter to data [HMF04]. As we integrate sharing policy into the storage infrastructure, we use *federation* to encompass both meanings.

Policy tailored to the local environment better maps to local practice/procedure, and thus promotes compliance. Further, an incorrect policy or malicious breach provides access only to local information, thus limiting the impact of loss [And96]. With local responsibility comes accountability: health domains are accountable, and thus subjected to auditing and monitoring procedures [DoH09c].

Medical professionals take their responsibilities seriously, many expressing concerns over the risk posed to patient confidentiality by centralised record systems [BMA07b, HMF04, ABD+09]. Current NHS data-storage projects involve much centralisation,[8] which are criticised as they increase risk of disclosure while diminishing local control and thus accountability [HSH+09]. Health providers are more comfortable with controlling the data for which they are responsible, as shown in a recent survey where 81% of respondents were against storing their local surgery data on shared NHS databases [eHe08b]. The preferred approach to centralisation is the use of smaller databases with controlled sharing [Dav07].

Sharing Protocols

An organisation providing care services is responsible for the information it holds [NHS07b]. Given that healthcare is highly collaborative, access to information must be restricted to the circumstances in which it is appropriate. To meet data management responsibilities, information is best shared according to the *need-to-know*[9] principle [NHS02, NHS07b], disclosing information only as required.

Accountability and transparency are vital to an organisation's handling and sharing of sensitive information [ABD+09]. A health institution should have clearly defined policies, termed an *information sharing protocol* [DoH03a], defining the circumstances in which information is shared. Such policy is developed as appropriate to the local environment, and should consider legal and ethical responsibilities (see [DoH07b] for an overview), business practices, clinical processes, service contracts with other organisations, NHS directives, etc. Such a protocol covers access by local entities, and those from remote bodies. The staff of an organisation are integral to the policy design process.

The sharing protocol assists a practitioner in explaining to the patient the information shared for the particular course of treatment [SRLH05, DoH03a]. In this way, a medical professional can obtain informed consent [NHS02] before performing actions disclosing patient information.

Patient privacy concerns focus on the reasons for sharing (the use of the information), the recipient, and the (perceived) sensitivity of the information. Patients will more readily share information when it is closely related to their treatment, less so as the relation becomes more indirect, such as for research [PIAG08b, WHEH06, CB95, RHH+04, NHS02]. Patients tend to trust their physicians to act as gatekeepers for their information [BRG00], which means that a prudent information sharing protocol will often meet the confidentiality expectations of patients. However, it must be possible to qualify any general protocol with patient-specific disclosure preferences.

[8] This may change with the government [Con09].

[9] We use the term *need-to-know* to mean as appropriate to satisfy data management obligations, accounting for consent, specific requests, codes of practice, legislation and context (e.g. patient condition). Our use is politically agnostic—for a description of concerns with the term see [FfIPR05].

Domains

This dissertation considers information sharing with reference to domains. We define a *domain* as a unit, governed by independent administrative policy, that provides particular services [BESP08]. Regarding national-level healthcare, a domain naturally maps to an organisation (a legal entity), or any body providing care related services that is responsible for its own business procedures, defines its own policies and manages its own infrastructure. Domains may be hierarchical (nested), in the sense one domain resides within another. We assume that a domain defines its local policy in accordance with policies imposed by higher-level domains, global directives and legislation. Example domains include a particular hospital, surgery, pathology laboratory or insurance company.

A domain grounds a number of entities, such as doctors, nurses, carers, accountants, managers, software systems, often through the use of employment contracts (if human).[10]

2.4 Data-Driven Healthcare

Modern healthcare is highly data-driven. Health information tends to consist of *incidents*, including: *actions performed*, such as a nurse administering a treatment or a patient taking a drug; *observations*, such as sensor monitoring reports, e.g. at timed intervals, or upon various readings; and *environmental changes*, such as a carer entering the household, or the detection of an emergency situation. Clinical information revolves around occurrences: health records contain information of observations, treatments administered, medicines prescribed and test results.

Given that healthcare is highly collaborative, information surrounding an incident must be communicated to the relevant parties. Thus, a health incident is often relevant to multiple entities. Actively notifying the entities of incidents is important to enable an appropriate response. From a communication standpoint, both the acute and preventative care models are reactive in that actions are taken in response to incidents,[11] regardless the severity. Incidents might trigger workflow procedures, or remedial actions—a particularly motivating example is an emergency situation. However, more generally, it is also important that information is transferred to ensure that health processes are undertaken with the latest representation of state, i.e. by propagating information to a particular data store. This is because relatively benign data may become significant when combined with other information.

Sensor and monitoring technologies enable a shift from organisational services; where care is administered only at specific intervals that directly occupy staff time, e.g. at an appointment, or as a nurse works the wards; to remote and/or ongoing care, where staff interact when required. In a homecare environment, aspects of patient state is monitored. Such information is relevant to staff from various organisations, to alert of anomalous situations (e.g. emergencies or technical exceptions), to inform of the actions performed by care staff, to update patient status, to provide information for audit, billing and secondary uses (e.g. research), and to provide patient feedback (e.g. reminders, positive reinforcement).

[10] This separation does not imply that domains and principals are autonomous: employees will help shape the policy of a domain, just as employees are bound to act in accordance with domain practices.

[11] This may be a current incident, or a history of incidents.

Technical infrastructure must provide the relevant entities with sufficient information to perform their duties, so that they may react/respond to events as they occur.

2.4.1 Data Relevance

The relevance of information to an entity depends on a number of factors such as their role in the care process, credentials, grounding health domain and managing organisation, in addition to patient particulars (conditions, demographics) and the current environmental state (current context, well-being). An incident (event) is often relevant to several entities from a number of domains. However, as shown in the example (Fig. 2.4), given that entities provide different services, only aspects of an incident are appropriate to each [SVB08]. Information disclosure policy must account for these differences in data requirements.

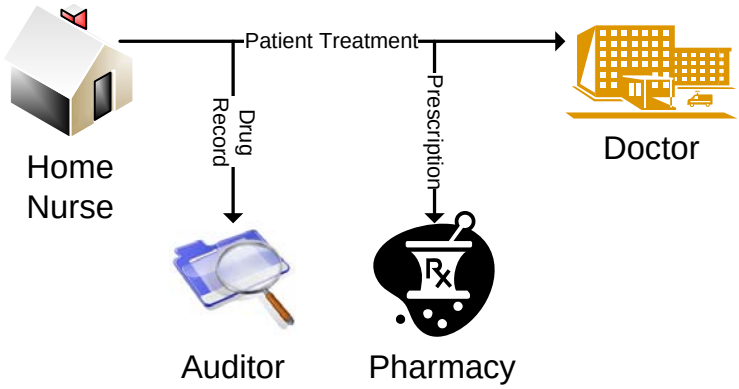


Figure 2.4: The act of prescribing has a different relevance to each care provider.

2.4.2 Contextual Control

As healthcare is data-driven, incidents affect state. This impacts on information disclosure, in that the appropriate level of data protection is often *context* dependent. For instance, access restrictions on patient data may be relaxed in an emergency situation, or a nurse may only access the information of patients in the ward in which she is working. The NHS proposals for a number of shared (large centralised) databases will require information to flow to the appropriate store [§2.5]. In practice, this is likely to be determined by the nature and circumstances of the incident itself.

Supporting infrastructure must account for the dynamic nature of healthcare in an environment of federated control. It must provide the means for managing the sharing of information to *appropriately* disclose information in the given context, accounting for identity, roles, credentials and aspects of environmental state.

2.5 National Program for IT (NPfIT)

England is in the process of implementing a *National Program for IT* (NPfIT) [NHSb], which aims to modernise the NHS. The goal is to evolve health processes, to improve care while providing a degree of visibility to the executive. The project involves a number of healthcare infrastructure contracts, consisting of the development of several hundred computer systems [Bre05]. Specific aims include the establishment of an electronic prescription

service, an online appointment booking service, a care record service, and the upgrade of communications and storage infrastructure [CU07, Bre05]. The NPfIT (sub)projects most relevant to this dissertation include:

Care Records Service (CRS) The NHS CRS is perhaps the most highly publicised aspect of the project, which describes the use of shared databases to store patient information. The *Summary Care Record* (SCR), available nationally, stores basic patient information, such as demographics, allergies, medications.[12] The proposed *Detailed Care Record* (DCR) is described to share clinical information from/between systems from local organisations to provide an integrated clinical record. A DCR is envisaged to contain only a subset of the information held by local systems. A *Secondary Uses Service* (SUS) is also defined, which aggregates data from a number of sources to provide visibility to the executive, and data for medical research and audit.[13] The proposal for shared (and centralised) databases requires that local information is transmitted to the relevant store.

Security mechanisms include the notion of a *sealed envelope* [NHS06c], in which particularly sensitive information can be placed inside an (electronic) envelope. The existence of a *sealed* envelope is visible to users with access to the patient record, though access to its contents is actively audited. A *locked* envelope is invisible except to the patient and the users belonging to the workgroup who created it.

N3 Network This is a secure network on which all NHS organisations must use for communication [NHS09d]. All network traffic is encrypted, and each user is uniquely identifiable, authenticated with a smartcard and PIN.

Electronic Staff Record (ESR) The ESR [McK] provides details regarding clients (system users), including their grounding domain.[14]

NHS RBAC Service This service provides a role-based privilege allocation system, where users are mapped to roles, and roles to particular activities [NHS08a, NHS06b]. Roles and privileges are centrally defined and allocated; an organisation may only augment a role with additional activities for the local environment [NHS08d].

Legitimate Relationship Service (LRS) The LRS [NHS06a] is a central service that defines associations between patients and staff—individually or via a workgroup. This is used to ensure that staff may only access information concerning patients they treat.

Criticisms of the NPfIT

The NPfIT has been the target of criticism from a number of fronts (see [CU07] and [NHS09b] for an overview). Some issues concern the cost and scale of the project [NAO08, eHe09]. The project is over budget, and has been subject to a number of delays. Requirements documents have been found to be under-specified, or rewritten with significant changes, and projects are subject to function-creep. The project is highly political; the Conservative Party, if elected, seek to re-negotiate NPfIT contracts and dismantle centralised infrastructure [Con09].

The project also raises issues of consent, concerning whether and how a patient can opt in/out of an electronic care record, and the resulting impact on their care [Bha09, NHS06d]. There are questions as to the lawfulness of the CRS, in particular the SUS [ABD+09].

[12] The SCR is generally purported to store basic medical information necessary in an emergency situation. However, there are concerns as to scope-creep [GSB+08, CU07, NHS06d].

[13] See [CU07] for details and discussion of the issues concerning the NHS patient record proposals.

[14] Note that the ESR technically sits outside of the NPfIT.

Much controversy surrounds the security aspect [NHS09b]. Closed specification documents, preventing peer-review, raise questions as to the quality of the security mechanisms. Concern was raised over the launch of the SCR before completion of the sealed-envelope functionality [eHe05]. Issues stem from the centralised nature of NPfIT services [PSC07]. Centralised data stores increase the risks of a confidentiality breach [§2.3.1], while centralised policy may fail to account for local concerns. This runs against the autonomy of a local provider [HSH+09]—inflexibility makes it harder for a provider to meet their data management obligations. Centralisation diminishes responsibility for a data breach, and may lead to (a sometimes necessary) circumvention of the control mechanism [eHe07b]. Arguments are for central standards, but local storage, control and responsibility [HSH+09].

2.6 Summary

There are a number of social and political issues surrounding healthcare information, to which technical infrastructure must adhere and support.

Health information must be shared, often with a number of domains providing care (and related) services. These domains operate autonomously, maintaining their own set of policies and business processes. Healthcare processes are data-driven, where incidents are relevant to numerous entities providing care services.

Somewhat paradoxically, health data must also be protected. There is responsibility imposed on those providing services to ensure the confidentiality of health information. The relevance of information to a domain/entity depends on its role in the care process. Data should be shared on a need-to-know basis. This is best realised through granular, context-sensitive controls. Responsibility brings accountability. Control mechanisms must enable adequate control over information flows, to allow a domain to meet their data management obligations.

3

Systems Background

This chapter presents background from the systems perspective, with reference to the requirements of healthcare systems, to frame the data control model presented in this dissertation. Specifically we introduce *event-based computing* [MFP06], which typically involves a decoupled interaction surrounding occurrences of interest. We describe the specifics of publish/subscribe, a paradigm used for event distribution, and summarise various approaches in securing the paradigm.

3.1 Event-Based Systems

Event-based systems concern the distribution of events, where an *event* is defined as "any happening of interest that can be observed within a computer system" [MFP06]. An event encapsulates information about an incident as it occurs within an environment, representing a change in state, such as a patient being discharged from a ward, or an observation, such as a sensor reading. Events underpin data-driven systems, where an incident may trigger a response.

Principals (or *clients*) are the software components, often user driven, operating within the system. A client may be an event *producer* and/or *consumer*. Producers generate events, while consumers process events of interest. Clients operate independently, in the sense they produce/consume events according to their own motivations and internal representation of state. *Complex event processing* (CEP) refers to the body of work concerning event consumption and processing [Luc02].

Associated with a particular semantic, an event may be of interest to a number of consumers. It is the responsibility of the *event notification service* to inform consumers of relevant events as they occur within the environment. In this way, producers and consumers are decoupled. Notification typically occurs through a *messaging middleware*, which encapsulates an event (and routing metadata) in a *message* for transmission between clients. We describe paradigms for middleware messaging in §3.2.

3.1.1 Events and Healthcare

Healthcare is information-centric: as described, data is the driver for healthcare processes. Health providers require information to perform their duties. Clinical information revolves around occurrences. Health records, observation charts, and the like all record details of incidents, such as symptoms, the administration of a treatment, or observations. Health information is inherently event-based, where an event may be relevant to a number of clients, based in various care and related organisations (§2.4.1).

Infrastructure for information sharing is crucial to supporting healthcare services. Monitoring technologies (sensors) can measure aspects of state, without requiring the physical presence of carers. Events representing an emergency situation are perhaps relevant to emergency (A&E) services, the patient's GP and family members. An event-dissemination model serves to notify clients of relevant events if and when they occur, so that they may respond appropriately. Emergency scenarios particularly motivate the need for event-based communication, as clients are notified of the incident as it is detected, as opposed to periodically polling for the current state. However, even in more benign situations, event dissemination helps ensure that health workflows—clinical or administrative—are undertaken with the most up-to-date representation of state. That is, events must be communicated to the relevant parties to enable the appropriate actions to be taken. These actions may be based on a current event or historical record.[1]

3.2 Messaging Middleware

Message Oriented Middleware (middleware) [BCSS99] provides a level of indirection between clients and the (network) infrastructure to enable communication irrespective of implementation specifics. In an event-based system, the middleware functions to deliver event instances (*notifications*) from producers to interested consumers. In this section, we provide an overview of *paradigms* for event dissemination.[2] We then (§3.2.5) describe *request-response*, an alternative interaction paradigm, the properties of which reinforce the suitability of an event-based pub/sub middleware to an environment such as healthcare (§3.2.6).

Note that we use the term *event* to refer to a message encapsulating an event instance.

3.2.1 Message Passing

A basic form of distributed communication is simple *message passing* [EFGK03], in which the producer directly sends an event to the consumer. The producer requires knowledge of the recipient address (and/or vice-versa), transmitting the event to the consumer as it occurs through a dedicated communication channel. Events can be sent asynchronously, where the producer resumes processing after transmission. The consumption of events is typically synchronous, in that the recipient (thread) blocks waiting for events to be sent by the producer through the active communication channel.

[1] Benign events in isolation may become significant when combined with other information. This issue motivates the work of complex event processing.

[2] This discussion concerns messaging paradigms as opposed to specific protocols, implementations or applications. Indeed, protocols can be built on top of messaging paradigms, and vice-versa.

3.2.2 Message Queues

Message queues work to mediate between producers and consumers. Producers send events to *event brokers*, which store events in queues. A broker enqueues events on receipt, which are dequeued by consumers. Typically, each event is consumed by only one recipient. As such, the model generally provides one-to-one or many-to-one communication: multiple producers to a single consumer.[3]

This approach differs from the message passing scenario by decoupling producers and consumers. Clients need only know the address of the broker hosting the queue through which communication occurs. As the queue provides storage, producers and consumers need not be simultaneously active.

There are various open-source and commercial implementations of message queues, such as IBM Websphere MQ [IBM09b], Apache ActiveMQ [Apa08], Microsoft MSMQ [Mic09a] and Oracle AQ [Ora09a], many of which implement the Java Messaging System standard (JMS) [SM02].

3.2.3 Publish/Subscribe

Publish/subscribe [EFGK03] (pub/sub) is an asynchronous, many-to-many communication paradigm suited for large-scale distributed systems. Consumers (subscribers) register their interest in receiving an event, or subset of events, through a *subscription*. A producer (*publisher*) produces (*publishes*) events (*publications*) independently of subscribers. Communication occurs through a pub/sub service, which might be centralised as a single event broker, similar to that of §3.2.2,[4] or decentralised as a network of event brokers [BEMP05]. In this model, a broker is concerned with routing event instances to(wards) interested consumers. The pub/sub service, operating through event brokers, is responsible for the asynchronous delivery of events (publications) to all subscribers with matching subscriptions. Fig. 3.1 presents the client-API for a general pub/sub service.

Operation	Issuer	Description
Advertise()	Publisher	Advertise the (possible) future publication of an event type/topic
Unadvertise()	Publisher	Remove an existing advertisement
Publish()	Publisher	Publish an event instance
Subscribe()	Subscriber	Create a subscription for a particular event type/topic
Unsubscribe()	Subscriber	Remove an existing subscription

Figure 3.1: Client API of a typical Publish/Subscribe service (with advertisements).

A key property of pub/sub is its loose-coupling, which brings about scalability [EFGK03, BV06]. Clients avoid the burden of managing the addresses of sources and sinks: publishers send events to a broker, while subscribers register their interest with and receive events from a broker. Typically publishers and subscribers are connected to different brokers. The pub/sub service is responsible for routing an event throughout the broker network, delivering a publication to the subscribers holding *matching* subscriptions.

Subscription Flavours

Pub/sub takes an information-centric approach to transmission, using subscriptions to perform routing operations. As such, pub/sub approaches vary on the expressiveness of

[3]Note that other communication patterns can be constructed [Var09].

[4]In contrast with message queues, a pub/sub broker (generally) operates without a queue.

their subscription models, which in turn affects the complexity of the matching operations. Pub/sub models can be broadly categorised as *topic*, *content* or *type* based.

Topic-Based Publish/Subscribe

In this model, a *topic* refers to a subject that groups a number of events. A subscriber subscribes to a topic to receive all events published under that topic. Routing occurs based on the topic (group) identifier. As this is analogous to group communication mechanisms; topic-based pub/sub is often implemented using *IP Multicast* [Dee89]. Some topic-based pub/sub implementations include TIBCO Rendezvous [TIB09] and Vitiria ESB [Vit09].

Topic-based subscriptions are limited in their degree of expressiveness, in that a subscriber receives *all* events published under their subscribed topic. Topics may be partitioned, perhaps into a hierarchy, to provide more granular subscriptions. In Fig. 3.2, a subscription to `Hospital` will match events of all subtopics, while a subscription to `Critical Care` will return only events from that category.

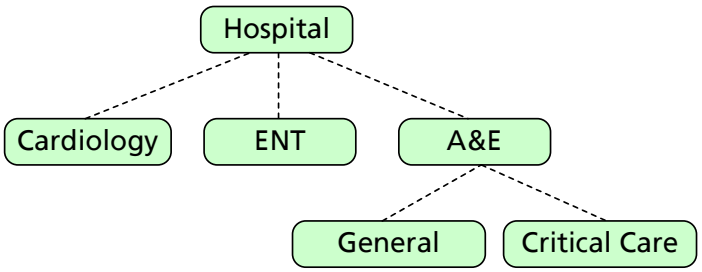


Figure 3.2: An event topic hierarchy.

Content-Based Publish/Subscribe

This approach enables subscribers to express their interest in receiving a set of events through specification of a subscription *filter*. A filter is a a set of predicates (conditions) that are evaluated over event content. An event is delivered to a subscriber if the event satisfies (*matches*) the subscription filter.

The degree of expressiveness depends on the particular implementation. Many pub/sub systems represent events as tuples consisting of a set of named attributes. Most *subscription languages* provide basic comparison operators for attribute values ($=,\neq,>,\geq,<,\leq$), some allowing particularly fine-grained and complex filter languages such as SQL [SBC+98, Var09, FJLM05]. Other pub/sub systems represent events as *XML* [W3C08] documents, using *path expressions*, such as *XPath* [W3C99], as the subscription language [CFGR02, DF03]. Intuitively, there is a tradeoff between the expressiveness of the subscription language and its cost of the matching operation [CRW99].

Type-Based Publish/Subscribe

Type-based pub/sub [BBHM95, Eug01, Pie04] means that each event published (*event instance*) conforms to an *event type*, consisting of a name and a particular (guaranteed) set of attributes of specific data types. Type-based pub/sub can be considered a merging of topic- and content-based approaches, where the event type name represents the topic, and subscription filters can be defined on the attributes of the event instance.[5] In addition to

[5]There is a distinction between *type-* and *type-and-attribute* based pub/sub, which relates to the accessibility of the information contained within an event instance—see [PB02] for details. In this dissertation, we take a type-and-attribute based approach, as described in this subsection.

providing a structured, yet expressive subscription model, type-based pub/sub allows both event instances and filters to be type-checked for conformance with the event type definition [PB02], and facilitates integration with programming languages [BBHM95, Eug01].

A related approach is *concept-based* pub/sub [CABB04], which uses ontologies to abstract client interests away from event-constructs (type/topic). This is to ensure correct event interpretation in heterogeneous environments.

3.2.4 Distributed Publish/Subscribe

It is through brokers that the pub/sub service delivers events from publishers to subscribers. In a single-broker environment, all publishers and subscribers connect to the same broker, which distributes the events as appropriate to the subscription model. Clearly, such an approach does not scale. As such, most research focuses on distributed pub/sub, which involves a network of *brokers* [EFGK03], independent communicating servers that co-operate to route events from publishers to subscribers. Each broker manages a subset of the publishers and subscribers within a system, forwarding messages to other brokers and delivering events to subscribers as appropriate to the routing and subscription model. Scalability is improved, as each broker manages fewer clients. Further, various routing techniques can be used to optimise dissemination [BV06].

Siena

One of the most cited pub/sub frameworks is *Siena* [CRW01], a content-based model that takes a filter-based routing approach to improve the scalability of an event-dissemination service. Clients communicate with brokers, which interconnect to form the distributed pub/sub service. Each broker maintains routing tables to transmit events. The goal is to improve scalability by reducing the number of brokers involved in event propagation. Siena introduces two classes of routing algorithms:

Subscription Forwarding On receipt of a subscription, a broker records the subscription in its routing table before forwarding it to all other directly connected brokers. The process repeats for each broker, to form a *spanning tree* covering the entire network.

Advertisement Forwarding Siena introduces the concept of *advertisements*, which are like subscriptions, but are issued by publishers to declare that they (may) publish particular events. Advertisements are used to set the paths for subscription forwarding. A broker stores an advertisement in its routing tables before forwarding it to other connected brokers, in a similar manner to that described above. On receipt of a subscription, a broker updates its routing table, and forwards the subscription only to connected brokers who have issued advertisements relevant for the subscription. Thus, the subscription moves towards relevant publishers by following the reverse path of the advertisements.

In both approaches, each broker forwards a publication (event) to all directly connected clients and brokers with subscriptions[6] that match the event instance. Thus, events travel the reverse path of the subscriptions.

[6] That is, brokers who have forwarded a subscription.

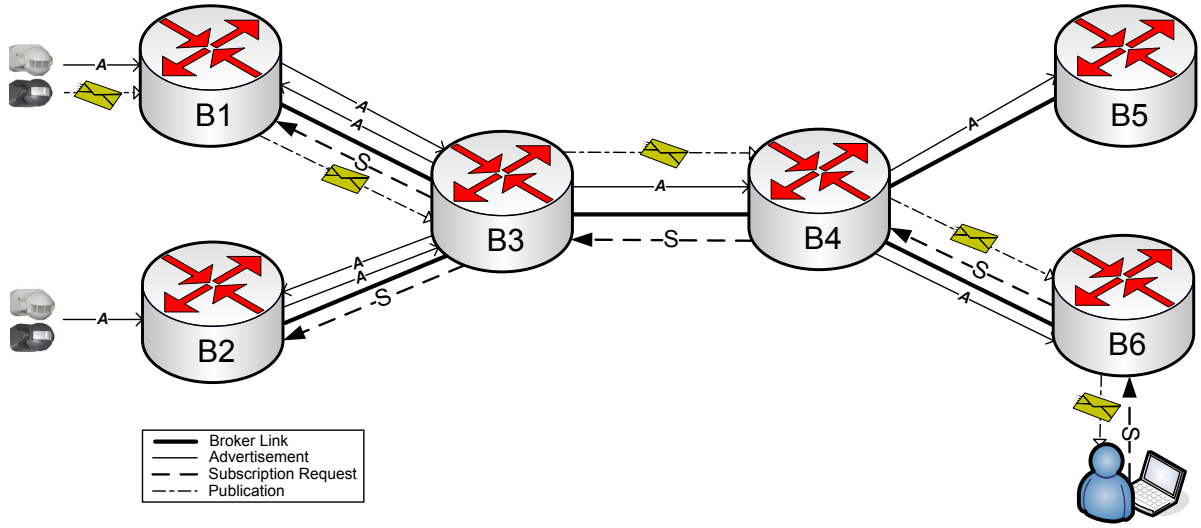


Figure 3.3: Illustration of the advertisement-based routing model.

Fig. 3.3 illustrates the process, where a sensor client issues an advertisement (A) to broker B1. This is propagated throughout the broker network. B2 later receives a similar advertisement from a different client; however this only propagates to B1, as the brokers B4–B6 have already received an advertisement covering similar publications. When the subscriber at B6 issues a subscription, it follows the reverse path of the advertisements, so that the subscription is stored in brokers B4, B3, B2 and B1. When the (sensor) client publishes an event, each broker propagates the message along the path of the subscription until it reaches the subscriber.

By propagating events only down paths with interested subscribers, bandwidth is saved and broker-processing time is reduced. Siena also exploits *coverage* (overlap) between subscription or advertisement filters, so that only the most general is forwarded. This reduces the size of the routing tables and results in transmission of fewer advertisements/ subscriptions. Note, however, that the practicality of implementing coverage depends on the complexity of the subscription language [CRW99].

We describe Siena as its routing approach is similar to that underlying the work in this dissertation. There are, however, other flavours of pub/sub, many of which focus on improving scalability [BV06]. Hermes [PB02], for example, takes a *rendezvous-based* approach, in which the dissemination paths are built by forwarding advertisements/subscriptions towards a particular *rendezvous broker*. This broker acts as the (globally agreed-upon) meeting point for the particular event type, thus avoiding advertisement flooding. Unlike Siena, Hermes focuses on dynamic topologies, considering the addition/removal of brokers and fault-tolerance.

The use of pub/sub is becoming increasingly wide-spread. Prominent research efforts include Siena, PADRES [FJLM05], Rebeca [MFGB02], Hermes [Pie04], Jedi [CNF01] and Elvin [SAB+00]. Commercially Microsoft describes a pub/sub design pattern [Mic09b], while IBM's WebSphere [IBM09b] inherits some functionality from Gryphon [SBC+98]. *Web Services Notifications* [Com06b] and *Web Services Eventing* [W3C06] are two competing standards describing pub/sub communication for webservices.

For an overview of pub/sub models, see [EFGK03], [BV06] and [MFP06].

It is important to note that most pub/sub implementations concern *application-level (overlay)* routing, where the pub/sub service is built as an application, using TCP/IP or some other network protocol for communication. As illustrated in Fig. 3.4, here brokers form an overlay network, an abstraction above the underlying network, where a (persistent logical) connection between two event brokers might span several hops and may be short-lived at lower-levels. That said, there is some research into the use of pub/sub in the lower levels of the network stack [PSI].

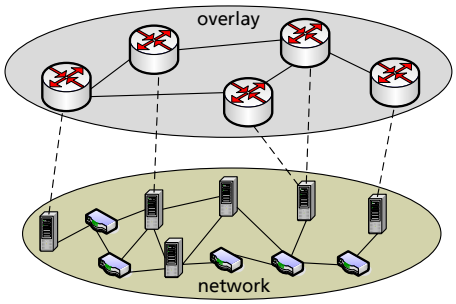


Figure 3.4: An illustration of an overlay network.

3.2.5 Request-Response Interaction

Although not considered a message-oriented middleware, *request-response* is a popular interaction paradigm underpinning the *client-server* model of communication. Request-response involves communication between the consumer and an information *provider*.[7] It is a tightly-coupled form of interaction, where a consumer initiates communication by issuing a request *directly* to a provider, located a particular address. Security is simplified as parties directly communicate, and thus 'know' each other. However, this coupling has limitations. Firstly, it is required that the information providers and their locations are known, or are able to be determined by consumers. This can become an issue in an environment with many communicating entities. Second, such architectures bring issues of scalability, as popular providers may become overloaded in processing a number of requests, and consumers interested in data must maintain direct connections with each information provider. Thirdly, changes and system evolution requires the modification of all dependent application components.

3.2.6 Healthcare Communication

As explained, healthcare providers must share information. As health is a data-driven environment, clients require notification of incidents (events) as they occur. The services provided, and hence information requirements, vary depending on the provider's role in the care process.

Such an environment benefits from the properties of the pub/sub paradigm. In healthcare interactions occur between many different entities. Consumers register their interest with a broker while producers publish events as they occur; the middleware is responsible for delivering the information appropriately. The decoupling of producers and consumers is appropriate for large-scale environments, as clients are not burdened with the addressing

[7]The provider serving the information is not necessarily the information producer.

specifics of every potential information source/sink. In this way, clients focus on the information, rather than possible sources. This simplifies the development and maintenance of client applications. However, the decoupled nature pub/sub also brings security issues. These are explored in §3.4.

The scalable nature of pub/sub systems is important in a sizable application domain such as a national care service. Often a particular health incident will be relevant to a number of clients, grounded in various domains. The model serves to reduce redundant transmissions (multiplicative event fan-out), by routing a single copy of an event as far as possible. Domains require the ability to manage their own network/server loads. Rather than dealing with bottlenecks in each application, a common middleware facilitates network manageability, e.g. through the addition of brokers or topology reconfiguration.

This dissertation considers an integrated database-pub/sub system using an advertisement-based forwarding mechanism. This is described in the next chapter.

3.3 Access Control

Given the sensitivity of health information, access to healthcare data must be carefully controlled. The purpose of *access control* is to restrict access to resources. Access control mechanisms define the *permissions* of principals (users) to access objects.[8]

With a history of use in military systems, *mandatory access control* (MAC) concerns the central allocation of privileges for an entire system. Some prominent access schemes featuring mandatory controls include Bell-La Padula [BP73], Biba [Bib77] and the Chinese-Wall [BN89] model. MAC is notably inflexible, requiring policy to be defined for all actions/operations in the system. Such schemes are inappropriate for an environment such as healthcare where there are multiple domains of control.

Discretionary access control (DAC) differs from MAC in that it allows user (or *ad-hoc*) specification of privilege. This brings flexibility, for example, allowing the encapsulation of exceptions. The *Access Control Matrix* [Lam71] is a model in which access rights form a matrix of privileges, where each principal represents a row and each object has a column. A cell in the matrix defines the access rights for the relevant principal and object. Given that such a matrix is likely to be sparse in a system of any size, implementations tend to partition the matrix. *Access-Control Lists* (ACL) represent the privileges for each object (a column of the matrix) and are typically used by operating systems for file security. *Capability-based* systems [Lev84] represent privileges available for each principal, i.e. a row of the matrix. Both present issues of manageability, such as when a new principal or object is added.

3.3.1 Role-Based Access Control

Role-Based Access Control (RBAC) [FK92, SCFY96] introduces a layer of abstraction (a *role*) between principals and privilege. As shown in Fig. 3.5, principals are assigned to roles, and roles are assigned privileges. This works to improve the manageability of access constraints: a new principal is assigned roles defining their privilege, while the privileges

[8] We use the term *object* to refer to a target for the privilege. An object may refer to a resource (file), action or data (record/column/event).

of all role-members are modified through a single operation. The motivation is that users join, leave and change roles in an organisation, while the policy concerning the objects is often independent of such changes. RBAC concepts are found in a variety of systems including databases, operating systems (security models) and webservers.

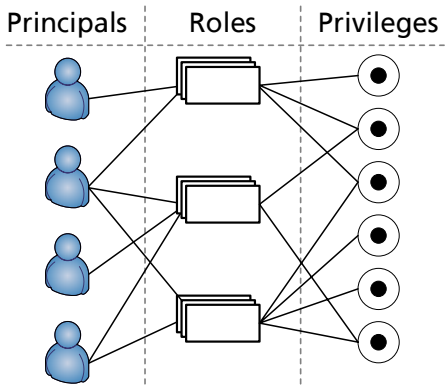


Figure 3.5: An illustration of RBAC assignments.

Extending RBAC

There exists much work in the area of extending RBAC. For example, the *Tees Confidentiality Model* [LLN03] presents a combined identity and RBAC based model that introduces specific *confidentiality permissions*. These permissions are assigned to objects to encapsulate particular privacy concerns. To assist in privilege management, the model defines *collections*: hierarchical groupings of roles, identities and data items enabling privilege inheritance. Confidentiality permissions are processed in a strict order; though the model allows permissions to be specifically overridden. *P-RBAC* [NTBL07] extends RBAC to enforce both privacy and access control policies, where roles are assigned privacy permissions stating the purpose and conditions for the permission, and any obligations to be performed. Context-aware RBAC is also a popular area of research—[HW04] provides an overview of some efforts in this area.

3.3.2 RBAC and Beyond

Some approaches consider access control in terms of a more extensive control framework, though often include some concept of a role. PERMIS [CZO+08] is an authorisation framework consisting of a RBAC implementation aimed at controlling privileges in a distributed (grid) environment. It allows stateful policy decisions and includes mechanisms for validation and delegation. *XACML* [OAS05a] is a well-known model for expressing and interpreting authorisation policies written in XML [W3C08]. It provides a policy definition language to define and enforce complex, fine-grained policies while allowing interchangeable, application independent policy specification. XACML aims to be expressive through customised definitions, also providing the mechanism for conditional authorisation, policy combination and conflict. The XACML RBAC profile [OAS05b] describes the method for realising RBAC in XACML.

Policy models govern actions in distributed environments. *Law Governed Interaction* (LGI) [MU00] is an approach in which trusted controllers interpret and enforce policies on behalf of their clients. *KaOS* [UBJ+04] takes an ontological approach to policy specification, supporting both authorisation and obligation policies in the realm of the *Semantic*

Web [W3C01]. *Ponder2* [TLDS08] is a stand-alone policy environment used to control system behaviour. The model includes authorisation policies [RDD07] that specify the actions permitted on objects, and obligation policies, which trigger actions in response to an event. Focused on environments of autonomous interactions, Ponder2 groups system components into self-governing administrative domains (*self-managed cells*) which use policies to manage themselves and their interactions. Ponder2 is extensible, in that every component is a managed object, which interacts with other objects to provide system functionality. Communication typically occurs through a pub/sub event bus, scoped for the particular domain.

Access policies tend to be generally defined, residing in centralised or distributed policy stores. *Sticky-policies* [KSW02] differ in that they are attached to, or reside with, the data itself. In a distributed system, such policies propagate with messages containing the data. Sticky policies raise issues of trust, in that the remote service must be trusted to enforce the policy, while an enforcing node must trust that the policy in the local environment is appropriate. Further, in a distributed environment, issues concern the capability of services to interpret, process and enforce policy, as this requires common state, definitions (e.g. types) and operations (e.g. functions). Application-independent policy components enable the enforcement of sticky policy obligations that apply to multiple nodes of a distributed application [CL08].

Hippocratic databases [AKSX02] build data disclosure controls into the database system. This reduces an application's data-management concerns. The approach involves defining *privacy metadata*, policy that works to control the collection, query, retention and disclosure of data. Noting that record/tuple control may be insufficient, methods are described to restrict information disclosure [LAE+04] at the field, table or query level, implemented through query rewriting or the construction of data views. Auditing and data inference techniques are described to detect information leaks [AEKV07]. Also considered are techniques for privacy-preserving data mining [BA05, EGS03], particularly useful where health data is used for secondary purposes. IBM is actively promoting such technology as part of their push towards *data-centric security* [BOS+06, IBM09a].

3.3.3 Healthcare Access Control

Protecting the confidentiality of health information is clearly an access control issue, concerning the appropriate circumstances for data access.

This is reflected in the NPfIT (§2.5) which proposes a RBAC service and a number of supporting services to control access to information. These services are described in §2.5 and §6.2.1. One access control effort of the NPfIT is that of sealed envelopes, in which particular information from a patient's record can be hidden from 'general' view. As described, the sealed envelopes are visible, but access to its contents is strictly audited, while locked envelopes are only visible to the patient and particular care team involved. Clearly such an approach causes concern, one would expect that *all* clinical data is stringently protected, revealed only when necessary. Hippocratic databases are marketed at organisations to assist in complying with US healthcare information regulations, by controlling the medical data returned from queries.

Some work in RBAC aims at extending the model with a particular focus on healthcare, e.g. [WFSM02, HW04]. These approaches integrate (medical) context into the access-control framework. Becker [Bec07] argues, in presenting *Cassandra*, that RBAC is, by

itself, unsuitable for handling the complexities of NHS policy. Cassandra [Bec05] is a formal policy language aimed specifically at specifying NHS authorisation policy. Based on datalog, NHS procedures governing particular actions are encoded in logic-rules.

In a medical security recommendation paper for the British Medical Association, Anderson [And96] describes a method for controlling access to medical records utilising tightly regulated access control lists. This involves each record having a clinician responsible for maintaining the access control policy for that record (a patient may have multiple records), where alterations to the access control list may only occur with explicit patient consent. Access to the record itself is audited for reasons of accountability.

Healthcare systems typically consider the security aspects for request-response scenarios, occurring between named endpoints; such as a query executed on a specific database to access a patient's medical data, or a directed request to perform a particular action, e.g. signing-on at the start of a shift. In these scenarios, the request is evaluated by the server, the action performed (if permissible) and response sent. Indeed, this is the interaction paradigm for the majority of medical systems. As opposed to accessing a record, here we consider the integration of access-control mechanisms into a pub/sub middleware. We are concerned with the *notification* of incidents, to control the flow of events as they occur in an environment of distributed interaction.

3.4 Publish/Subscribe Security

Most work in pub/sub focuses on issues of scalability, performance and expressiveness; there is substantially less literature concerning the security aspect [MFP06]. This is unsurprising, given the decoupled many-to-many nature of paradigm, and because many application scenarios consider anonymous communication and/or the distribution of relatively insensitive information, such as stock-quotes or weather readings. As stated by Wang et al. [WCEW02], security is a challenge in pub/sub given that it involves "information dissemination across distinct authoritative domains, heterogeneous platforms and a large, dynamic population of publishers and subscribers". That said, given that pub/sub deals in information distribution, controlling access to information is clearly of importance. Security considerations must account for the fact that delivery group membership is dynamic, evaluated on each event instance [OP01]. There have been various approaches taken to pub/sub security: some require client involvement, others middleware focused; some involve encryption schemes while others are policy based. In this section, we introduce approaches for pub/sub security, to place *Interaction Control* with respect to other work in the field.

3.4.1 Policy-Based Control

One of the earliest efforts in pub/sub security is that of Miklós [Mik02], which involves using policies to define the validity ranges for pub/sub operations. Built on Siena, a policy consists of an attribute-based filter, much like a subscription. Restrictions are assigned to clients who hold a particular credential. Policies are enforced by exploiting and extending Siena's existing coverage mechanisms, where an advertisement, subscription or notification must satisfy the maximum/minimum security restrictions as specified by the policy filters. Miklós also defines *screening*, which allows specification of the attributes

visible to a particular subscriber. Note that while content-based, Siena is not type-based, thus policies are specific to particular sets of attributes.

Symmetric publish/subscribe [TGP06b, TGP06a] also takes a filter-based approach to access control. Here, the publisher includes constraints, a subscription-like filter, with their publications. This constraint is attached to and thus routed with the event instance. Matching occurs by computing the intersection of the subscription and publication constraints. This allows a publisher to define a class of subscribers for which a particular event is relevant, based on the subscriber's interest. The broker network is assumed trusted.

Opyrchal et al. [OPA07] define *event owners*, clients who conditionally licence others with event privileges. Owners specify client-specific policy rules to both assign and delegate rights. The examples define publishers as owning the instances they produce. This gives a publisher control over their publications, where a publisher assigns (licences) particular clients with privileges to subscribe to and receive their events. Policy is centrally stored and enforced by a single event broker, though issues of distribution are discussed as future work. Also suggested is the use of dedicated applications (agents/proxies) to act as privacy filters to control information visibility.

Wun and Jacobsen [WJ07] present a policy management framework that couples *Event-Condition-Action* (ECA) rules to pub/sub operations. Policies consist of conditional executable actions tied to particular operations of the pub/sub service: `un/advertise`, `un/subscribe` and `publish`. These policies may be specified by a client or set in a broker by an administrator. The action associated with a policy is executed immediately before or after performance of the operation. Policies can be attached to pub/sub messages (sticky policies) for association and subsequent enforcement by other brokers. The authors advocate a *post-matching* policy approach, where the action is executed after the matching phase of the operation. This avoids policy overheads, except for the time taken to execute the policy. The model is generic, able to be used for various purposes such as system monitoring, event processing/composition, workflow triggers and security. The authors provide some security examples, considering content-based firewall functionality, and controlling messages according to security groupings.

Zhao and Sturman [ZS06] present an approach in which a centralised ACL stores the privileges for all clients operating within the system. A broker queries the ACL to permit/deny the request as appropriate to the privileges of the issuing client. A client may publish events that match the applicable publication rules, and receives messages that match both its subscription filters and the applicable subscribing rules. The approach is dynamic as a change in the ACL (privilege) is pushed to brokers hosting affected clients, and has mechanisms for dealing with versioning and timing issues for policy updates.

Belokosztolszki et al. [BEP+03] describe the use of RBAC to secure a pub/sub service. The work integrates Hermes and OASIS to control a client's interactions with their local broker. This allows definition of RBAC policies to control client access to the advertisement, subscription, connection and type management functionality of the pub/sub infrastructure. Policy is specified by event-type *owners*, typically the creator of the event type. This renders a policy type specific. A client must satisfy the policy for the broker to accept/perform the particular operation. *Restrictions* may be defined to limit a particular advertisement or subscription, e.g. downgrading a subscription to an event's subtype, or imposing an additional filter.

Belokosztolszki et al. primarily focus on client access control, enforced at the boundaries (edge-brokers) of the pub/sub infrastructure. Broker trustworthiness is considered at the type (cf. content) level, where a web-of-trust is formed via a certificate chain originating from the event-type owner. Only an authorised broker receives events, advertisements and subscriptions for the particular type. This, however, is described in the context of a single administrative domain. These concepts were extended to address client and broker security concerns in multi-domain environments [BEMP05, PEB07b], enabling decentralised management of roles, message types and policy. The authors employ encryption and key-management techniques to deal with issues of trust within and between domains.

Pesonen et al. [PEB06] further explore control in multi-domain environments, defining a *capability-based* access control model in which clients and brokers are granted privileges by the *access control manager* (ACM) of a domain. Taking a *decentralised trust management* approach, authority is delegated via a certificate from the resource owner to the ACM of a particular target domain. The ACM may then perform further intra-domain delegations to control the access privileges of its clients. Note that although the owner cannot control the intra-domain privilege allocations, they can always revoke the privilege for the entire domain. The client requesting an action presents the pub/sub service with its capability and the chain of delegation certificates linking the domain's ACM to the owner. This allows validation of the capability.

3.4.2 Encryption-Based Control

Encryption is used to hide information from untrusted principals.

Opyrchal et al. [OP01] present an efficient key distribution algorithm to protect information transmitted from brokers to subscribers. However, as pub/sub models typically concern application-level routing, often it is assumed that some *transport-layer security* (TLS) [DA99] exists between clients and brokers.

Raiciu and Rosenblum [RR06] present a method for routing information where encryption/decryption operations are performed by mutually trusting publishers and subscribers who share a common key. Considering all brokers untrustworthy, mechanisms are presented to allow brokers to perform content-based routing without access to the (plaintext) content itself. This, however, impacts on the expressiveness of the filter language.

EventGuard [SL05] presents a suite of security guards in an environment of untrustworthy brokers. Confidentiality is protected through a *per-topic key*, and topic specific *token* that the trusted central authority allocates to approved[9] clients on an advertisement or subscription request. These tokens are required to issue requests to the EventGuard pub/sub service. A publisher encrypts an event with a random key, and the random key with the topic key, before publishing the information with the token. As a subscriber shares the topic key, it can decrypt the random key to access event content. The token obfuscates a topic name to allow topic-based routing while removing any semantic associated with the topic's name.

In *PSGuard* [SL07], the same authors consider security in content-based pub/sub. *Hierarchical key derivation* algorithms are used to create keys for subsets of an attribute's

[9]The trusted central authority is separate from the pub/sub service. The method for defining policy authorising a client request is not specified.

value. Publishers encrypt events according to content, so that subscribers can only derive keys to decrypt events matching their subscription filter. An algorithm, based on encrypted data searching, enables brokers to perform matching operations without access to event content. The authors propose probabilistic multi-path routing to minimise the information a broker can infer from observing routing behaviour.

Pesonen et al. [PEB07a] describe the execution of encryption operations within the broker network. As a client accesses the pub/sub infrastructure through a broker, the model assumes that a client trusts its local (directly-connected) broker to handle event content on its behalf.[10] This allows encryption tasks to be delegated to brokers, without client involvement. Access control is enforced over encrypted event content by controlling access to the keys via *key groups*. Encryption operations are avoided when routing to brokers with compatible levels of authorisation.

Pesonen's approach allows one of two methods of encryption for an event type: *whole-event encryption*, which encrypts an entire event instance, and *attribute encryption*, which associates keys with particular attributes to give finer-grained control over the access to event content. Subscription filters are also encrypted, either in their entirety or per attribute filter, depending on the method of encryption employed. A broker decrypts the event/filter/attributes for the keys it is allocated and performs content-based routing; otherwise routing is based on the type-name.

Khurana et al. [Khu05] also considers the encryption of event attributes, but where brokers are assumed untrustworthy. The approach uses a *proxy re-encryption scheme*, where the sensitive attributes of a publication are encrypted with the key of a trusted external service. On matching a subscription, the broker sends the event to the service, which decrypts and re-encrypts the event using the key of the particular subscriber. The service also verifies and signs events. Routing is performed with the unencrypted attributes, as it is assumed that only some attributes of an event require confidentiality.

3.4.3 Secure Event Types

Secure event types were proposed by Pesonen el al. [PB05] to control the management of type definitions in a pub/sub service. The components of a secure event-type definition are presented in Fig. 3.6. The naming components uniquely identify the event type, which include a descriptive name of the type, and the key of the *type issuer* (author or owner) to create a naming scope for the issuer and to facilitate type verification through signing. Versioning assists type evolution, providing the means to avoid conflicts with previous definitions still in use by the system. The *Body* consists of the standard attribute (name/data type) definitions. Delegation certificates enable scalability, allowing a type owner to delegate management capabilities to *type managers*. A digital signature is used to guarantee the authenticity and integrity of the type definition.

3.4.4 Scoping

A *scope* [Fie04, FMMB02] bundles a set of *components* (clients, brokers and other scopes), to control the visibility of event instances. Assuming its components are trusted, a scope isolates intra-scope traffic from the rest of the system. This grouping is independent of application/component concerns, controlling event visibility orthogonally to types, topics

[10] This dissertation assumes a similar notion of trust (§5.2.1).

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

Figure 3.6: Contents of a secure event type definition.

and subscriptions [FZB+04]. In this way, a scope essentially defines and manages information boundaries. Note that access control mechanisms are still required to manage client access to the pub/sub service.

An event is initially visible only to other components belonging to the scope in which it is published. Multiple scopes can exist for a broker overlay—routing tables are partitioned to manage each scope [FZB+04]. A *scope interface* works much like an advertisement/subscription to allow events to move between scopes. This requires at least one broker to belong to the scopes involved in the interaction. To deal with issues of heterogeneity, *notification mappings* can be defined to translate an event instance when crossing a scope boundary. Encryption allows events to be *tunelled* through untrusted brokers where scope components are disconnected.

Fig. 3.7 presents a scope graph, where a publication from C1 is visible to Scope3 and Scope2 and all components (subscribers) belonging to those scopes, including C2. It is also visible in Scope1 if it matches the scope interface specified by Scope3. Scope4 does not automatically receive the event, despite C2 being a member of both.

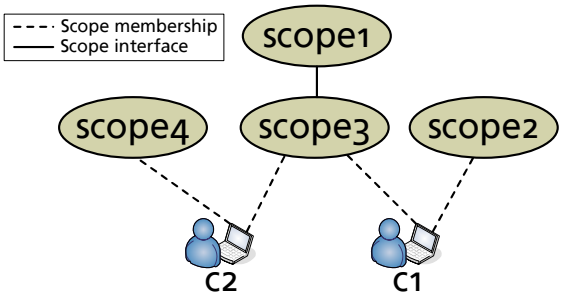


Figure 3.7: An example scope graph.

3.4.5 Applicability to Healthcare

This dissertation considers the confidential nature of health information, where mechanisms are required to control the circumstances in which information is disclosed.

Healthcare brings some interesting concerns to issues of pub/sub security. Firstly, unlike many application scenarios, healthcare is *not* an environment of anonymous interaction. Clients, brokers and domains must be authenticated and verified. Credential services are defined to identify principals operating throughout the entire health domain [NHS06b].

Identity is important, as it allows the assignment of (local/shared) privilege: access to the pub/sub API and access to event instances. Further, identity forms the basis for establishing trust, in that the entity is registered with the health service. This provides insight into the purpose of the information flow, e.g. it is known that a user who is pharmacist deals with medications.

Healthcare is highly collaborative, consisting of a number of interacting entities. Middleware is an appropriate enforcement point for enforcing access control policy, as it abstracts and isolates policy management away from the clients themselves. That is, policy can be defined in the middleware, rather than in each application. As clients communicate through the middleware, all data transmitted is subject to a common control regime. This removes the need for clients to have knowledge of every possible source application and the appropriate disclosure protocol for each event instance for every possible recipient. Despite the fact that the identity of a client is known, it is unreasonable, unmanageable, if not dangerous for clients, or entire applications, to handle the disclosure policy for every potential producer/consumer. A policy failure by middleware may have a greater impact. However, middleware provides application independent policy enforcement, involving fewer policy definition and enforcement points, operating under a common regime and managed under a common interface. Further, as middleware encodes the general sharing protocol of a domain, it is likely subject to greater scrutiny.

Trust is an important consideration regarding information disclosure. Some security models assume a trusted broker network, some assume that a client trusts their local broker, while others distrust all brokers. In healthcare, a base-line level of trust can be established given the legal, social and ethical responsibility for information. That is, users are responsible for maintaining the confidentiality of the information they receive, and a domain is responsible for information it holds.

A domain controls its own technical infrastructure. This allows a domain to define its own policies concerning the information accessible by its staff, and when it may be shared with other domains. An organisation meets its data management responsibilities by releasing data on a need-to-know basis; at which point, the recipient becomes responsible for that information. This means that the infrastructure must allow each entity in the health service to control the circumstances in which data is shared. Responsibility influences trust. A client trusts their domain, typically the client's employer, while a domain trusts its clients and other domains to appropriately handle the information they produce. Any failure brings repercussions, though an entity absolves its responsibility by transmitting only the appropriate information. This requires control within the broker network.

The encryption methods outlined seek to tackle issues of trust. In healthcare, *all* (personal) data is *perpetually* sensitive, in contrast to a military scenario where information remains sensitive until the time of attack. Key management becomes an important, but complicated issue. The liberal distribution of encrypted events throughout a pub/sub network, where key allocations control security, is inappropriate for healthcare. Compromised keys, or a broken encryption scheme at *any time* in the future potentially provides inappropriate access to sensitive data. It follows that health events, even if encrypted, should only flow to parties authorised to receive that information. Information controls should be enforced at each connection, including those between brokers. This avoids unauthorised disclosures and aids in accountability as the recipients of the information are clear—as opposed to the recipient list including every intermediate broker that processed the encrypted event instance.

It follows that a pub/sub implementation for healthcare requires mechanisms for the fine-grained management of event flows. Policy must control the interactions of clients *and* brokers, regulating access to the pub/sub service (connections, advertisements and subscriptions) and the events transmitted. The control mechanisms must also account for the dynamism of healthcare, e.g. where restrictions may be relaxed in an emergency situation.

3.5 Summary

In this chapter we discussed issues concerning healthcare information management with reference to previous work in the areas of pub/sub and access control.

Healthcare is a highly data-driven environment, amenable to an event-based infrastructure. Health incidents must be delivered to the relevant entities, to allow for an appropriate response. Events must be transmitted within and between administrative domains. Pub/sub is an asynchronous, scalable, many-to-many distribution model. Its decoupled nature is useful for healthcare: an event may be relevant to a number of parties, and clients need not enumerate all possible information sources/sinks. Consumers specify their interests in receiving particular information, which may be realised from a number of sources. However, entities are legally and ethically responsible for protecting health data. The sensitivity of health information means that its transmission must be carefully controlled.

This dissertation considers the security aspects of pub/sub as relevant for healthcare. Middleware is the appropriate point for policy enforcement, as it ensures client compliance and facilitates policy management. Clearly client access to the pub/sub service and the delivery of event instances must be controlled. The premise of this work is the notion of local control, local responsibility. Given the overarching legal/social responsibility for protecting health information, domains must have the the ability to control information flows. Regarding pub/sub, healthcare requires the ability to control message flows to each component of the pub/sub infrastructure—*including brokers*. This enables those responsible for information to meet their data management obligations, through precise control over the events transmitted from the infrastructure they control.

4

Publish/Subscribe and Databases

This chapter outlines the motivation for coupling pub/sub functionality into database infrastructure, followed by a description of the operational details of PostgreSQL-PS, an integrated pub/sub-database system on which our data control mechanisms are built.

4.1 Databases

The term *database system* (or *database*) refers to software for data management, controlling the storage, management and retrieval of related data. A database can be thought of as "a computerised record-keeping system" [Dat04]. The concept of a database is well-understood. Databases underlie the information management infrastructure of most organisations, providing persistence for their business processes. As such, data is considered one of an organisation's most valuable assets [Ste97].

A database organises data into a particular structure, according to its *data model*, which influences the mechanisms for storage and retrieval. Many database systems implement the *relational* model [Cod83], representing data as tuples in a relation (i.e. a table). Relational operators, such as `select`, `project` and `join`, work to derive further relations from relations. The *Structured Query Language* (SQL) [ANSI92] is a declarative language standard that facilitates the insertion, deletion, modification and query of relational data. It is the query language of the majority of modern database management systems, though the features (and extensions) implemented vary between systems. Popular database management systems include DB2 [IBM08], Microsoft SQL Server [Mic08], MySQL [SM09], Oracle Database [Ora09b], and PostgreSQL [PGDG09a].

4.1.1 Active Databases

An *active database* [PD99] refers to one in which actions are executed in response to certain events. Databases are typically passive, serving application requests. Active database functionality allows the database to become *reactive* to particular (database)

occurrences. Defining responses in the database, as opposed to each application, removes the need for applications to poll for changes.

Active database functionality is typically implemented through *Event-Condition-Action* (ECA) rules [Cha95]. The *event*[1] component of the rule specifies the operations that trigger the rule, the (optional) *condition* refines the circumstances in which the *action* is taken. In a database environment, ECA rules are also known as *active rules*, where the event is a database operation and the action typically a set of SQL statements [EN04].[2]

Most modern relational database systems include *triggers*, active rules that define procedures[3] to execute on (particular) `insert/update/delete` operations for a relation. A trigger can be defined to fire on the execution of a SQL statement, or for each row (tuple) affected by a statement [PGDG09b]. Triggers allow the execution of compensatory and/or complementary actions to encode application logic, and to ensure data consistency, such as cascading deletes or the maintenance of a materialised view. Triggers are frequently used to realise audit functionality.

4.2 Database Publish/Subscribe

Databases are used to store related data, often holding information from a number of client applications. The occurrence of incidents, actions, operations and/or observations all impact on database state, reflected in the data store through the addition, modification or deletion of data. Often client applications and other databases require notification of such changes. An event encapsulates the data of an occurrence, and often event-data requires persistence; thus, events and databases are tightly coupled. It follows that a database system is an appropriate point for the integration of sharing mechanisms.

A database-broker can function as an event client, as opposed to passively routing event instances. Given that databases represent state through stored data, and support stored procedures, a database can produce and consume event instances, performing various operations (storage/processing) while routing information to interested parties.

There are a number of advantages to such a system. Management operations are facilitated through a common interface for data storage and transmission. Common event and data type definitions simplify persistence and event processing. The coupled infrastructure assists data replication and reliable communication as storage operations can be tied to message flows. The messaging substrate is able to exploit database functionality, which includes persistent storage with powerful data management, manipulation and query capabilities, transactions, and active rules. By reducing communication overheads and leveraging database performance capabilities, such as automatic query optimisation and performance tuning, performance gains are realised over separate messaging and data storage systems [VBM08, Var09].

[1] In this subsection, *event* refers to some database occurrence as opposed to the definition of §4.3.1.

[2] Or some functionality defined in some other database-supported language.

[3] Most relational databases support *stored procedures*: user-defined functions in a variety of languages that are executable by the database system. Such functions are used to execute a prescribed set of SQL commands or to provide additional functionality in other languages not possible with SQL. Stored procedures enable the encoding of application-level functionality into the database system itself.

4.2.1 Continuous Queries

Continuous queries (CQ) [TGNO92, LPT99] provide a mechanism for monitoring change in a database. In CQ systems, a client issues a query that is repeatedly evaluated by the database system. Information is delivered to the client when the conditions specified in the query hold. Example CQ systems include Tapestry [TGNO92], OpenCQ [LPT99], TelegraphCQ [Ber08], NiagaraCQ [CDTW00], SQL Notification Services [Pat06] and Oracle CQ [WBL+07].

CQ and pub/sub systems differ in focus [VBM05]: CQ systems concern consumer-defined queries to report changes in data, while pub/sub concerns distributed event dissemination.

Clients in CQ systems install queries directly (or through a CQ service [LPT99]) to specific databases; that is, the addresses of data providers must be known. CQ systems have issues of scalability, in that each query is stored and evaluated separately, and replies are sent individually to each client issuing a query. Concerning distribution, a CQ system is somewhat analogous to a single-broker pub/sub implementation.

CQ systems require the appropriate triggering functionality to be defined in the relevant database(s), which raises issues of security if clients are left to specify notification specifics through relational statements. In pub/sub, event types partition and advertise the space for which notifications are possible. This restricts subscriptions to a specific, pre-defined scope.

4.2.2 Database-Publish/Subscribe and Healthcare

An integrated database-pub/sub system is useful for healthcare for several reasons. Healthcare is highly data-driven, where health incidents require storage in databases. Given the collaborative nature of the health environment, information must also be shared. For example, the Detailed Care Record service stores only a subset of the information held by the local systems of a particular region [CU07]. As such, it is natural to couple database and messaging infrastructure, to manage information storage and distribution under a common interface. Databases are integral to healthcare processes; such an approach adds a layer above technology already common-place in health infrastructure.

Subscriptions can be used to create information flows (routing paths) between local and remote datastores. In healthcare, the relevance of information is context-dependent. As the purpose of a database is to persist data (state), a coupled infrastructure renders brokers context-aware. This enables the detailed specification of interests, as filters can (potentially) reference anything accessible by the broker. Further, as a database itself can act as an event client, replication and other workflow processes, e.g. triggers to update information or event composition, are facilitated.

Health information is sensitive; thus, the transfer of information must be audited. This implies that the messaging system requires storage. A coupled database/messaging infrastructure assists audit processes (Ch. 10), removes storage redundancy and improves performance over separate storage and messaging systems [Var09].

4.3 PostgreSQL-PS

PostgreSQL-PS is an integrated database-pub/sub system.[4] It extends the PostgreSQL Database Management System [PGDG09a] to provide a database server with event broker functionality.[5] These extensions leverage from, and interact with, various database server components, such as system catalogues, schemata and active rule functionality. This allows for tight integration between the pub/sub and database substrates. PostgreSQL-PS forms the basis upon which we build our data control mechanisms.

4.3.1 Event Model

PostgreSQL-PS uses a type-and-attribute-based event model [PB02, Var09]. Here an event type ε is a tuple of the form $\varepsilon = (n, A)$, where n is a unique, system-wide name for the event type and A is a non-empty set of attributes. An attribute a is defined as a name/type pair, such that $a = (n_a, t_a)$ where n_a is the attribute name and t_a is its data type. Each attribute name is unique per event type definition, where the data type for each attribute corresponds to a valid type recognised by the database system, including at least those specified by the SQL92 standard [ANSI92].[6]

An *event*[7] is the result of a publication, either from a connected client (via a message) or generated programmatically by the system through active rules. An event is a tuple with a set of attribute values (name/value pairs) that correspond to an event type definition: $e :: \varepsilon = \{(n_{a1}, v_{a1}), ..., (n_{an}, v_{an})\}$, where an event e is of event type ε and (n_{ax}, v_{ax}) is the name and value pair of an attribute.

An event type is represented in the DBMS as an *abstract data type*. This integrates the event type and associated schema into PostgreSQL's object-relational model. As such, event types are recognised, accessible and validated as relational objects by various server components (i.e. the parser, the query engine and executor) in the same way as table definitions and function prototypes. PostgreSQL-PS exploits existing database functionality to ensure that event instances, filter definitions and procedural code conform to the relevant event type schema. This is illustrated in Fig. 4.1.

SENSOR_READING (event type definition)	
Attribute Name	*Data Type*
Patient_ID	int8
Heart_Rate	integer
Temp	real
Location	text
Time	timestamp

EVENT INSTANCES (publications)		
2323232320	3434343431	NEA55124
66	87	HIV
38.5	39.12	132
TL447588	TQ290803	15.0
30-01-09 22:11:33	10-02-09 06:12:39	11-04-09 16:02:55
✓	✓	✗

Figure 4.1: The relation between event types and instances.
The first two instances are valid as they conform to the type specification.

[4] See [Var09] for a detailed description of PostgreSQL-PS.

[5] For the remainder of this dissertation, we use the term *broker* to refer to an integrated database-pub/sub broker.

[6] Database types vary depending on the database system and version: PostgreSQL allows user-defined (base) types.

[7] In the literature, this is also referred to as a *notification*, *publication* or an *event instance*.

In PostgreSQL-PS, event instances and types bear similarities to records and tables. The schema of an event type resembles that of a table. A *record* is a tuple, with a set of name/attribute values that correspond to the schema of its table. The difference is that a record relates to data in a table, whereas an event instance is the result of a publication, the details of which can be represented in records across a number of tables (Fig. 4.2).

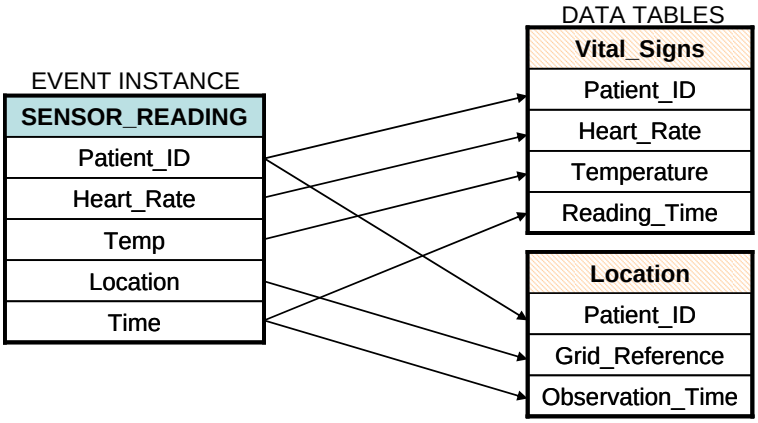


Figure 4.2: Aspects of an event may persist in multiple tables.

The implementation of PostgreSQL-PS introduces some (client-specified) Boolean properties to an event instance:

Guaranteed Determines whether the event is delivered *reliably*—exactly once, in order[8]—or whether a best-effort approach to delivery is taken.

Scope `[Local/Global]` Determines the scope of the event. A *local* event will only be delivered to subscribers directly connected to the broker that receives the publication. A *global* event propagates to subscribers connected throughout the broker network.

Subscriptions

A subscription specifies the interest of a client in receiving particular information. A subscription establishes an *event channel*, which is a unidirectional, typed channel between a broker and a connected client/broker.[9] Multiple channels can exist for a single connection.

Generally a subscription is a tuple of the form $sub = (n_\varepsilon, F)$, where n_ε is the name of the event type for the subscription, and F the subscription filter that refines the subscriber's interest. In PostgreSQL-PS, a filter consists of a set of SQL predicates combined by Boolean operators. In this way, a filter resembles a SQL `where` clause. Subscription filters are associated with an event channel to control the propagation of event instances. A filter is evaluated by the PostgreSQL-PS query engine in the context of an event instance (e), i.e. $F(e) \rightarrow \{true, false\}$. If a filter *holds* (evaluates to *true*), the event will be delivered to the subscriber through the event channel.[10]

[8]Reliable delivery involves persisting the event and periodic retransmission until (positive) acknowledgement from the recipient. Although there is visibility of successful/failed deliveries, there is no guarantee that the client actually receives the event; e.g. where delivery fails due to disconnection and the client never reconnects.

[9]Note there are other definitions of an *event channel*, for example, referring to a bidirectional connection (conduit) through which events of all types propagate [EPTS08].

[10]The literature typically refers to filter evaluation as *matching* an event instance. In PostgreSQL-PS, although filters are evaluated in the context of an event instance, filter predicates need not necessarily reference event content.

Using SQL as the filter language enables highly expressive definitions of interest, as predicates can access event attributes, internally defined database functions and stored data. PostgreSQL maintains a catalogue of operator types and associated operands, which in conjunction with relational schemata are used to validate a filter. Example filters are presented in Fig. 4.3.

```
sensor_reading.patient_id = '2323232320'
```

a) A filter for events pertaining to a particular patient. The `patient_id` is derived from the `sensor_reading` event instance.

```
sensor_reading.heart_rate > 150 AND sensor_reading.temp > 41
```

b) This filter returns events on any patient whose vital signs are of concern. This is evaluated against values from the event instance.

```
emergency(sensor_reading.patient_id) OR
     (sensor_reading.heart_rate > 150 AND sensor_reading.temp > 41)
```

c) This filter is similar to b), except that it also uses `emergency(id)`, an internal Boolean function, that determines whether the patient's status is critical. The function is passed the patient identifier from the event instance.

```
count(*) = 1 from patient_details
           where patient_id = sensor_reading.patient_id and
                 not home_reference = sensor_reading.location
```

d) This filter delivers events when the patient is not home, in that their current (sensed) location is different to that of their (registered) home. This compares the map reference of the event instance against that of the patient's home, as stored in the `patient_details` table.

Figure 4.3: Example subscription filters.

In PostgreSQL-PS, a broker can act as an event client. As such, a subscription may be *internal* or *external*. External subscriptions are issued by clients connected to the broker. An internal subscription means that the subscriber is the local database instance, which consumes events through a locally defined function. In this way, an internal subscription is essentially an active rule defined to apply on the occurrence of particular event instances. This mechanism enables the management of incoming data, e.g. to process and persist event data in various tables.

PostgreSQL-PS introduces two additional properties to a subscription:

Scope `[Local/Global]` Defines the scope of the subscription. A *local* subscription only applies to events published at (this) the local broker. *Global* subscriptions can be satisfied by events published by clients connected to other brokers.

Function The function to consume matching events (internal subscriptions only).

A broker records additional metadata with a subscription to assist in routing; storing the user identifier for external subscribers, or the function identifier for internal subscriptions.

4.3.2 Advertisements

Advertisements are issued to inform the middleware of the potential flow of events. An advertisement establishes an event channel between the producer and the broker. In PostgreSQL-PS, an advertisement is a singleton consisting of the type name: $adv = (n_\varepsilon)$.

4.3.3 Event Delivery

The basic function of an event broker is to route messages. Clients connect and communicate through a broker, which provides the middleware functionality.

In a distributed broker network, brokers co-operate to route information from publishers to relevant subscribers. PostgreSQL-PS uses an advertisement-based content routing approach, similar to that of Siena [CRW01] as described in §3.2.4. Like Siena, PostgreSQL-PS assumes a pre-defined network topology. This is appropriate for database-pub/sub, given that each broker is a database system with specific information responsibilities. Database infrastructure is carefully managed by an organisation; this is different to pub/sub infrastructure aimed at supporting unconstrained peer-to-peer applications. Here, brokers do not anonymously join the network and begin routing; instead, broker connections are defined for a reason, i.e. to store and route particular information. In PostgreSQL-PS the topology of a distributed broker network is described through *links* that specify a broker's (direct) interconnections. The inherent store-and-forward capabilities of database brokers assist in managing failures.

An advertisement is created by a publisher, consisting of a message defining the event type name. Advertisement propagation occurs by the method described in §3.2.4, where an advertisement is sent to a local broker, which forwards it to its neighbour (linked) brokers if they have not yet received an advertisement for this type.

We have previously described the routing process, as illustrated in Fig. 3.3. When a global subscription request is received by a broker, it is propagated by following the reverse path of the advertisement messages. Each connected broker records the subscriptions received from and sent to its neighbouring brokers, noting that the subscription request originated from a remote broker. This forms a dissemination tree from the broker hosting the subscriber to all those advertising the event type. PostgreSQL-PS uses *reverse path forwarding* [Com06a] to ensure that the topology is acyclic, thus avoiding duplicate notifications. This technique uses a distance metric, so that a broker only processes those messages received along the shortest path.

On receipt of a globally-scoped publication, a broker evaluates the subscription filters in the context of the current event. For local clients, if the subscription filter holds, the event is delivered to the client or internal subscription (executing a function). If an event's scope is global, the filters from subscriptions received from external brokers are evaluated. If an external subscription matches, the event is forwarded to the neighbouring broker. Only a single copy of a message is routed to a broker, regardless of the number of subscriptions satisfied by the request.

Unlike Siena, PostgreSQL-PS does not consider routing optimisations based on coverage as it is generally undecidable to detect overlaps in SQL [RI80, Jar84]. This is to provide maximum flexibility: coverage comes at the cost of expressiveness [Var09]. For this reason advertisements are type-based—they do not include filter predicates.

Event Representation & Serialisation

PostgreSQL-PS extends PostgreSQL's data model to include event type definitions. Thus an event is represented as a tuple consisting of name/value pairs. In this way, event schemata closely maps to that of a table, where an event instance is similar to a (table) row. However, an event is not automatically assigned a designated table for long-term event persistence.[11] Events are encapsulated in XML for transmission as it is a platform/language independent standard popular for information exchange. Indeed, XML is the messaging format of the NHS and the UK Public Sector [Cab05].

```
<sensor_reading id="741" guaranteed="true" scope="global">
    <patient_id>2323232320</patient_id>
    <heart_rate>66</heart_rate>
    <temp>38.5</temp>
    <location>TL447588</location>
    <time>30-01-09 22:11:33</time>
</sensor_reading>
```

Figure 4.4: The XML representation of a `sensor_reading` event instance.

All communication between clients and brokers occurs through XML messages. Clients publish events by sending XML messages to a broker. The broker deserialises these messages into a tuple: the internal representation of an event. This is subject to the message being well-formed, conforming to the relevant event type specification. After processing, a broker serialises the tuple into an XML message for transmission.

Reliability

Reliability is particularly important in environments such as healthcare. PostgreSQL-PS considers two aspects of reliability:

Event Processing Reliability *Database queues* (FIFO data structures, akin to an ordered table) and transactions are used to assist in reliable event processing. Three types queues are associated with an event type—`in`, `out` and `exception`. Each queue encodes the schema of the event type, for temporarily storing the content of an event instance, along with other attributes for storing metadata to assist in event processing. Example queue schemata are presented in Fig. 4.5. The purpose of the `in` queue is to store incoming publications, including the event content and publisher details. The `out` queue deals with reliable event delivery, recording matched events and the details of the recipient.[12] The `exception` queue stores events where the execution of the function for an internal subscription fails.

There are two queues created for the `in` and `out` queues: a memory-only, non-persistent queue and a persistent queue using a table to store data. The queue is selected according to the `reliability` property of the publication; non-guaranteed publications use the non-persistent queues, while guaranteed publications use the persistent queue. A single `exception` queue exists to persist all events that fail function execution.

[11] This functionality is added for auditing purposes—see Ch. 10.

[12] Given that health data requires audit, in this dissertation we consider only reliable event delivery (§7.2.1, Ch. 10).

SENSOR_READING_IN								
Patient_ID	Heart_Rate	Temp	Location	Time	user	id	txid	timestamp

SENSOR_READING_OUT							
Patient_ID	Heart_Rate	Temp	Location	Time	subname	id	timestamp

SENSOR_READING_EXCEPTION								
Patient_ID	Heart_Rate	Temp	Location	Time	procname	id	errdesc	timestamp

Figure 4.5: An event type's queueing structures.
The system-assigned attributes are underlined.

Queues are integral to reliability as they allow reprocessing/resumption in the case of failure. On receipt of an event instance, a broker adds an event to the `in` queue. The matching process is presented in Fig. 4.6(a). On successful delivery, an event is removed from the `out` queue. For atomicity, queueing operations are performed within a single transaction. If the transaction fails and is rolled back, the event returns to the queue for subsequent reprocessing. Errors in queueing operations are unlikely, given they merely involve en/dequeueing a system-validated event.

```
For each active subscription:
     Match event against filter

Begin Transaction:
     For each matching subscription:
          Copy event to out queue
     Remove event from in queue
Commit Transaction
```

(a) Event matching procedure.

```
For each internal subscription event on the out queue:
     Begin Transaction:
          Execute subscription function
          Remove event from out queue
     End Transaction

     On Transaction End:
          If Error:
               Rollback transaction
               Copy event to exception queue
               Remove event from out queue
```

(b) Internal subscription delivery process.

Figure 4.6: Transactional and queue operations in PostgreSQL-PS.

Internal subscriptions have greater scope for failure, as they entail executing a user-defined function on the event instance. As shown in Fig. 4.6(b), the subscription function is executed within a transaction. If the transaction fails, after the rollback the event is moved from the `out` queue to the `exception` queue to allow for investigation into the failure.

Reliable Delivery PostgreSQL-PS uses an acknowledgement-based protocol with unbounded sequence numbers to guarantee delivery. This attempts exactly once, ordered

delivery of events between a sender and receiver. The sequence number is encapsulated in the `id` attribute of the event type as per Fig. 4.4. Queues play an important role in this process, allowing the unacknowledged events on the `out` queue to be retransmitted. For further details, see [Var09].

4.3.4 Access Control

The access control mechanisms of PostgreSQL-PS are inherited from PostgreSQL.

PostgreSQL supports transport layer security for its clients through SSL. This can be applied in a similar manner to the connections of the pub/sub service.

Each client is uniquely identifiable by a database-defined username. The username is associated with a connection, meaning that each publisher, subscriber and broker has an account registered in the database. Authentication occurs through an MD5, Kerberos or a custom authentication module.

PostgreSQL provides an RBAC security model to manage the privileges for various database operations.[13] These privileges are automatically enforced by the system. Extension of the model allows the granting or revocation of privileges to `[ ADVERTISE | PUBLISH | SUBSCRIBE ]` to an event of a particular type. Default privileges are defined to protect the integrity of queues. Client-specified subscription filters are validated to ensure that the client has sufficient privileges to evaluate all predicates.

4.4 Summary

Databases form an integral part of an organisation's infrastructure. Pub/sub has been shown to be an efficient, scalable paradigm for wide-scale data distribution. In collaborative environments where data must be stored, audited and shared, a database system is the natural place for the integration of pub/sub functionality. This integration provides a common interface for the management of data definition, storage and transmission concerns. The pub/sub substrate benefits from the advanced data handling capabilities of databases, making brokers context-aware and facilitating data storage and audit, while the database system benefits from a powerful data distribution framework, capable of handling events.

To our knowledge, PostgreSQL-PS is the only fully-integrated, distributed pub/sub-database system available. As such, we introduce PostgreSQL-PS as the framework on which we build mechanisms for controlling data disclosure. The following two chapters generally describe our control model. In Ch. 7 we detail the necessary modifications to PostgreSQL-PS to realise our mechanisms for controlling information flow in a pub/sub-database infrastructure.

[13] Prior to version 8.1, PostgreSQL security was managed through an access control list.

5
Interaction Control

Publish/subscribe is an effective model for wide-area information dissemination. However, there is tension between the convenience of open information delivery, and the need to protect data from unauthorised access. This chapter introduces *Interaction Control* (IC) as a layer above that of a general pub/sub service, to allow fine-grained control over the circumstances for data transmission. IC integrates context-aware policy rules into event brokers to restrict and transform information flows in accordance with circumstance. This allows those responsible to meet their data management obligations, while the enforcement of access control policy in middleware ensures client/application adherence.

In this chapter we describe IC mechanisms and their position within a pub/sub architecture. In subsequent chapters we detail the specifics of context, policy specification, policy conflict and the integration of IC into PostgreSQL-PS.

5.1 Motivation

Healthcare involves interactions between many clients, whose information requirements vary depending on the tasks they provide as part of the care process. Clients are grounded in administrative domains: groupings under common administrative policy that are associated with providing particular health services (§2.3.2). Each domain manages (and trusts) their own infrastructure to handle the specific information requirements of their operations. It follows that each domain collects and holds information *relevant* to the service they provide.

The provision of care services requires information sharing between clients and systems, often across administrative boundaries. As health information is sensitive, sharing must occur as appropriate: in line with consent, the interests of the patient, and legislation. Given the legal requirement to maintain confidentiality, information is best shared on a need-to-know basis. For this reason, communication in healthcare is not anonymous; there is some knowledge about a client and the reasons for which they require information. For

instance, a subscriber from a pharmacy is known to deal with dispensing details; their data requirements differ from those of a doctor in a surgery, or from a research institution. At an individual level, certain information may be relevant to particular doctor but not others. This allows the formation of general policy, subject to consent, concerning the circumstances for data transmission.

The aim of this work is to enable those managing an administrative domain to define the circumstances in which the information it holds/receives is shared, both with local clients and those in external domains. IC gives those responsible for data the ability to meet their data management obligations.

5.1.1 Middleware Control

To recap, pub/sub is appropriate for highly-collaborative environments, such as healthcare, where an event might be relevant to a number of clients grounded in various domains. This is particularly so where notification is required as an event occurs, e.g. in homecare situations, and to ensure that the disparate data stores in a federated environment remain consistent. Middleware provides the obvious point for enforcing data disclosure policy—as all communication occurs through middleware (Fig. 5.1), client compliance is ensured. The decoupled nature of pub/sub not only simplifies addressing concerns, but removes the burden on clients of maintaining and enforcing policy. In environments where information is sensitive, data flows must be controlled. IC provides fine-grained mechanisms for controlling the information released by a pub/sub broker, through policies that define what data should be disclosed, and when this is appropriate.

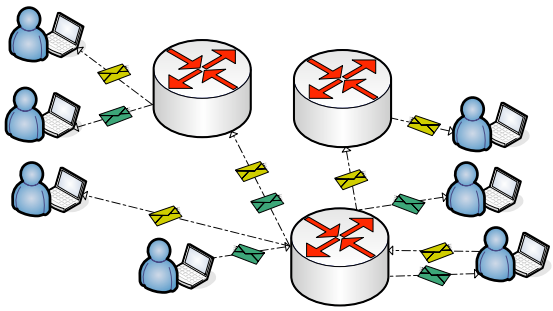


Figure 5.1: All communication occurs through the publish/subscribe middleware.

We define IC mechanisms as a layer above a coupled storage/messaging infrastructure, where data persistence and distribution are managed under a common interface. Such an integration, in addition to performance advantages (§4.2), gives the messaging substrate access to an extremely rich representation of state, through stored data and associated database functionality. This provides the contextual information upon which to base disclosure decisions. The data control layer can exploit the powerful data management and manipulation capabilities of database systems. Event persistence is simplified by local storage infrastructure and common type definitions. Audit capabilities are improved, recording not only the events sent/received, but also the details of the policy and context authorising transmission [Ch. 10].

In this chapter we describe our data control model. Contextual considerations and implementation specifics are detailed in Ch.s 6 and 7. We revisit IC its application to healthcare in Ch. 8.

5.2 Assumptions

Before introducing IC, we first outline the assumptions underlying the approach.

5.2.1 Trust

Issues of trust exist at various levels in healthcare. Here we consider trust from the systems perspective (see also §3.4.5 and §8.4.1). Firstly, we assume that a domain manages and controls its infrastructure. Clients access the messaging infrastructure through the brokers of the domains in which they operate. Therefore we also assume that a client trusts its local (directly connected) broker to appropriately handle information processed on its behalf.

In our model, a broker is also a database system. As such, the broker implicitly has access to data, and stores information produced by the client. The client trusts that the domain will share information only as appropriate, in accordance with the domain's sharing protocol.[1] Such a characterisation is natural, given that the domain's managing organisation typically employs the user of the client application, and manages local applications.

To meet its information sharing responsibilities, a domain defines the appropriate access privileges for its local intra-domain brokers, external brokers and other connected clients. This policy reflects a domain's level of trust with others, operating to precisely control the information released. Brokers simply enforce local access policies: communication occurs only when authorised by its internal policy store.

5.2.2 Other Assumptions

Most pub/sub implementations, including IC, consider application-level routing. As such, we assume *transport-layer security* [DA99] protects connections at lower-levels of the network stack. It is also assumed that every principal in the system is uniquely identifiable, e.g. through the smartcard/PIN services of N3. We do not consider key/certificate management concerns, although such issues will be relevant in an actual deployment.

A number of brokers may be associated with a domain, in line with its information storage and transmission requirements. Here we assume that brokers hold a particular position in the network topology—links exist for a reason, to share particular information with specific brokers. This does not entail that brokers are continually connected, instead the *possible* connections (links) between brokers are pre-authorised (§5.6.1). In IC the connections themselves are dynamic, in the sense a broker may connect/disconnect from another.

To enable sharing, common definitions are required. *SNOWMED* [IHTS, NHS07a] is a clinical terminology used to facilitate unambiguous NHS communications, allowing interoperability between health applications. Similarly, we assume that event type definitions are centrally registered.[2] While this dissertation does not explore type-management specifics, we expect that types are handled in a similar manner as described in §3.4.3. For simplicity, in this dissertation we consider only the *user-friendly* type name.

[1] Which must also account for patient consent.
[2] So too are *fluent* definitions, described in Ch. 6.

5.3 Broker Context

Each data control rule is defined for a particular event type (ε), to control information of a particular semantic.

Information disclosure policy is expressed through context-sensitive policy rules. In a coupled database-messaging infrastructure, context encapsulates anything accessible from the broker, which in our model includes:

1. *Messaging information:* e.g. event types/content, timestamps, current stage in the pub/sub process;

2. *System information:* e.g. schemata, transactional information, audit logs, system time;

3. *Data:* serving to represent state;

4. *Credentials:* e.g. unique ID (i.e. domain/staff/broker ID), qualification, employer, job-role, grounding domain.

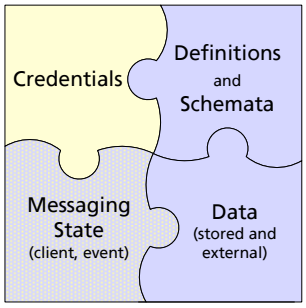


Figure 5.2: Components of context accessible by a broker.

IC policy definitions, in the same manner as subscription filters, reference context through sets of general predicates combined by Boolean operators. Thus policy rules are highly expressive as these predicate sets allow a rule to reference any broker-accessible state.

In this model, we distinguish between credential predicates and environmental predicates. *Credential predicates* assert some characteristic about a principal, including their unique identifier, group memberships, qualifications, roles or delegation certificates that they hold. Such predicates define the class of users to which a rule applies. *Environmental predicates* refer to other aspects of context, such as the current step in a workflow process, system time or patient status. This separation is significant for the monitoring of context.

5.3.1 Permission Attributes

There will be situations where information from outside the broker is required. Permission attributes force a client to include some extra information with a request (advertisement/ subscription) for use in policy evaluation. This external data provides some insight as to a client's intentions, giving background to the request.

A *permission attribute*[3] (α) takes a similar definition to an event attribute, defined as $\alpha = (n_\alpha, t_\alpha)$ where n_α is the attribute name (unique within the evaluation context) and t_α is its data type. A client supplies named-values and the associated data types with their request to populate the permission attributes for a rule.

Permission attributes relate to the contextual predicates of a rule, supplementing broker-accessible context to allow computations with external data. That is, rule predicates can reference the values of the permission attributes specified in a client's request. For example, a permission attribute might require a subscriber to `sensor_reading` events to include the `patient_id` of the patient in which they are interested. This forces the subscriber to direct the subscription towards a specific patient, which the system can use to validate a (treating) relationship between the subscriber and the patient. This improves safety, and avoids evaluating the condition on every event instance.

We have introduced context in terms of general predicates to describe our data control policy rules. In Ch. 6 we describe contextual specifics.

5.4 Interaction Control

We now describe *Interaction Control* (IC), a context-aware policy layer that controls the information released from a broker.

As discussed in Ch. 4, database security can be extended to control basic pub/sub operations. Here we describe context-sensitive restrictions to provide flexible and dynamic data security.

Request authorisation rules define the circumstances in which an event channel is established. This channel is qualified by *imposed conditions*, filters that restrict certain information from propagating, and *transformation rules*, which customise event content to the particular circumstances. Together these mechanisms work to control the release of data.

5.4.1 Request Authorisations

Clients must issue *requests* to a broker to publish (through an advertisement) or subscribe to particular information. If a request is authorised, a directional event channel is established to enable the transmission of the particular type of event (§4.3.1). At the time of request, a decision must be made as to whether this client should generally deal in information of this kind. *Request authorisation rules* encapsulate the policy surrounding the establishment of an event channel. That is, whether this client is allowed to publish, or subscribe to particular information in the current context.

A request authorisation rule A is a tuple of the form:

$$A = (rt, n_\varepsilon, C, E),$$

defined for requests pertaining to a particular event type name n_ε. The *request type* (rt) defines whether the rule applies to advertisements (requests to publish) or to subscription requests. The target of a request authorisation rule is defined by its set of credential

[3]These were termed *mandatory attributes* in our previous publications.

predicates C, which are matched against the credentials held by the requesting client: publisher or subscriber, depending on the request type. The set of environmental predicates E, further refine the circumstances in which the rule applies. An empty predicate set is considered to apply to all circumstances (i.e. a catch-all wildcard). To enable evaluation, a client must supply the permission attributes required by the rule's predicates.

Monitored Conditions

An authorised request establishes an event channel. These channels are durative, persisting until the channel is closed. As rules are context-sensitive, a change in state can affect the applicability of a rule. To account for this, the environmental predicates of a request authorisation rule can be defined as *monitored*, to trigger re-evaluation of the request should the value of the predicate change. All credential predicates are implicitly monitored as they define the target of the rule. If a client's credentials change, the request must be re-evaluated as the rule(s) establishing the event channel may no longer apply.

Request authorisation rules ensure that only valid event channels are established, for particular clients in particular circumstances. Once established, information flow (event transmission) is controlled through imposed conditions and transformation functions.

5.4.2 Imposed Conditions

Imposed conditions restrict the propagation of event instances through an event channel. They are similar to the subscription filters described in Ch. 4, except that they are specified by administrative policy, as opposed to subscription filters which are defined by the subscriber. Fig. 5.3 represents a condition imposed on accident and emergency (A&E) services ensuring that the information (sensor readings) transmitted only concerns emergency patients.

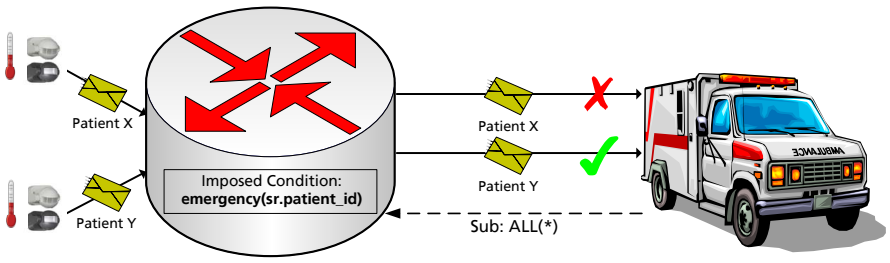


Figure 5.3: A&E only receives sensor readings regarding Patient Y, who is in an emergency situation.

An imposed condition rule I consists of the tuple:

$$I = (ip, n_\varepsilon, C, R, h).$$

The components n_ε and C of an imposed condition share the same definitions as described for request authorisation rules (A). The interaction point ip determines whether the filter applies to an event being published or delivered. That is, it determines the filter's enforcement point. Together, these components specify the target event channel(s) for the rule.

The set of restriction predicates (R) serve to filter the event instances in the channel. They are evaluated in the context of an event instance as it arrives at the interaction point,

to determine whether the event may propagate. These are evaluated just as general predicates, which may reference event content, the permission attributes of the request, and other aspects of state.

If an imposed condition rule is defined for a publication channel, the publication is accepted subject to the evaluation of the predicates in R. However, event delivery must also account for the subscriber-specified filter. As such, an event instance is delivered to a subscriber if all restriction predicates in $R \cup F$ hold, where F is the set of predicates defined by the subscriber.[4]

The Boolean component h defines whether the imposed restrictions are hidden from the client. If hidden, the filters are imposed silently: publications appear to be accepted though are ignored by the system, while subscribers have events filtered without their knowledge of the restriction. Hidden restrictions avoid disclosing any sensitive information that may be encoded in the restriction itself. For example, the restriction `not Treatment = HIV` can prevent information regarding a patient's HIV treatment from flowing to a particular doctor. Revealing this restriction suggests that the patient is HIV positive. There may, however, be situations where a client requires knowledge of the restriction, perhaps to avoid publication errors, e.g. to specify a valid range; or to inform a subscriber that they may only receive data in a particular situation, e.g. sensor data in emergency situations. The lack of feedback regarding a restriction can in some situations have flow-on effects: a client may continue to publish incorrectly, or draw incorrect conclusions by assuming that they are receiving the complete set of information. If an imposed condition is visible, clients are informed of the restrictions imposed as the event channel is established—at the time of the request. Further, publishers are also notified on publication by (negative) acknowledgement if an event fails to satisfy a visible advertisement filter.

From a privacy perspective, it is generally expected that conditions will be imposed on subscription channels. These filters act as a barrier to delivery, being applied at the final stage of the messaging process. Rejecting a publication stops the event from entering the system.[5] This prevents data processing, which may be useful, e.g. for taking remedial actions.

5.4.3 Transformations

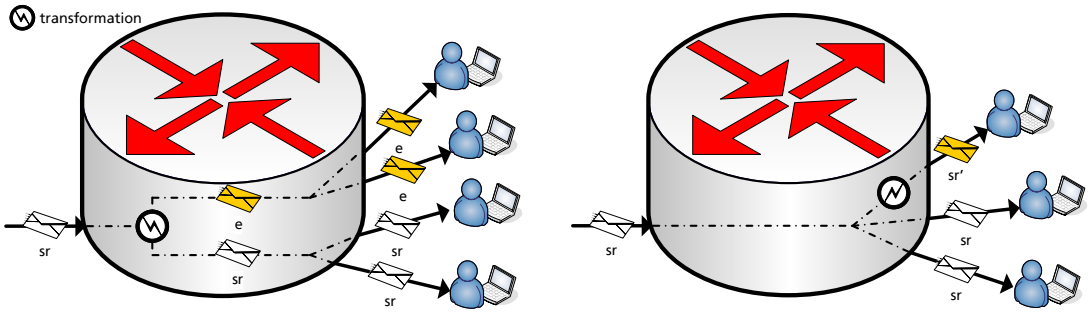
Transformation rules alter the content of an event. They enrich, degrade or produce new events that are related to the original event in some application-specific manner. Transformations can assist in interoperability, altering an event to suit the data models of other systems. From a data security perspective, they allow more than binary (permit/deny) access control as an event can be tailored to the particular situation. This allows fine-grained control over the data disclosed. Further, transformations avoid clients publishing multiple instances of the same event with differing levels of visibility (Ch. 9).

A transformation rule T, is a tuple of the form:

$$(ip, n_\varepsilon, C, G, f, n'_\varepsilon, c).$$

The components n_ε and C are as defined for imposed condition rules. The interaction point ip determines the point in which the transformation applies: on publication or

[4] Multiple rules might impose conditions on an event channel, these are applied in conjunction (§5.7.2).
[5] Regardless, this information is audited (Ch. 10).



(a) Publication (b) Notification

Figure 5.4: Transformation interaction points.

notification (Fig. 5.4).[6] These components relate the transformation rule to the relevant event channels.

In a transformation rule, G represents the set of predicates that define the circumstances in which the transformation is performed. G is evaluated in the context of the current event instance—if G holds, the transformation function is executed.[7]

The transformation is effected through a function f, which takes an event instance of type name n_ε and returns an event that is an altered version of the original or an event of another type. The name of the output event type of the function is specified by n'_ε. A transformation function is locally defined, thus may reference event content, current context, stored data, other database functions and external services.

Transformation rules can be defined as *consumable*, through the Boolean component c. Consumable functions prevent the original event instance from propagating further—only the event resulting from the function proceeds to the next stage of the messaging process. Non-consumable transformations allow both instances to proceed.

5.5 Policy Enforcement

IC policies are defined to apply at a particular point of the pub/sub process. This section details the procedure of a broker enforcing policy in response to a client-issued publication or request. Fig. 5.5 presents the interactions between logical components of the architecture inside a broker.

5.5.1 Request Validation

The procedure of authorising a request and establishing an event channel is similar for both advertisement and subscription requests. This is represented in Fig. 5.6 which presents the sequence diagram showing the interactions between the (logical) components of a broker.

The first stage of the process involves request validation. A request is denied if, in the circumstances, it fails to satisfy the definition of any authorisation rule. Otherwise,

[6] In practice, the selection of the appropriate interaction point depends on the scenario. See Appx. A for details.

[7] Subject to conflict resolution definitions (§6.4.3).

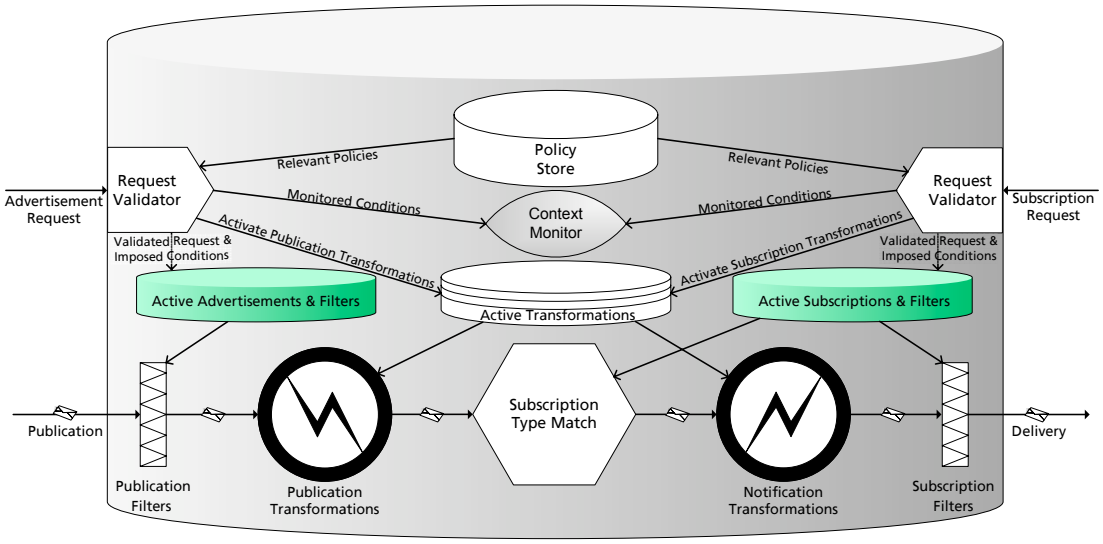


Figure 5.5: The interactions of the components involved in policy enforcement.

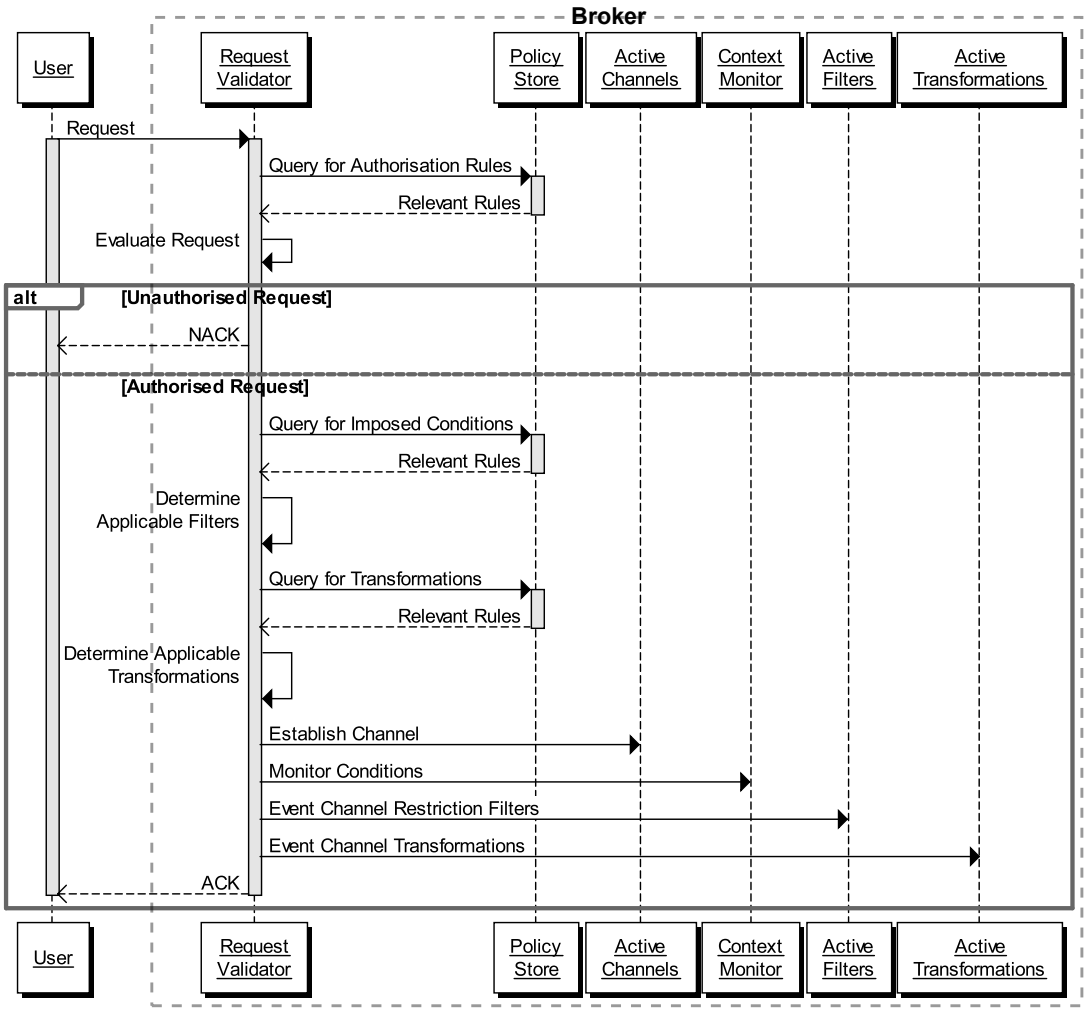


Figure 5.6: Sequence diagram for the request validation process.

the request is authorised. The restrictions relevant to the request are determined by evaluating the ip, n_ε and C components of the imposed condition and transformation rule definitions. The event channel is then established, and the monitored conditions relevant to the request are loaded for monitoring. This is achieved through the use of active rules, which trigger the re-evaluation of a request on a particular occurrence. The set of imposed restrictions,[8] along with any client specified filters (subscription filters) are activated to filter messages as they pass through the channel. The transformation functions applicable to the channel are loaded through the creation of active rules, where a function executes on an event instance when the predicates of its guard (G) hold. At this point, the request processing is complete.

The re-evaluation of a request, as triggered by a monitored condition, follows the same procedure. If, due to the change in circumstance, no authorisation rules satisfy the request in the current context, the event channel is closed. Otherwise, the channel remains and the set of applicable transformation functions and imposed filters are again determined and loaded. This is because the change in context may cause different restrictions to apply.

5.5.2 Publication Enforcement

The restrictions activated for a request are enforced as an event instance arrives at particular points of the messaging process. Fig. 5.7 illustrates the process of enforcing IC policy on an event publication.

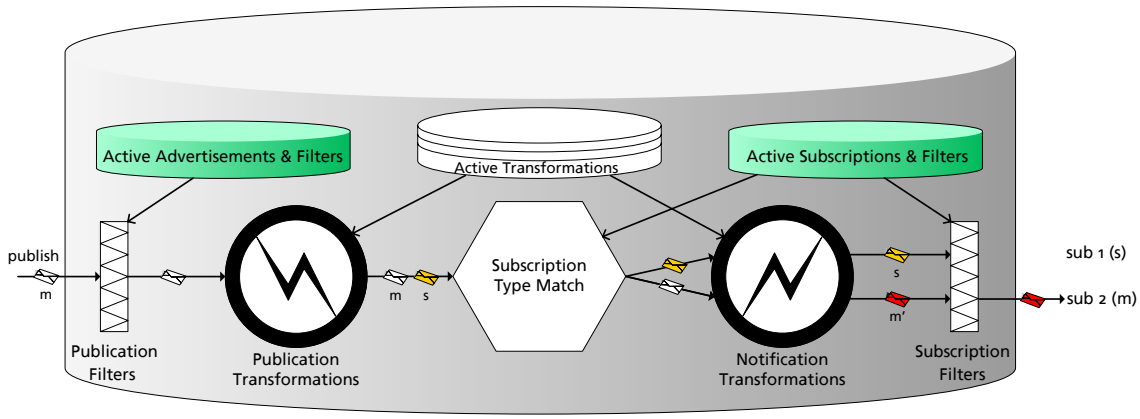


Figure 5.7: Enforcement process for the publication of the event `m`.

The first stage of processing a publication involves validating the event instance against any filters imposed on the publication channel. If the instance satisfies the filters, it is subjected to the relevant publication transformations. In this example, there is one applicable publication transformation function, which takes event `m` and produces the event `s`, which is of a different type. As the transformation does not consume the original event, both events move to the delivery phase of the pub/sub process.

Event delivery involves moving a copy of an instance through each active notification channel for its type. In this example, there exists one subscription for each type. The events are subjected to any active notification transformations applicable to the subscriber

[8]This includes the filters imposed by the specification of permission attributes (§5.7.1).

in the circumstances. Here, a transformation function exists for `sub 2` that consumes event `m` returning a modified instance `m'` of the same type. As there is no notification transformation defined for `s`, it passes through the transformation stage unperturbed. The final step of the delivery process involves evaluating the event instance against the subscription (and imposed) filters. Only event `m'` is delivered, as event `s` fails to satisfy the filter of `sub 1`. Note that internal subscriptions do not involve transmission, instead the function is executed on the event instance. In this way, an internal subscription acts as an active rule responsive to events.

Subscription filters act as a barrier to prevent certain information from propagating. IC involves a two-phase subscription matching process. First, the events are matched against active subscriptions considering only the event type—filter predicates are applied *after* the notification transformations. This prevents notification transformations from circumventing any restrictions.

5.6 IC in Distributed Broker Networks

Brokers interconnect to form a distributed event-dissemination network. As described in Ch. 4, brokers forward advertisement and subscription requests to other brokers to facilitate the propagation of event instances throughout the network. Requests and events are forwarded through *links*, the connections between brokers, which may be associated with a number of event channels.

A broker maintains a set of IC policies to control the flow of information to *directly connected* principals. IC does not distinguish between events/requests received from clients and those received from brokers, who may be forwarding requests issued by others.[9] Instead, an event received through a link is treated as a publication from the remote broker, while an advertisement or subscription received through a link is characterised as a request issued by the remote broker.[10] It follows that IC rules are defined for, and enforced against, brokers just as they are for clients—both are subject to the same policies and enforcement processes as previously described.[11] In this way, the policy of a broker *only* concerns interactions with the next hop. This is in line with notions of local/federated control (see Ch. 8).

5.6.1 Link Authorisation Rules

In PostgreSQL-PS brokers interconnect via a *link*, a connection through which events and requests are forwarded. A link does not guarantee a flow of information, as the establishment of event channels and the transmission of events and requests is governed by other IC rules.

[9]Typically a pub/sub broker only forwards requests; however, an integrated database-pub/sub broker can act as a client, itself issuing requests.

[10]However, this affects routing tables (Ch. 7).

[11]The credentials of the original requestor and the local context are *not* propagated throughout the network. It is impractical for the broker hosting the publisher to define the privileges for every possible principal in the entire distributed broker network. Further, doing so requires the maintenance of separate delivery channels for 'versions' of the same event instance, diminishing the scalability of the infrastructure. The propagation of credentials also raises security issues, as the information could be misused (Ch. 8).

In PostgreSQL-PS, a link is defined to connect two specific brokers, requiring an enumeration of every broker combination. This is unmanageable given the dynamism and scale of the application environment. For instance, a surgery may wish to allow connections from all other surgeries, to deal with situations where a patient requires treatment in a different region. This would require definition of a separate rule for every surgery in the country. As such, we provide *link authorisation rules* to govern broker interconnections. Such rules reference only the credential predicates (C) of the remote broker, and thus take the form $L = (C)$. These rules are simple as they concern only the establishment of a connection—other IC rules work to control channel establishment and event propagation.

5.6.2 Request Forwarding and Processing

IC brings control to the process of establishing an event channel, by characterising an advertisement or subscription as a request requiring authorisation. If a request is authorised by the system, it may be forwarded to other brokers to establish the dissemination network.

The control mechanisms for request forwarding are similar to those of event notification, where requests can be transformed and filtered before being transmitted to the remote broker.[12] Request restrictions are defined by administrators, who have specific knowledge and concerns regarding network topology. The rules relevant to a broker depend on its position in the network infrastructure. As such, we expect request restrictions to set the general boundaries for interaction—specific disclosure issues will typically concern the propagation of data (event instances).

5.6.3 Request Forwarding

Conditions can be imposed on links to filter the requests forwarded to a remote broker. This can ensure, for example, that advertisements are only forwarded to brokers authorised to subscribe to the event type, or that subscriptions are only forwarded to remote brokers with some relationship with the patient to which the request pertains. Such rules (I_r) take the following form:

$$I_r = (rt, n_\varepsilon, C, R_r),$$

defined for a particular request (rt) and event type (n_ε). The set of credential predicates (C) refer to that of the remote broker.

Rules are defined to control the forwarding of the particular type of request (rt): advertisement or subscription. The restrictions (R_r) are evaluated in the context of a request pertaining to an event type, *not* in the context of an event instance. Thus the filter predicates reference *request content*, along with other aspects of environmental state.

Request Transformations

The purpose of a request transformation rule is to tailor a request specifically to a remote broker. Request transformations facilitate interoperability, e.g. to translate a patient identifier from a local system to a shared NHS ID, in addition to protecting disclosure, e.g. removing any sensitive information contained within the request.

[12] Given that requests construct the network, they are only transmitted to brokers.

The difference between request and event transformations concerns the input. Event instances represent data, where transformations produce event instances, perhaps of different types, encapsulating different semantics. Event transformations are specified for a number of reasons including consent preferences, interoperability, event composition and data fusion. That is, event transformations concern information delivery. Requests serve a different purpose. A request only propagates to brokers for the purposes of routing.

Request transformation rules alter a request before propagation to another broker. A request transformation rule (T_r) takes the form:

$$T_r = (rt, n_\varepsilon, \varsigma, G_r, f).$$

This definition differs slightly from that of event transformation rules (§5.4.3). Firstly, instead of an interaction point, the request type rt defines the type of request to which the rule applies. The rule's guard (G_r) is evaluated in the context of the request. The transformation function f takes a request and returns a modified one. Request transformations implicitly consume (i.e. replace) the incoming request tuple. This is because unlike event transformations, a request transformation is focused at controlling the routing tables—the output is always a request tuple.

Request transformation rules are defined specifically for a link. This is defined by ς, which uniquely identifies the link authorisation rule (L) allowing the connection to the remote broker. Events convey information, therefore it may be appropriate to propagate multiple events. A request, however, serves to establish a channel for the particular type, and thus only a single transformation function may be defined for an event type for each link.

Enforcement Process of Forwarded Requests

The sequence diagram for enforcing restrictions on forwarded requests is shown in Fig. 5.8. The figure illustrates a scenario of three brokers, where *Broker 2* acts as the intermediary, constructing the event channels to forward events from *Broker 1*, which hosts a publisher, to *Broker 3*, which hosts a subscriber.[13]

On the connection of a remote broker via a link, the relevant advertisement restrictions for that broker are loaded. This is done on connection, as unlike a subscription, advertisements are not transmitted in response to a request.

When a broker receives an advertisement request, it is validated and the appropriate restrictions are applied (§5.5.1). If the (authorised) advertisement is received through a link, the subscription forwarding rules relevant to the remote broker and the advertised event type are loaded, before the advertisement request is acknowledged. The advertisement is then forwarded to each remote broker after the execution of the request transformation, subject to validation against any request filters defined for the broker and the event type. Forwarding only occurs to brokers who have not yet received an advertisement for this type, as determined by the broker's routing tables (§7.7.4).[14]

The process is similar for subscription requests. After the establishment of the subscription channel, the subscription request is forwarded to remote brokers advertising that

[13]In this scenario, the brokers authorise the establishment of the event channel; thus, there are no NACKs. We begin the illustration after *Broker 2* and *Broker 3* are connected.

[14]In our implementation, we further restrict advertisements to being forwarded only to brokers with the possibility of subscribing, through some authorisation policy, to the particular event type (see §7.7.3).

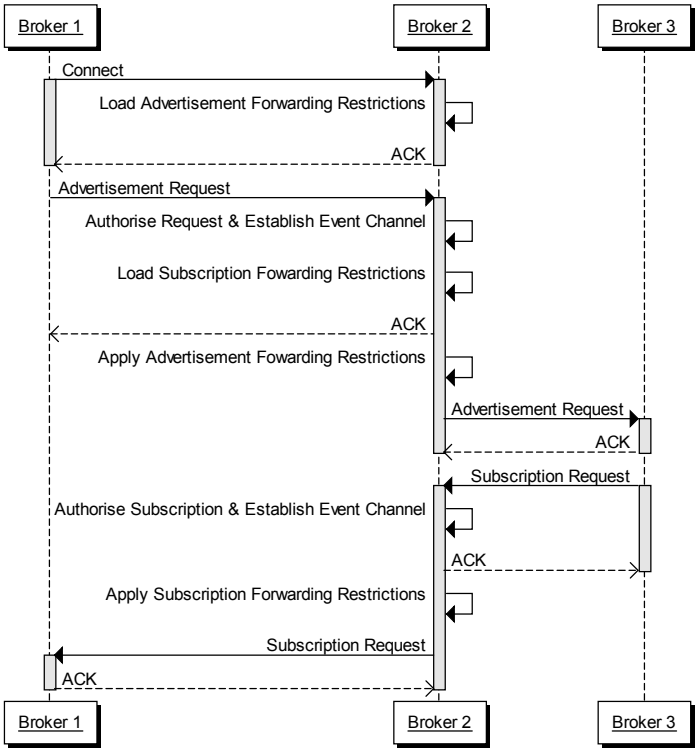


Figure 5.8: Sequence diagram for channel establishment across brokers.

event type, subject to any subscription transformation functions or filters defined for the advertising brokers.

Re-Forwarding Requests

The purpose of a request is to establish an event channel, which if authorised, persists until it is closed. As policy rules are context-sensitive, monitored conditions are used to cause a re-evaluation of the request associated with an event channel should conditions change. By default, a change in an event channel, such as its closure, must be forwarded throughout the network, updating and revoking advertisements and subscriptions as appropriate.

Request forwarding also concerns denial: a change in context might now authorise the forwarding of a request that was previously disallowed. This is of significance where the event channel pertaining to the request is still active. Consider the situation where a subscription concerning a patient is only forwarded to domains involved in their care. Initially, the request was not forwarded to a broker of a remote domain, as the domain had no treating relationship with the patient. If some time later the domain begins treating the patient, the subscription request should be forwarded to this broker, assuming the subscription channel is still active. The issue is similar to the maintenance of pub/sub routing tables, where, for example, the receipt of a new advertisement triggers the forwarding of existing subscriptions. However, here the concern is at a higher-level, influenced by application-level context.

Monitored conditions are used to (re-)forward the requests pertaining to the active event channels for a particular broker and event type (when appropriate). This involves propagating the original request through the request transformation function and request filters before delivery to the remote broker.

Fig. 5.9 outlines the process of forwarding (transmitting) a request to a particular broker. The existence of an event channel pertaining to the request means that the request was previously forwarded—implying a re-evaluation. Clearly, if a request is unauthorised, it is not forwarded; however, if a channel exists, the original request is revoked to close the channel with the remote broker. An authorised request is forwarded if no channel exists. However, if a channel exists, the current request is compared to the original request that established the channel. To avoid redundancy, an identical request is not transmitted; alternatively, if there is some difference (e.g. changes in permission attributes) between the requests, the original request is revoked and the current request forwarded. This re-establishes the event channel in line with the updated restrictions.

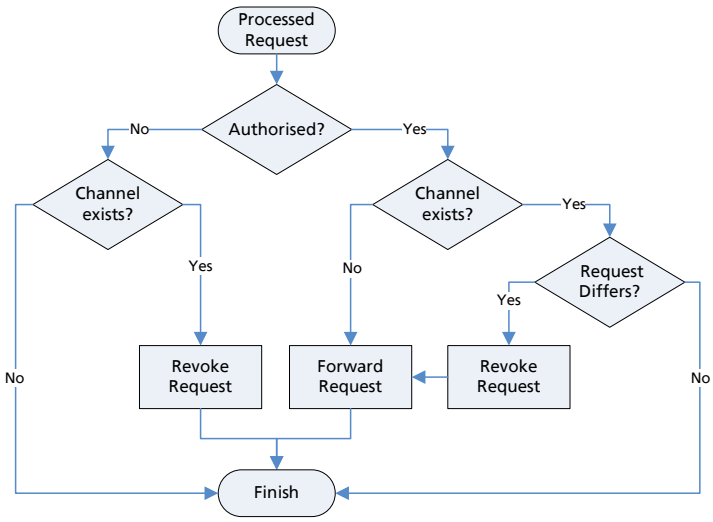


Figure 5.9: The process of (re-)forwarding a request to a broker.

A record of the requests pertaining to active event channels are maintained in the routing tables of the brokers (§7.7.1, §7.7.5).

5.7 Relationships Between Rules

IC mechanisms operate at various stages of the messaging process. Here we describe the relationships between rules. As rules are context sensitive, it is possible that several may apply at a particular enforcement point. Indeed, it is often appropriate, if not required, that multiple rules apply. Here we consider the general case, where it is assumed that rules do not conflict. Conflict is discussed in Ch. 6, where resolution operations can resolve policy conflicts at an enforcement point.

5.7.1 Rule Fragmentation

Each authorised request causes the creation of an event channel. This triggers the loading of the relevant transformation and imposed condition policies, which are enforced as events move through the channel. Request authorisation rules do *not* encapsulate transformation or restriction rules; instead, rules are defined independently. This allows a many-to-many relationship between rules, where different sets of rules can apply in different circumstances (Fig. 5.10). Policy separation avoids re-authoring rules to deal with specific requests, e.g. to avoid rewriting all authorisation rules concerning an event

type when a patient has a consent requirement that involves a restriction (imposed condition). Further, this separation avoids complexities where rules apply to different sets of credentials, e.g. if an authorisation rule applies to a `Doctor` or `Nurse`, but the transformation applies only to `Nurse`s, the definition must be careful to ensure that a `Doctor`'s event stream remains unperturbed. Rule separation directs policy to particular messaging operations targeted at particular classes of client.

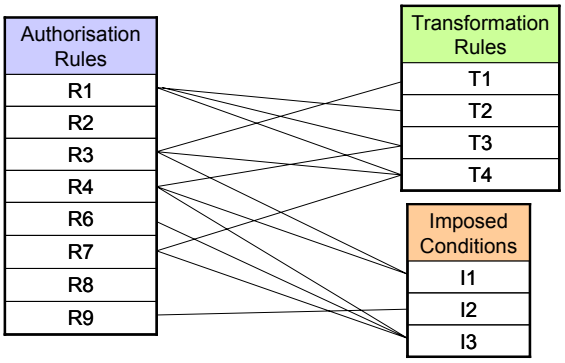


Figure 5.10: Rules of different types apply together, depending on the circumstances.

Context Revisited

If the permission attribute of a request authorisation rule corresponds to the definition (name/type) of an attribute of the event type, the attribute value operates as an equality filter for the request. This works to constrain an event channel, e.g. as per the example of §5.3.1, to ensure that sensor events pertain only to a particular patient. To illustrate, an authorisation rule might require the inclusion of `patient_id` to ensure that the subscriber has a treating relationship with the patient. This identifier is also used as a filter, to ensure that the events delivered correspond to the supplied ID. In this way, permission attributes serve to direct a request, which limits potential 'data-leaks' from (relatively) unconstrained subscription filters.

A change in the state of a monitored condition causes a request to be re-evaluated. The credential predicates (C) of a rule definition are implicitly monitored. It is important that a change in credential allocation triggers re-evaluation of the request, as credentials specify the targets of an authorisation rule. Environmental predicates (E) can be defined as monitored; they are not implicitly monitored as this may be inappropriate. For example, if paramedics can only subscribe to a sensor stream in an emergency, they may still require access to the sensor stream for some time after the (perceived) emergency to ensure the patient is stable. Monitoring does not apply to the predicates forming the execution conditions for transformations (G) and the filters of imposed conditions (R), as these are evaluated on the arrival of each event instance.

Permission attributes and monitored conditions facilitate the management of event channels (requests). Without them, tasks such as enforcing a relationship between the client and a patient—through an imposed condition—becomes cumbersome, requiring evaluation of the relationship on *each* event instance. Instead, conditions can be evaluated once, on construction of the event channel, here by specifying the patient of interest and monitoring for any change in the relationship. This reduces the number of potentially expensive filter evaluations. Safety is improved, as policy/contextual errors denying or

cancelling the request become immediately evident to the client, e.g. a denied subscription request for a particular patient ID indicates a problem in the patient-carer relationship registry.

5.7.2 Application of Multiple Rules

As rules are context sensitive, it is possible for multiple rules of the same type to apply at an enforcement point.

A request must satisfy an authorisation rule to establish the event channel, otherwise the request is rejected. If a client satisfies the credential predicates of the rule without providing values for the permission attributes, a decision cannot be made. As such, the request is also rejected (if no other rules apply) as evaluation cannot proceed. A client can be prompted to retry the request with values for the appropriate attributes.

There may be a number of rules that authorise a request; however, one is sufficient to authorise the event channel. On channel establishment, the monitored conditions, including credential allocations, defined for the authorising rule are loaded for system monitoring. The channel persists until a change in value of any of these monitored conditions, at which point the request is re-evaluated.[15] Again, re-evaluation does not necessarily entail channel closure, as another rule may authorise, and thus maintain, the channel—though the restrictions imposed on the channel may change.

There may be a number of imposed condition and transformation rules that are applicable to an event channel. The filter predicates of an imposed condition are loaded, to be evaluated in the context of an event instance in conjunction with the subscription filter (if applicable) and other imposed conditions. Included in this evaluation are any filters derived from the permission attributes (per §5.7.1).

An event instance, in the appropriate context, might cause a number of transformation rules to apply. In this situation, the transformation functions are executed in parallel, in the sense that each function takes as input (a copy of) the original event instance, the output of which moves to the next stage of processing. This is illustrated in Fig. 5.11. A transformation function is executed only once per event instance at an enforcement point, regardless of the number of policies causing the function to execute. This is to avoid duplicate messages resulting from multiple executions of the same function.

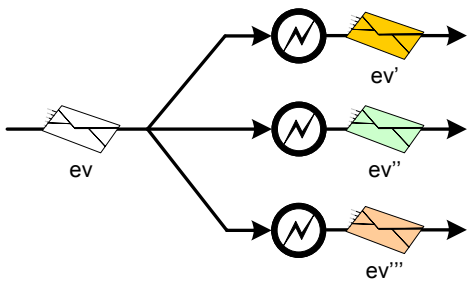


Figure 5.11: The parallel application of (event) transformation policies.

[15] This from a system perspective; of course, a channel can be closed on request.

An output event is not subject to further transformation functions at the same enforcement point. This is to avoid complex transformation loops that can be difficult to reason about, particularly when the output is of the same type.[16] The assumption is that transformations are defined for a purpose, so a transformation function can (and should) encompass all relevant operations for the event. The following chapter describes conflict resolution mechanisms, allowing the specification of ordering and overriding constraints for rule enforcement.

Transformation functions do not directly interact with each other, except in situations of conflict resolution.[17] However, the original event will not propagate if any applicable transformation function is marked as consuming (c) the input event.

5.8 Related Work

In Ch. 3 we described other work in the area of pub/sub security. Here we compare IC to the literature.

Several approaches, such as [Mik02, BEP+03, ZS06], control access by restricting (downgrading) and/or validating request filters. As IC is built in a database environment, filters are far more expressive, having access to a rich representation of state (data and functions) in addition to event content. Further, permission attributes allow external information to be considered as part of an access control decision. This enables fine-grained, context-sensitive event dissemination control.

Some work in pub/sub considers the use of event transformations [SBS08, Fie04, BHM+01, CABB04]. These, however, focus on interoperability and/or event composition. The work of Miklós [Mik02] describes a highly-prescriptive approach to attribute hiding, where a fixed policy applies to all publications. Opyrchal et al. [OPA07] suggest that specific services can act as *privacy filters*, operating to filter/alter the content of events sent/received by clients. These privacy services appear[18] to be implemented as external applications (i.e. as separate clients/agents), rather than integrated into the dissemination infrastructure. IC defines transformation rules for confidentiality purposes, to restrict the information transmitted. Such rules enable more than mere binary (permit/deny) access decisions. Transformations avoid clients publishing multiple copies of an event with differing levels of visibility (see §9.1.6). Further, we provide details of enforcement, defining the specific points in the pub/sub dissemination process where the transformation applies.

Belokostolozski et al. [BEP+03] used RBAC to control client access to the pub/sub API, along with a method for type owners (and delegates) to authorise particular brokers to handle particular event types. This work concerned a single administrative domain. Zhao and Sturman [ZS06] also describe a method to control client access to the API, considering

[16] Note our original implementation applied transformations in sequence, where the output of one function was taken as input for the next. We consider this method inappropriate, particularly where the function changes the event type, or nullifies events or attribute values. The semantics of compound transformations are complex to manage. Therefore, we apply transformations in parallel, which in combination with conflict resolution mechanisms (Ch. 6), provides a more intuitive approach to event management.

[17] As functions are database procedures, rules might indirectly interact by altering stored data.

[18] This suggestion appears in the "Conclusion and Future Challenges" section of the article, and thus lacks a detailed description.

issues of latency and network failure. However, a single ACL is insufficient for a large-scale deployment. Clearly pub/sub operations must be controlled. In IC, context-aware polices are enforced not only against connected clients, but also *brokers*. IC policy enables control over all connections and transmissions, including the construction of routing paths. Further, IC considers control in multi-domain environments.

A *scope* [Fie04, FZB+04] operates as a grouping structure to constrain the visibility of events to its members, and to control the propagation of events to other scopes. Scoping aligns well to the concept of domains. However, it is necessary to control intra-scope communication. The authors note, in citing Belokostolozski et al. [BEP+03], the requirement for client access controls to protect the pub/sub API. An IC broker controls communication to *all* connected principals, regardless of domain structure, in a context-aware manner. Clearly, the notion of a scope is compatible with IC, where scope membership can represent a predicate in a rule definition.

Wun and Jacobsen [WJ07][19] describe a general policy framework for the execution of (ECA) actions in a pub/sub framework. Similar functionality underpins IC in the sense that rules are enforced at particular parts of the pub/sub process. While our IC implementation uses extended ECA rules for policy enforcement (§7.5), we do not attempt to describe a generic ECA framework. Instead, IC specifically concerns information control, defining particular security operations and their appropriate point of enforcement. For instance, notification transformations should apply before the matching operation, to prevent a transformation from circumventing subscription filters/restrictions. Here, this security constraint removes the possibility for, and thus the benefit of, *post-matching* policy application. Further, the generic model does not describe methods for dealing with policy conflict, a necessary consideration in a security implementation (see Ch. 6).

IC takes a data-centric approach to security. Hippocratic databases [IBM09a] are similar in that the database enforces disclosure policy. The focus, however, is different. Hippocratic databases concern query control, while IC considers the context-aware distribution of sensitive information.

Sticky policies involve attaching policy to a message (event or request) for enforcement by other nodes. Some pub/sub security examples include symmetric pub/sub and the work of Wun and Jacobsen. IC takes the position that each broker is responsible for enforcing policy in accordance with its own (local) policy store. In this sense, broker-specific policy reflects higher-level notions of trust. In this dissertation, we do not consider the use of sticky policies due to issues of trust: the policy author must trust others to enforce a policy correctly, and the policy enforcer (recipient) must trust that the policy they are executing is apt. Such an approach can muddle accountability. We do, however, consider this an area worthy of exploration (§11.1).

We have previously described how pub/sub encryption schemes are inappropriate for healthcare, due to the perpetual sensitivity of healthcare information. The issue concerns the distribution of an encrypted event instance. Assuming secure connections (TLS) between clients/brokers, higher-level encryption operations become redundant if IC policy releases only that data the recipient is authorised to receive.

[19] Incidentally, this was developed independently at a similar time to our work.

5.9 Summary

IC enables control over information flows in a distributed pub/sub infrastructure. Control is effected through context-sensitive policy rules that are enforced by brokers at particular stages of the messaging process. These rules provide the ability to control: 1) the establishment of event channels, 2) the flow of data, by filtering and modifying events in transit, and 3) the routing infrastructure, by controlling the distribution of advertisement and subscription requests. Policy rule definitions are fine-grained: rules are evaluated within the database infrastructure, and thus have access to a rich representation of state. The middleware enforcement of policy ensures client compliance, while having fewer definition/enforcement points simplifies policy management.

IC was designed specifically to support an environment of federated administrative control. Those managing database infrastructure have a responsibility to protect stored information. IC provides the means for meeting data management obligations, by allowing broker-specific control over the content of, and circumstances for, information disclosure. In Ch. 8 we discuss IC with respect to healthcare.

It is important to note that IC describes only the *mechanism* for controlling dissemination in a pub/sub infrastructure. In this dissertation we do not attempt to prescribe policies with certain characteristics, as clearly requirements will differ depending on the domain. Instead, the goal of this work is to *enable* those responsible for information to define policies appropriate for the local environment.

6

Context and Conflict

Predicates define the situations in which IC rules apply. This chapter explores context in more detail. We describe how fluents provide a representation of state suitable for a data control model. We also outline how credentials are assigned to a user, and show that aspects of state can be managed by external systems (remote to a broker).

As context forms the basis for rule application, there will be situations where several rules apply at an enforcement point. This means that policy can conflict. We take the position that conflicts are best resolved by policy authors, as automated detection and resolution mechanisms can be dangerous—their effects can be unforeseen. This is especially so in an environment such as healthcare, where information is critical for a patient's well-being. Instead, we provide support for policy authors, through mechanisms to detect potential conflicts. This allows a conflict to be 'authored-out' of the policy set, or resolved through a run-time resolution strategy.

6.1 Representing State

The predicates of an IC rule define the circumstances—the context—in which the rule applies. Clearly, the richer the representation of state, the finer the granularity of control. Rule predicates take the form of SQL conditional clauses, which are evaluated in the database-broker space. As described, predicates have access to messaging information (events/requests), stored data, internal functions, external services and schemata.

Predicates are constrained by the database environment, in that they must be valid SQL conforming with database and messaging schemata. In an environment where clients and brokers interact, accessing state through detailed queries (`select` statements) can, in the face of complex policy, be cumbersome and prone to error. It is advantageous, particularly in a distributed environment, to provide clients and policy authors with simple representations of state. As described in §5.2.2, global event type definitions enable the

routing of events throughout the broker network.[1] For routing to occur, each broker must be able to evaluate the subscription filters pertaining to an event type. This implies that each broker requires similar schemata, at least regarding the state accessible by filter predicates. Clauses are statically checked for conformance as part of the subscription propagation process.

In essence, information sharing requires shared schemata. As with event types, there are advantages to having shared definitions of state. Standardising methods for accessing context allows each broker to maintain their own internal data representation (e.g. table structures), which are used to derive a globally recognised and accessible representation of state.[2] Fig. 6.1 shows that standardised representations can provide a more intuitive interface for referencing context, without (from the client perspective) a detailed knowledge of underlying data structures. This simplifies both subscription filters and policy rule definitions. Further, the conditions activating rules become more visible, which facilitates rule comparison.

```
1(a) count(patient_id) > 0 FROM current_emergency
     WHERE patient_id = 2323232320
1(b) emergency(2323232320)

2(a) count(participant_id) > 0 FROM research_project_code_RP412
     WHERE participant_id = 2323232320
2(b) researchConsent(2323232320, 'RP412')
```

Figure 6.1: Two methods for evaluating state: (a) a query that directly references tables, and (b) a Boolean function providing an abstracted representation.

The next section describes how *fluents*, as defined by Kowalski and Sergot's *Event Calculus* [RM86], provide a useful mechanism for representing context in our control model. Fluents provide representations of state, which change in response to events. We use fluents to encapsulate parts of context (a predicate) under a named definition, perhaps representing a single aspect of state, a derived semantic or composite events. In a similar manner to event type definitions, fluents can range in scope from local to global.

6.1.1 Fluents

Event Calculus provides a mechanism for reasoning about state changes. It concerns the relationship between events (actions) and their effects on state [Sha99] with reference to time. A *fluent* in Event Calculus is defined as a reified, half-open interval in a given time domain [RM86]. Fluents are named, representing a particular state of affairs. A fluent is said to *hold* if the state of affairs it represents is true at a given timepoint.

Fig. 6.2 informally describes a subset of the basic predicates of the Simplified Event Calculus. These operate on the value of a fluent `F`, and an action (event) `a` at a particular

[1] Event type definitions can be locally defined, though the associated interactions will only be relevant to directly connected clients.

[2] Note the semantics associated with context may have different connotations depending on the environment: an emergency in a home is probably less severe than an emergency in an intensive care ward. However, similar issues exist with event type definitions. The meaning is clarified through other aspects, such as the event type, the publisher, the publisher's domain, and other events and contextual information. Alternatively, separate event types can be defined to deal with specific situations.

Function	Description
`Initiates(F,a,t)`	Fluent `F` holds after the action `a` at time `t`
`Terminates(F,a,t)`	Fluent `F` ceases to hold after the action `a` at time `t`
`HoldsAt(F,t)`	Whether the fluent `F` holds at time `t`

Figure 6.2: An informal description of Event Calculus Predicates.

time `t`. We borrow from Event Calculus to show how fluents provide a neat representation of state. Full details of the Calculus are described in [RM86] and [Sha99].

The `HoldsAt(F,t)` predicate determines whether a fluent holds at the given time. Informally, this is derived by determining whether the fluent was initiated at some point, and not subsequently terminated until `t`. This illustrates the declarative nature of fluents, where the status of a fluent is determined on demand, by inspecting previous occurrences. Fluents can be parameterised, which simplifies the management of named definitions.

Emergency Example

Fig. 6.3 illustrates the actions affecting the state of an emergency fluent concerning a particular patient. The value of the fluent `emergency` is initiated by three `sensor_reading` events with a heart rate of concern, or by the patient raising a `panic` event by pressing an emergency button. The fluent is terminated by an `emergency_clear` event. This scenario illustrates the use of composite events, where a number of event instances act cumulatively to alter fluent state. Further, this example shows how fluents can represent state transitions. Given that rules and subscription filters will reference fluent state, a change in the value of the `emergency` fluent may be of greater significance than the individual event instances.

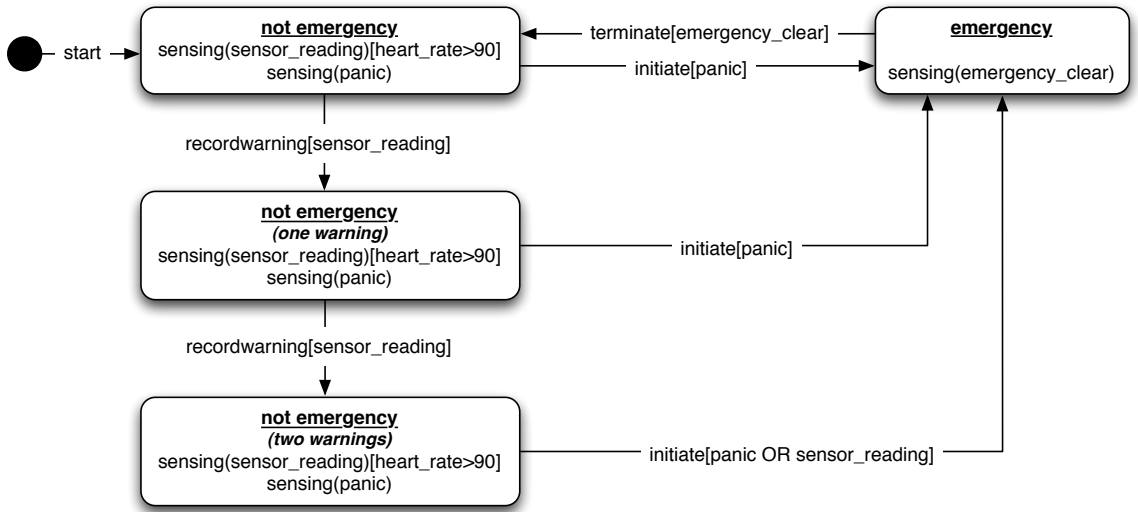


Figure 6.3: The events impacting on the state of an `emergency` fluent.

Fluents, Databases and IC

A fluent represents whether a particular aspect of state holds at a particular time, where actions (events) influence the value of a fluent. It follows that there is a simple mapping between IC and fluents: a predicate of an IC rule references a fluent. IC rules are context

sensitive. Rule definitions tend to concern a state of affairs rather than the occurrence of an event. Fluent definitions are useful as they describe how events impact on state.

Just as a view forms a query over data in tables in the relational database model, fluents can be thought of as queries over sequences of recorded events. Both provide a derived representation of some underlying state, and both can be *materialised* [CW91], which involves constructing a table (cache) to store the derived values. In IC, rules are evaluated in the current circumstances, determining whether a particular condition holds at the time of evaluation. Materialised views can be used to maintain the current value of the fluent, reducing search complexity by avoiding the query of event histories. Active rules (internal subscriptions) maintain the view by initiating or terminating fluents as appropriate to particular event(s). As historical events are persisted (Ch.10), the view can be reconstructed should initiating/terminating conditions change.

We use a function to provide an intuitive interface for accessing the state of a parameterised fluent, e.g. the function `emergency(patient-id)` returns whether the `emergency` fluent currently holds for the particular patient. Such functions obtain fluent state by querying the relevant materialised view or event histories, depending on the implementation.

6.1.2 Fluent Discussion

Fluents[3] provide a useful mechanism for managing context in a dynamic environment. A fluent represents a particular state of affairs, capturing contextual information, including state transitions and composite events, where a sequence of events impact on state. Fluents clarify the relationship between events and context.

Fluent operations are facilitated by the search and storage capabilities of database systems. As fluents are named, their definition highlights a particular aspect of context. Fluents can replace a complex query to allow for more natural predicate definitions. This simplifies the definition of rules and subscription filters, and clarifies the circumstances in which policy rules apply— the meaning of an appropriately named fluent is far more obvious to a user than a general SQL query (Fig. 6.1). In addition to abstraction and encapsulation, the events affecting state are made explicit by a fluent's initiation and termination definitions.

In a distributed environment, fluents bring about local control; each broker implements local functions to update and evaluate a fluent in a manner consistent with local procedure and its internal data structures. IC rules reference fluents, which are used to ascertain the current state of affairs (rule definitions/enforcement), and also to monitor conditions to trigger privilege re-evaluation should conditions change. The models naturally integrate: fluents are defined with reference to events, and thus require an event infrastructure. The integration of fluents into IC infrastructure acts as a precursor for future work into formalisation, enabling examination into the relationship between events, fluents and rules (§11.1). We describe the implementation specifics of fluents in Ch. 7.

[3]We borrow concepts from Event Calculus to illustrate a method for standardising the representation predicate definitions. The use of fluents provides a foundation for future work using formal logic to deal with issues of policy verification and obligation monitoring—see §11.1.

6.2 Credentials

In environments of sensitive information, access must be controlled. Credentials are a primary consideration in determining rights and information privileges, as they define the targets of the control mechanisms. They serve to assert a characteristic about an entity in a system, such as a role or qualification held. Some credentials might be persistent and exist for a period of time, such as an ID, or those representing a job-role or an employment contract; others are temporary, perhaps related to a session or a particular task (Fig. 6.4). We assume that each entity, including brokers, holds a credential asserting a unique identity.

Staff ID	*Persistent Credentials*	*Temporary Credentials*
NHS_412	NHS_412, Doctor, Paediatrician, Drug_Auditor, Employed_NHS_Provider_9132	Lead_Surgeon_Surgery3, Drug_Investigator_9132
NHS_1162	NHS_1162, Nurse, Head Nurse, A&E_Provider, Employed_NHS_Provider_9132	Ward_Manager_1152, Ward_Roster_Administrator
NHS_61112	NHS_61112, Tech_Support, NHS_Contractor_Organisation_3935	IT_Admin_Provider_9132, IT_Admin_Provider_5122

Figure 6.4: Example credential allocations.

While credentials represent context concerning a principal, they differ from general environmental state. In large-scale environments, some credentials tend towards being centrally managed, or at least shared across domains [PEB07b]. Credential allocations might include the employer or NHS qualifications (e.g. registered doctor/nurse). Changes in user credentials are usually less dynamic than other aspects of state: job roles change infrequently, workgroup allocations more frequently, but these are relatively static when compared to local environments that must represent and react to many dimensions of state. For instance, a patient might be in a ward for several days, while the nurse managing that ward is employed for years. As credentials assert something about a principal, the separation of credentials (who) from other system-related contextual conditions (when) allows directed policy, with visible targets.

Credentials are an important aspect of IC policy, as they define the targets of the rules. Credentials allow IC rules to apply to individuals, or particular classes of clients/brokers. As described, credential predicates combined with environmental predicates define the circumstances in which a rule governs privilege. Both types of predicates can be represented by fluents, which are defined locally but may involve calling external systems. IC is agnostic to the credential system that manages the mapping of users to privilege. All that is required is a mechanism for determining the credentials held by a user, and for notifying (raising events) in situations of credential change. Therefore, IC is compatible with a variety of access control methodologies. That said, robust and reactive access control mechanisms are required in large-scale, dynamic environments. This is especially so for healthcare, where access control is a priority.

6.2.1 Credentials in Healthcare

The importance of credentials is reflected in NHS infrastructure, where persistent credentials are used to define privilege. As introduced in §2.5, the NHS is proposing and has developed some services to assist in the privilege management. These are currently at various stages of design and implementation.

Electronic Staff Register (ESR) and N3 Network. The ESR is the definitive source for NHS staffing information. It is a centrally administered staff register, managing information including NHS (staff) IDs, domains of employment (providers), job allocations, contact details, and contract/leave dates. At present, ESR updates are sent as regular batches to local providers [McK]—evidence that staff-related privilege changes less often than aspects of environmental state. The N3 network uniquely identifies each user, assigning their NHS ID. Authentication occurs through a smartcard and PIN. These services provide the base for the assignment of privileges and credentials to the connected client.

NHS RBAC definitions. The NHS are advocating the use of RBAC to control access to information. Those holding a particular job role, perhaps qualified by an area of work, will be authorised to undertake a defined list of activities. Roles are mapped to users through a role profile, which are associated with identified clients (via N3). Role allocations are centrally defined by the NHS, with a bureaucratic process for any change in definition. Given that events encapsulate a particular semantic, activities can be readily associated with particular events.

Legitimate Relationships Service (LRS). The LRS records associations between carers and patients, and thus asserts some (medical) relationship between a client and patient data. Relationships are defined through workgroups, which function like patient-specific roles, existing for the duration of the treating relationship.

Such mechanisms on their own give a domain only limited control over privilege. For example, privilege allocations are pre-defined by central authorities; service providers may only add additional tasks to a (pre-allocated) role in their local environment [NHS08a]. This runs against the goal of greater freedom for health providers to manage their services [Dar08]. In large-scale healthcare infrastructure it is natural that (some) credential allocations are shared and/or globally defined. IC brings expressiveness, allowing service providers to address local concerns by defining policy that considers information from both centralised services *and* local representations of state. That is, with IC a domain can account for information from numerous sources, including central services such as those of the NPfIT, in a manner appropriate for local practice.[4] This gives domains flexible, responsive control over the dissemination of information for which they are responsible.[5]

6.3 Context Summary

IC rules are defined to apply in particular circumstances. The sensitivity of IC mechanisms depends on the representation of context, and the expressiveness and accessibility of state. In large-scale environments, such as a national healthcare service, it is natural that some aspects of context are centrally defined. However, care providers have their own local

[4] §7.3.3 describes the technical process of evaluating local and remote representations of state.

[5] Clearly domain policy must be in accordance with NHS directives; however, globally assigned privileges can diminish local control, by failing to account for the subtleties of the local environment.

processes, workflows and administrative policy. They require sufficient information, and the flexibility, to manage their local service.

Fluents succinctly represent state. They are useful for data dissemination control, providing an abstraction over queries and database tables, thus facilitating the sharing of contextual definitions. Further, they provide visibility of the interactions between events and context. Credentials assert characteristics about the users in the system; typically privileges are allocated with reference to credentials. These may be derived from external systems, perhaps through a dedicated credential management framework.

IC mechanisms are relatively unconstrained in their definitions of context. The use of fluents is an attempt to structure context, enabling access to local and remote representations state (see also §7.3.3). This is to assist policy definition and management without overly-constraining expressiveness.

6.4 Policy Conflict

As policy rules are context sensitive, a number may be applicable at a particular policy enforcement point. This means there will be situations in which *conflict* arises, in the sense that policies are incompatible. A common example of conflict concerns specialisation or specificity, e.g. where a rule authorising a request from a specific doctor should override the rule authorising doctors generally.

It is argued that application-level conflict resolution is often better addressed by careful policy (re)authoring, instead of complex automated resolution [Cha06]. Indeed, recent work concerning access policy involves warning authors of possible conflicts, through the static analysis of policy rules [SVBM08, SYA09]. Our approach is to warn policy authors of potential conflicts, which they can ignore or design around. This might involve redefining the policy-set, or specifying a conflict-resolution strategy for the conflicting rules. We do not attempt to resolve conflicts automatically, as doing so in a complex environment such as healthcare can be dangerous and complex. Instead we provide the tools for policy authors to detect and define policy conflicts and associated run-time resolutions.

6.4.1 Static Conflict Detection

Mechanisms are required to determine rules with the potential to conflict. A rule can be statically compared with others to detect situations where multiple rules apply. From this, a policy author can determine whether there exists a conflict between the rules.

Here we provide a detailed description of our conflict detection process, originally introduced in [SVBM08]. The recent work of Shu et al. [SYA09] describes formally a method for statically detecting conflicts in access policy based on rule reduction and binary search. The approaches bear similarities; however, we take a pragmatic approach dealing specifically with IC rules, which concern not only authorisation, but also impose conditions and transform.

Our process of conflict detection involves two steps. The first is to determine whether it is possible for rules to conflict. Following which, it is necessary to provide feedback as to the degree of the conflict: whether the conflict is due to an overlap in definition, or due to circumstance. Extra information can be used to further grade the degree of conflict, e.g.

whether two overlapping transformation rules produce the same event type. To detect conflict, a rule is compared with others of the same kind (request/link authorisation, transformation or imposed condition), defined for the same event type and enforcement point.

To ease explanation, while discussing conflict detection we state that policy rules *overlap* if they can be simultaneously activated by a particular set of conditions. This does not entail a conflict, as it might be wholly appropriate that several rules apply.

Possibility of Conflict

First is it necessary to determine whether it is possible for two rules to apply at a particular enforcement point. This is a *Boolean Satisfiability* problem (SAT) [Coo71], which involves determining whether there exists an allocation of state that activates both rules. The process of comparison involves converting the predicates of both rules into a *conjunctive normal form* (CNF) function. A CNF function consists of a conjunction of clauses, where each clause is a disjunction of literals [HR00]. A literal is a Boolean-valued variable, possibly negated. Here, a literal represents a predicate of a rule: an aspect of context. The idea is to determine whether the CNF function is *satisfiable*: that there is some allocation of the values (`true/false`) to the set of literals referenced by both rules which satisfies (makes true) the function. If the function is satisfiable, it means that there exists (at least one) contextual situation in which both rules can apply. Conversely, if the function is *unsatisfiable*, the rules cannot conflict. A solution may also be *undecidable*, if no definitive answer is found.

The search for satisfiability is a proven NP-Complete problem [Coo71]. Intuitively, showing a CNF function to be unsatisfiable may involve 2^n evaluations, where n represents the number of literals of the formula, or in this case, the number of different predicates (representations of state) referenced by the rules. There has been much work on reducing the search space; many approaches are based on the classic DPLL algorithm [DLL62] which uses backtracking and function simplification. There are implementations capable of finding solutions for formulas consisting of thousands of constrained variables,[6] and often practical (real-world) problems are solvable. In terms of IC rules, comparisons occur between pairs of rules, and are likely to consist only of a few predicates per definition, making the satisfiability search space comparatively small.

Fig. 6.5 presents some rule definitions and their CNF equivalents. We use shortened predicates for reasons of space. The credential predicates $dr(s)$ and $nrs(s)$ refer to whether the member of staff (s) holds a doctor or nurse role, respectively. The $duty(s, w)$ predicate determines whether a staff member is on duty in a particular ward, the $home(p)$ predicate describes whether a patient is at home, and the $em(p)$ predicate determines whether a patient is in an emergency.

Degree of Conflict

As the purpose of conflict detection is to assist policy authors, it is useful to grade the extent of the overlap. If it is possible for two rules to apply, as determined by the satisfiability search, it helps to consider whether the overlap is static, due to shared predicates, or whether the overlap is the result of dynamic circumstances.

[6] See `http://www.satcompetition.org/` for some examples.

Rule 1	Rule 2	CNF	SAT
dr(s) OR nrs(s)	dr(s)	$dr(s) \vee nrs(s)$	Y
dr(s) AND em(p)	not em(p)	$dr(s) \wedge em(p) \wedge \neg em(p)$	N
dr(s) AND em(p)	nrs(s) AND duty(s,w)	$dr(s) \wedge em(p) \wedge nrs(p) \wedge duty(s,w)$	Y
em(p)	NOT (home(p) OR em(p))	$em(p) \wedge \neg home(p) \wedge \neg em(p)$	N
(nrs(s) AND home(p)) OR dr(s)	NOT home(p) OR em(p)	$(nrs(s) \vee dr(s)) \wedge (home(p) \vee dr(s)) \wedge (\neg home(p) \vee em(p))$	Y

Figure 6.5: Rule definitions and their associated CNF representation.

The importance of this distinction is that statically overlapping policies are directed at similar targets; as opposed to some coincidental set of circumstances that cause both rules to apply. This may be indicative of a policy error. Determining a static overlap involves detecting like predicates in the definitions of the rules. This involves converting the activating predicates of a rule into *disjunctive normal form* (DNF), where each disjunct is a series of clauses consisting of literals (and negations) and conjuncts [HR00]. If rules share a common disjunction, they overlap in definition; otherwise, they overlap due to context. Fig. 6.6, shows some rules that statically overlap, sharing common disjuncts. When the common disjunct holds, both rules necessarily apply. Thus, dynamically overlapping rules have the *potential* to conflict.

Rule 1 *DNF equivalent*	**Rule 2** *DNF equivalent*	**Overlap**	**Common Disjunct**
dr(s) $dr(s)$	dr(s) OR nrs(s) $dr(s) \vee nrs(s)$	Static	$dr(s)$
home(p) $home(p)$	em(p) $em(p)$	Dynamic	
NOT (dr(s) AND nrs(s)) $\neg dr(s) \vee \neg nrs(s)$	NOT nrs(s) $\neg nrs(s)$	Static	$\neg nrs(s)$
dr(s) $dr(s)$	dr(s) AND home(p) $dr(s) \wedge home(p)$	Dynamic	
NOT (nrs(s) AND (home(p) OR em(p))) $(\neg home(p) \wedge \neg em(p)) \vee (\neg home(p) \wedge \neg nrs(s)) \vee (\neg nrs(s) \wedge \neg em(p)) \vee \neg nrs(s)$	NOT (home(p) OR nrs(s)) $\neg home(p) \wedge \neg nrs(s)$	Static	$\neg nrs(s) \wedge \neg home(p)$

Figure 6.6: A comparison of rule definition predicates.

Apart from overlapping definitions, other considerations might suggest a policy issue. In comparing transformation rules, it is useful to consider the output types of the transformation functions. If both functions produce events of the same type, two (semantically) similar events will be produced. This may indicate a policy error, especially for notification transformations, where the subscriber may be misled by receiving two instances of the same type. There may also be some other domain/implementation-specific considerations that can be incorporated into the detection process to rank a possible conflict, e.g. policies authored by inexperienced members of staff.

6.4.2 Selection Summary

The procedure for detecting conflicts for a rule is as follows:

- For each pair of rules of a similar kind, defined for the same event type and interaction point:
  - Determine whether it is possible for both rules to apply together:
    - Convert the predicates of both rules into a CNF function.
    - Check whether the function is satisfiable.
  - If satisfiable: Determine the degree of overlap:
    - Convert the predicates of each rule into DNF.
    - Search for any common disjuncts between the two rules.
    - Consider any other possible overlaps, e.g. function outputs.
- Present results to user, categorised by the type of overlap (static or dynamic), along with any other policy conflict ranking metrics.

The above procedure describes a basic method to assist policy authors and administrators in determining potential errors in the policy set. We present a general approach; as mentioned, extra constraints or other indicators of potential conflicts can be included as relevant to the local implementation/application environment. In practice, policy authoring processes should be strictly controlled, audited, sandboxed and reviewed to prevent polices from being added without considering potential interactions.

6.4.3 Resolution Strategies

The purpose of detecting potential conflicts is to provide policy administrators with sufficient information for maintaining a consistent policy set. We previously considered overlaps, where policies are simultaneously activated by a particular set of conditions. We now consider *conflict*, where policies are incompatible in some respect. The most obvious method of conflict resolution is to examine and redefine the policy set. There are strong arguments that this is preferable to a system-automated conflict approach [Cha06, SYA09]. As such, potential conflicts are detected and presented to the policy author to deal with any necessary resolution. It might be the case that certain classes of conflicts do not warrant re-definition, for example if the conflicts occur infrequently, or only in particular situations. Therefore, we allow the definition of constraints between rules, which instruct the system how to apply the policies at runtime. The runtime resolution of rules allows for exceptions, in cases where one policy should apply before or instead of another. IC resolutions (constraints) are described separately, specifically for the particular rules involved in the conflict.[7] This provides certainty and visibility as to how the rules are combined.

Our model provides the following mechanisms to define and resolve conflicts at an interaction point:

Ordering

This allows definition of the order in which the rules are applied/evaluated. This is particularly relevant for transformation functions, which can perform operations that affect state, such as altering data or raising an event. There may be circumstances in which one policy must apply before another.

[7] Recall that a broker locally stores and enforces rules and resolutions, cf. distributed policy sets.

```
graph : initial directed graph
roots : set of all the root nodes for the graph
ordered : list of ordered policies (initially empty)

while roots.not_empty():
      node = roots.pop()
      ordered.append(node)
      for each child in node.children():
            child.remove_incoming_edge(node)
            if child.is_root_node():
                  roots.append(child)

if graph.has_edges():
      ERROR: cyclic graph
else:
      return ordered
```

Figure 6.7: Pseudocode for a Topological Sort.

To account for this, a rule pol_1 can be defined as preceding another pol_2. This means that in the situation where both policies apply, the function for pol_1 will be enforced before pol_2. Ordering constraints (oc) are defined outside of the rules themselves, such that $oc = (pol_1, pol_2)$. At an enforcement point, there may be some ordering constraints relevant to the applicable set of policies. This presents a partially ordered set, where policies and their ordering constraints are represented as a directed graph ($pol_1 \rightarrow pol_2$), '$\rightarrow$' representing an edge. Ordering is resolved through a *topological sort* [CLRS01], presented in Fig. 6.7. The algorithm guarantees correct ordering among those with ordering constraints; unconstrained policies may be enforced between those with constraints. The time complexity of the sort is linear to the number of policies plus the number of constraints. Ordering is not possible where the graph representing the constraints is cyclic. In this situation an event is raised (and logged) detailing the policies, context and conditions giving rise to the issue (Ch. 10). Fig. 6.8 illustrates an ordering example.

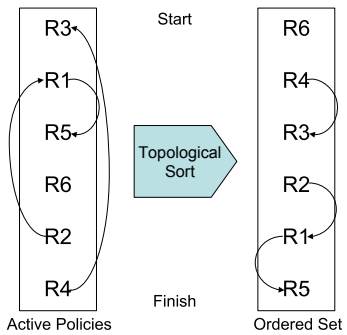


Figure 6.8: An illustration of policy ordering constraints, resolved by a topological sort.

Overrides

When policies conflict, a policy can be defined as overriding another. This prevents enforcement of the overridden policy, by removing it from the applicable set.

As with ordering constraints, an overriding constraint (or) is defined as $or = (pol_1, pol_2)$ where rule pol_1 overrides pol_2. Overrides apply directly to the policies for which they are

defined, i.e. overrides are not transitive. This is for flexibility, as there may be situations, such as annulment [AE07], where one policy constrains some policies, but is unconcerned with others. Additional override constraints can effect transitivity, useful for situations such as rule specialisation.

Override constraints are enforced through a similar mechanism to ordering policy. Overriding constraints can be represented as a directed graph, where $pol_1 \rightarrow pol_2$ means that pol_1 overrides pol_2. A topological sort on the graph returns the set of policies where an overridden policy appears after the overriding policy. Each policy in the set in examined in order; if the policy overrides another, then the overridden policy is removed from the applicable set. This process is shown in Fig. 6.9.

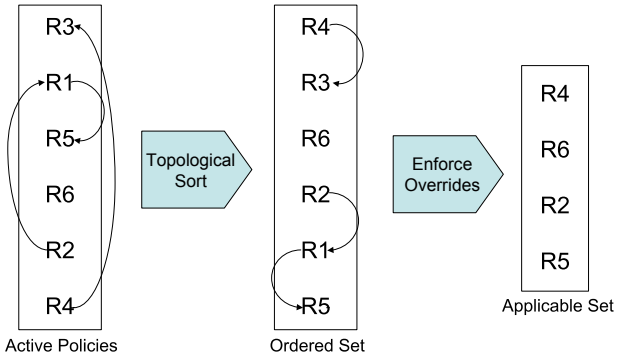


Figure 6.9: An illustration of the overriding process.

Incompatibility

A set of policies can be marked as incompatible. That is, there should *never* be a situation where they all apply. This is intended to act as a *safety-net* guarding against particular situations. This strategy prevents application of the rules, instead raising an event to inform of the incompatibility. This strategy is used implicitly to deliver warnings in cases where cycles are detected in the graphs of overriding and ordering constraints.

6.4.4 Runtime Application

The process of applying conflict resolutions is shown in Fig. 6.10. The first step involves determining the set of policies applicable at the enforcement point. Any overriding constraints are applied and the overridden policies are removed from the set. If the policy set is compatible, the policies are ordered in accordance with any ordering constraints. Policies are then applied/executed, in order, from the resulting set. The conflict resolution process is subject to audit, where the applicable, resolved and applied policies are recorded (§10.1.4).

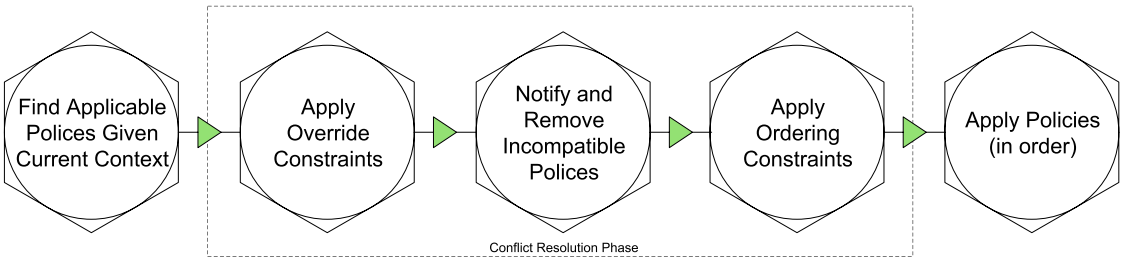


Figure 6.10: The process of policy application.

6.4.5 Detection and Resolution

Conflict resolution mechanisms impose runtime constraints on the application of policy rules. In this way, they not only dictate the appropriate action, but define the conflict itself. As such, it is important that these constraints are considered in the conflict detection process. The insertion of a particular rule might require an overriding or ordering constraint. The author should be able to discover what impact this may have—to impose a constraint, it is necessary to know whether the rule imposes constraints on other rules.

Conflict definitions can be statically analysed to determine cycles. The process involves selecting policies that might cause a cycle, where they are both parent and leaf node of a constraint; e.g. $p1 \rightarrow X$ and $Y \rightarrow p1$ where $p1$ is a policy and X and Y are some policies in a constraint definition. Such policies are considered root nodes, where cycles are detected by performing a depth first search over the tree of constraint definitions. If a cycle is detected, and all policy rules in the cycle can be activated (are satisfiable), this may be of concern. A statically-detected cycle is not necessarily an issue if the predicates cause the rules to be mutually exclusive: e.g. a patient cannot be in a ward (`inWard(ward_id,patient_id)`) and at home (`isHome(patient_id)`) at the same time.

6.4.6 Resolution Usage

Although runtime policy resolutions are an important aspect of IC, they are best used by administrators to handle exceptional situations. For example, a transformation rule t_3 can be defined to apply instead of transformations t_1 and t_2 when all are activated. However, the need for a resolution strategy might indicate an issue in the original policy set; perhaps t_1 and t_2 can be replaced, avoiding the need for t_3 and the resolution definitions.

Studies show that the majority of patients trust their practitioner to share information appropriately when required, without special requests [BRG00]. Assuming consent, organisations will have standard guidelines in which information is shared with particular entities. Runtime conflict resolution mechanisms enable the qualification of general rules with particular patient-issued restrictions. As described in Ch. 9, the definition of a consent preference does not necessarily imply the definition of an IC rule. Instead, some constraints can be encapsulated by context. For example, there may be a fluent representing consent to provide data for research purposes, where a change in a patient's consent preference involves updating the fluent rather than creating an IC rule. This shows that the use of resolution strategies can be minimised through system design.

6.5 Summary

This chapter describes constructs aimed at improving system manageability. As policy is defined and enforced within a database system, rules have access to a rich representation of state: a predicate may reference essentially anything accessible through the query engine. This is a key motivation for database-broker integration—it enables fine-grained control. However, such expressiveness comes at the cost of manageability, in that it can be difficult to define the queries to access the relevant aspects of state. Constraints on the representation of context help direct/clarify rule definitions and assist in rule comparisons.

Fluents provide a mechanism for balancing expressiveness and contextual control. Just as an event type represents a particular semantic regarding an occurrence, a fluent encap-

sulates a semantic regarding an aspect of state. Fluent definitions describe how events interact to alter their values. This makes visible the events that impact on state. Fluents preserve expressiveness, using the database infrastructure to derive state from queries (and functions) over data/event sequences, and providing the means for managing state transitions and composite events. We use fluents as an abstraction, moving queries from rules and filters into the middleware infrastructure, so that reference to a fluent replaces a (potentially) complex query predicate. However, this also impacts expressiveness and manageability, in the sense that policy authors (and subscribers) may only reference the conditional predicates defined for the particular broker. Another consideration is the performance impact of fluents, in that fluents require additional storage and introduce processing overheads in the evaluation and maintenance of state.

Given the dynamic nature of policy definitions, it is natural that policy conflicts occur. Conflict can be authored out of a policy set, through redefining policy rules. We provide for the run-time resolution of conflicting policy. Rules can be marked as incompatible, and ordering constraints can be defined. Rules may also be overridden, which is useful for qualification, e.g. where a specific (consent) policy overrides a general (domain) policy.

To detect conflicts, we employ a simple form of static analysis to identify the overlapping predicates in rule definitions. Using fluents to represent state facilitates this comparison, as fluents work to standardise the aspects of context referenced by rule predicates. Given that it is generally intractable or undecidable to detect overlaps in SQL queries [RI80], such constraints assist conflict detection processes. Here we consider overlap in terms of offline analysis to provide policy authors with feedback. This process may also indicate errors in the policy set, or in the sharing protocol itself. However, this process does not guarantee the detection of possible overlaps, instead it acts merely to support policy authors. As such, this approach is inappropriate to incorporate into the routing model itself (e.g. to implement coverage), as this requires real-time determinism. As this dissertation focuses on messaging infrastructure, we are concerned with the appropriate application of rules at an IC enforcement point. An area for future work is in system validation, where static and runtime analysis techniques can be used to reason about policy in relation to the semantics of a real healthcare system.

7

Integration into PostgreSQL-PS

This chapter describes the integration of IC mechanisms into PostgreSQL-PS brokers, through the addition of a data control layer. We detail the extensions necessary to integrate policy rule enforcement into database-messaging infrastructure, and how an IC implementation exploits existing database functionality, such as active rules and the query engine, to control information flows.

Given the focus of this thesis is on control, we discuss only those aspects of the PostgreSQL-PS implementation relevant to realising IC. See [Var09] for more information on database-broker integration and the PostgreSQL-PS implementation.

7.1 Data Control Layer

We implement IC as a data-control layer above a general pub/sub service to enable definition of policy rules for controlling information dissemination in a distributed pub/sub infrastructure. This layer is responsible for the storage, validation and enforcement of policy rules. Such operations are effected through interactions with the pub/sub layer.

Fig. 7.1 depicts the integration of pub/sub functionality into a database broker. The *user space* of a database concerns data, such as table specifications, rows and user-defined functions. The pub/sub layer resides between the system and user space of a broker. This is because some pub/sub functionality occurs in the *system space*, as communication specifics including serialisation and the (server-level) integration of event types are implemented in system code; while other aspects of the layer reside as data in the user-space, e.g. event type definitions and subscription filters are stored in *catalogues* (system tables).

The data control layer resides in user space, as policy rules are represented as data (rows) stored in tables, and enforced through (extended) active rules and user-defined functions. This facilitates the management of IC policy, as we can leverage from database functionality, including the query engine, executor, transactions, supported database languages and active rules.

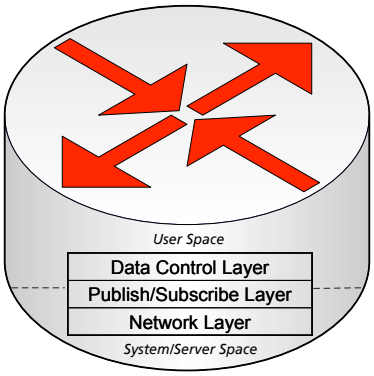


Figure 7.1: Layers of the messaging substrate.

7.2 Client Operations

Before describing the interactions between layers, we first introduce the actions of clients. Clients must be registered and authenticate with the database server in order to connect to the pub/sub service.[1] Clients can advertise (request to publish) and subscribe to (request to receive) event types, and publish event instances.

Clients represent requests and events in XML, which are parsed and converted into a tuple: a broker's internal data representation. Fig. 7.2 shows the schema of the request tuple. Advertisements and subscriptions are maintained in different catalogues. Apart from the fact that advertisement tuples do not have a `filter` attribute, the structure is the same for both.

REQUEST TUPLE									
req_id	eventtype	permatt_names[]	permatt_types[]	permatt_values[]	filter	impcond	authorised	feedback	userid

Figure 7.2: The schema of a request tuple.

Some attribute values are derived from the client's request, others, as underlined, are assigned by the system. The `authorised` and `feedback` attributes are used in request validation, where the former specifies whether the request is authorised, the latter allows for a message to be included with the acknowledgement sent to a client, e.g. to prompt the inclusion of a permission attribute, or to inform of a (non-hidden) imposed condition. The `impcond` attribute stores the filters imposed on this request. The values are determined as part of the request validation process (§7.6.2).

An event instance consists of a tuple specifying values for the attributes of the event type (see Fig. 4.1). Unlike a request, an event tuple is not populated with system-level information. Instead, metadata pertaining to the event instance, such as an event's (internal) identifier, is encapsulated in a wrapper surrounding the event instance.

7.2.1 Client-Specified Properties

PostgreSQL-PS requires that a client specify the `scope` of their subscription, advertisement or publication, which determines whether it is forwarded throughout the broker

[1] PostgreSQL allows specification of customised authentication modules (§4.3.4).

network (§4.3.1). IC shifts information flow controls into the middleware, thus rendering client-specified scoping redundant.

In PostgreSQL-PS, a publisher uses the `guaranteed` property of an event to specify whether the event instance is delivered reliably, exactly once in order, or whether a best effort approach is taken (§4.3.1). Again, as IC governs information flows, such a property is best defined by middleware policy. Given the importance of health information, and the requirement for audit, in this dissertation we consider only reliable delivery. While there may be healthcare situations where best-effort delivery suffices, e.g. in the transmission of video streams,[2] arguably this should be defined by policy pertaining to relevant event channel(s), rather than at the discretion of the publisher.

7.3 Context

In our implementation, predicates are evaluated through queries on local data or (internally defined) functions.[3] Evaluation occurs through the database query engine, in a manner similar to that of executing a SQL query returning a Boolean value: (`SELECT <filter predicates>`). This gives predicates access to a rich-representation of state, essentially anything accessible from the database system, subject to privilege (§4.3.4).

Evaluation takes place in the context of an event instance or request. Both event types and requests are defined as relational objects. Predicates are able to reference event content through `<eventtypename>.<attributename>` and request information through `req.<attributename>`.

The identifier of the connected user is also inserted into evaluation context. This is accessible through the use of the keywords `publisher`, which refers to the connected user sending the event/request, and `subscriber`, the recipient in the notification stage. During publication processing, `usernm`[4] represents the publisher, while at the delivery stage it represents the subscriber. The attribute `derived` records whether the event is the result of a transformation function.

PostgreSQL's planner is used to validate the conformance of predicates to database schemata, i.e. the validity of types and operators. This is done in the context of the connected client, to ensure that the issuer maintains sufficient privilege to access the predicates as defined by PostgreSQL's security model (§4.3.4).

```
SELECT NHSCred(usernm,'drug_auditor') AND underInvestigation(prescribe.prescriber_id);
SELECT NHSCred('NHS_777325','drug_auditor') AND underInvestigation('NHS_91253');
```

Figure 7.3: The query for evaluating a rule definition (a) and its associated evaluation context (b). The function evaluates the predicates, returning (`true`/`false`) if the statement holds.

[2]This dissertation considers more encapsulated events, rather than high-volume data streams.
[3]Functions can perform processing, and are able to access state from external systems and services.
[4]This is because `user` is a reserved word in PostgreSQL.

7.3.1 Permission Attributes

Permission attributes supplement the evaluation context with values specified by the client. The included permission attribute values vary per request, represented in the arrays of the request tuple. For reasons of convenience, attribute values are added to the evaluation context through placeholders: `att.<attributename>`. For example, the fluent `treats(usernm, att.patient_id)` will be evaluated as `treats('NHS_4234', '2323232320'::int8)`. Attributes values are cast to the appropriate type. Except for request de/serialisation, the pub/sub layer is unconcerned with permission attributes.

7.3.2 Representation: Fluents

The expressiveness of IC policy is governed by the representation of context. Fluents provide a neat abstraction of state useful for IC (§6.1.1). A fluent's value is accessed through a Boolean database function defined for the name of the fluent: `fluentname({params})`. That is, an emergency fluent, `emergency(2323232320)`, will return `true` if the patient is in an emergency situation, `false` otherwise. This accessor function is parameterised, giving an intuitive interface to accessing fluent state.

A *materialised view* is a table that acts like a cache, whose values are derived from (source) tables. Materialised views are generally used for performance reasons, as maintaining and querying a cache is in many circumstances faster than repeatedly querying the underlying sources (see §9.4). The value of a fluent is determined by a search that queries whether a fluent was initiated and not subsequently terminated. As IC rules consider state at the time of evaluation, materialised views can simplify the search process by maintaining the current value of a fluent. This avoids the need to search the entire event history for every request. Active rules (including internal subscriptions) are used to initiate and terminate the state of the fluent, through the respective functions `fluentname_init` and `fluentname_term` that maintain the fluent value in the materialised view.

Appx. B presents example source code for implementing the `emergency` fluent scenario of §6.1.1 as a materialised view. This shows how a fairly complex representation of state can be defined and maintained in a few database commands.

7.3.3 Monitored Conditions

Monitored conditions are used to trigger the re-evaluation of a request when its value changes. Our implementation requires that monitored conditions are defined as fluents. Monitoring occurs through the creation of active rules, which execute a function causing re-evaluation of the relevant request on a change in fluent state.

To ease the enforcement process, each fluent must define *monitor* and *demonitor* functions, which take the request ID as one of the parameters. These functions respectively setup and remove the mechanisms triggering re-evaluation of the request in a manner appropriate for the particular fluent. This standardises the interface to the monitoring functions by abstracting the specifics away from policy authors, and facilitates other associated (housekeeping) operations. The simplest monitoring functions might create an active rule, as shown in Fig. 7.4, to trigger re-evaluation on a change in a relevant fluent value. *Demonitor* functions exist to remove monitoring rules, e.g. after the closure of an event channel.

```
CREATE RULE "tie-request-to-fluent" AS
     ON UPDATE TO <fluent_allocation_table>
          WHERE OLD.param_1 = some_param_val
               DO select re_evaluate_request(<reqid>);
```

Figure 7.4: A generic rule to trigger request re-evaluation on a change in fluent value.

Credentials

Credentials assert a characteristic about a particular principal. They are used to define the targets of a rule. We represent all predicates, including credential allocations, as parameterised fluents. All credential fluents are monitored, to trigger request re-evaluation should credential allocations change. The event infrastructure can be used to update fluent values.

Fig. 7.5 illustrates the process of monitoring the credential allocations for local and remote representations of state for a particular request. The first step involves determining the values for the credential fluents for the requestor (`NHS_1234`). This entails querying a remote service (`NHS Creds`) for the centralised credential (`doctor`) (**1**), and local state for the domain credential (`WardManager`) (**2**). The request authorisation rule is satisfied as both fluents hold, thus the monitor function for each fluent is executed with the request ID (975) passed as a parameter (**3**). The monitor functions then set the active rules for monitoring state. The local credential is monitored through an active rule watching the `fluent_WardManager` table to detect a change in the credential allocation (**4**). The external credential is monitored through a subscription to the remote credential service (**5**). At this stage, all monitor functions are in place; the request is re-evaluated should conditions change (**6**).

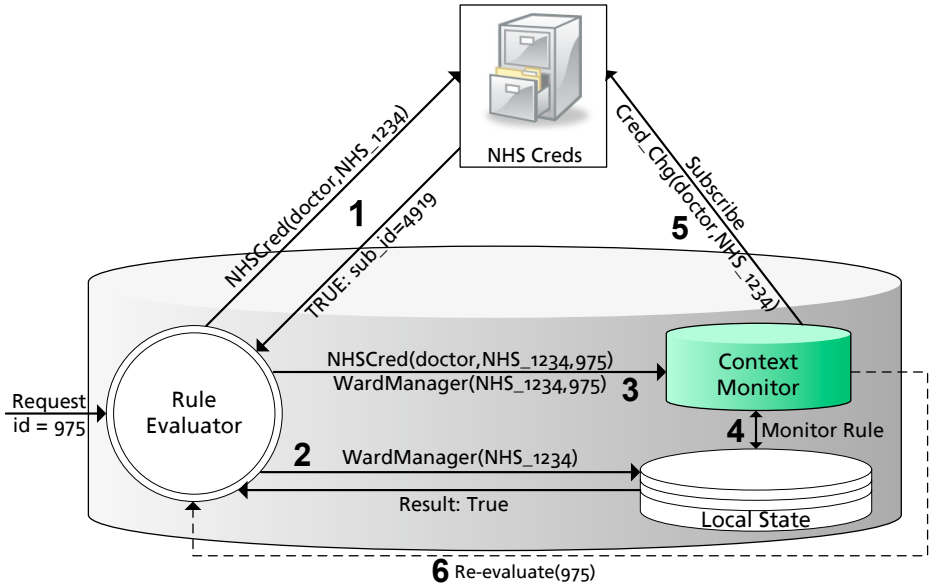


Figure 7.5: The process of monitoring conditions for a particular request.

7.4 Policy Definitions

This section concerns the definition of IC policy, describing its representation and the data structures by which it is stored.

7.4.1 Representation

We stated that data control policy resides in user space (§7.1). As policy is represented as data, policy could be defined directly through the database console (client interface). However, we force XML policy definitions, which are converted into the relevant data tuples. This is because representing policy in the same format as event instances allows rules to be shared using the event infrastructure, i.e. policy definitions can be defined as events. Further, the conversion process acts as a safety-net; rather than allowing direct manipulation of policy catalogues, policy is more easily validated (and manipulated) before being instantiated. The XML representation of policy follows the definitions described in Ch. 5. See Appx. C for examples.

7.4.2 Policy Storage

Data control policy exists in user space. Each type of policy has its own table(s), the schema of which corresponds to the rule's structure. Each rule is represented as data row(s). We define a policy conversion function that analyses policy, integrating policy definitions through SQL. Fig. 7.6 shows the relationships between the structures storing policy rules.

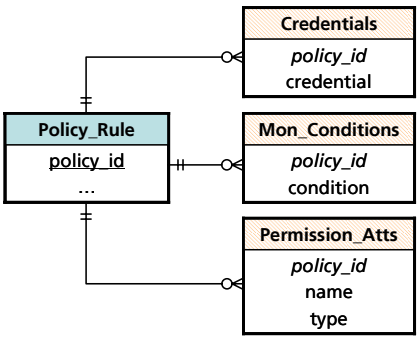


Figure 7.6: The relations for storing IC policy rules.

Each policy has a unique name that acts as a key for identifying the rule in the relations. Naming constraints are enforced by relational integrity. Policy tables are audited by triggers, providing information as to the policy author, time and details of the change (§10.1.6).

Predicates must be uniquely identifiable in order to be monitored. For this reason we restrict monitored conditions and credentials to be fluents. The monitored conditions and each credential predicate for a rule are stored in associative tables, as shown in Fig. 7.6. This allows the evaluation engine to easily identify the predicates that require monitoring.

The policy conversion function ensures that rule predicates conform to database schemata. Validation occurs through use of PostgreSQL's query planner. This level of validation con-

cerns the last step in the authoring process, working as a protective mechanism. In practice, authoring applications should provide more stringent validation before transmission to brokers.

Policy authoring processes must be tightly controlled: certain clients are restricted to specifying certain types of policies. Basic control can be allocated through the database's ACL pertaining to the tables defined for the policy types. However, if policy definitions are encapsulated in events, IC mechanisms can be used to control policy authoring processes. Rules authorising advertisements can define those who may produce policy definition events. Imposed conditions can filter such events, preventing certain policy definitions from being processed, e.g. restricting the event types for which a user can define policy. Transformations can also be useful, to ensure that policy is properly defined, e.g. to ensure that patient defined restrictions apply only to events pertaining to them. Policy arriving through links (remote brokers) might be subjected to more rigorous controls. Issues of policy sharing are discussed in Ch. 8.

7.4.3 Conflict Definition

In Ch. 6 we described how policy conflicts can be resolved at runtime, through the definition of conflicts and the appropriate resolutions. The entity-relationship diagram for the storage of conflict resolution definitions is presented in Fig. 7.7. Conflicts are defined with reference to policy names. Overriding and ordering constraints are stored in a table consisting of two policy identifiers and a `note` attribute that describes the conflict. For reasons of visibility, they are defined in separate tables. Incompatible policy definitions are stored in a table with the count of policies involved in the incompatibility, which are connected to policies by an associative table. A `join` considering the count determines whether an incompatibility rule applies.

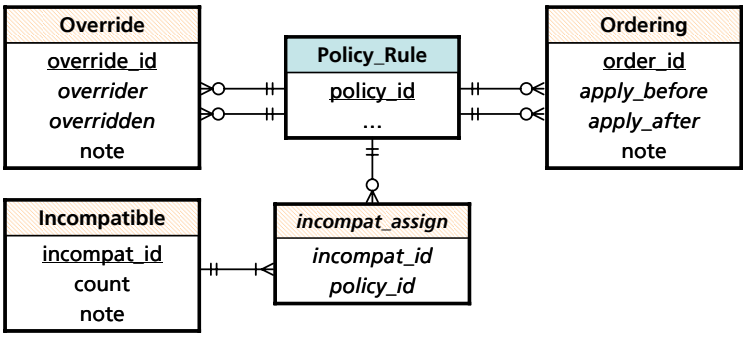


Figure 7.7: The relations for conflict definitions.

7.5 Layer Interactions: Hook Rules

The pub/sub and data control layers co-operate to provide IC functionality. The two layers interact through *hook rules*: active rules defined to apply at particular stages of the pub/sub process. The data control layer uses a hook rule as a *callback* mechanism, defining rules so that the pub/sub layer executes a function (in the data control layer) to effect some data control operation. Relevant data and state is passed to a hook function,

which the function (possibly) alters in some respect. Hook rules can be defined with conditions to refine the circumstances in which they apply. Fig. 7.8 provides an overview of hook rule functionality.

Hook rules are implemented in a similar manner to PostgreSQL triggers, in which values are 'injected' into the execution space of the function. This enables hook functions to access and alter the content of an event or request. All hook rules can be considered transformation functions:[5] a tuple is passed to a function in the data control layer, the result (output) of which is returned to the pub/sub layer on which processing continues.

Rule Type	Input/Output	Purpose
LINK VALIDATOR	Connection Details	Validates and authorises a link and establishes advertisement forwarding restrictions.
REQUEST VALIDATOR	Request Instance	Validates and processes the incoming request.
LINK ADVERTISEMENT PROCESSOR	Advertisement Details	Executed when an advertisement is received through a link. Establishes subscription forwarding restrictions.
REQUEST TRANSFORMATION	Request Instance	Modifies the request for delivery to a specific broker.
RESOLVE TRANSFORMATIONS	Applicable Rules	Resolves any conflicts between applicable transformations.
EVENT TRANSFORMATION	Event Instance	Executes the transformation function on the event instance.

Figure 7.8: Hook rule types and their associated description.

Hook rules are used to enable the data control layer to exercise control over certain points of the pub/sub process. As such, the enforcement points of the hook rules, as illustrated in Fig. 7.9, map to the policy enforcement points described in Ch. 5. Transformation hook rules are defined per *client*, applying to all relevant event channels (of the given type) for their connection.[6] See Appx. D for hook rule syntax and operational details.

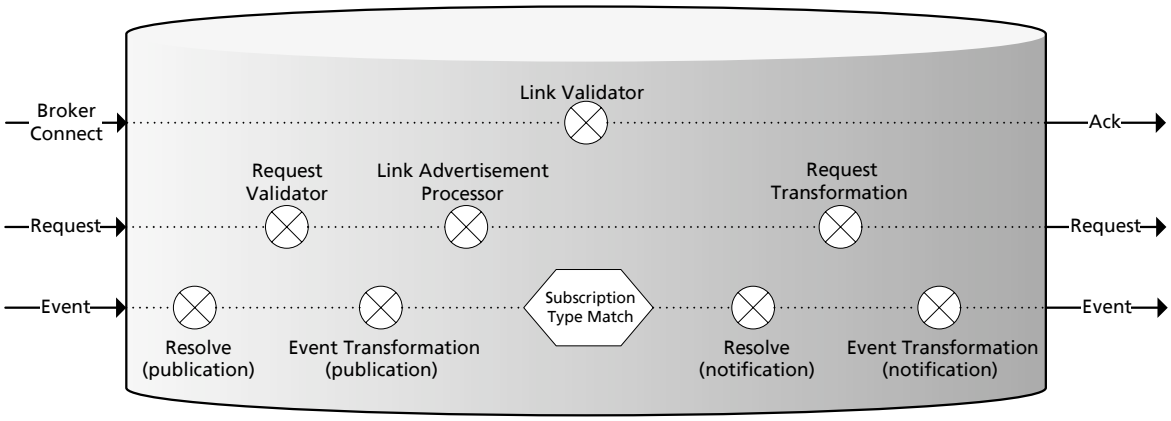


Figure 7.9: Hook rule enforcement points.

[5]For consistency, we continue to use the term *transformation* to refer to the policy mechanisms that alter event and request tuples, as described in Ch. 5.

[6]Remember that a client maintains a single connection with a broker, whereas an event channel is a logical channel defined within a connection concerning the unidirectional flow of a particular event type (§4.3.1).

7.6 Request Processing

A broker processing a request undertakes the following operations:

1. Deserialisation
2. Request validation and channel establishment
3. Propagation to other brokers

The following sections describe the processes of each in detail.

7.6.1 Deserialisation

Requests are issued in the form of an XML message. A broker that receives a request deserialises this message into the relevant request tuple: advertisement or subscription, the structure of which is presented in §7.2. Each successfully deserialised request is added to a processing queue and assigned a unique request ID.

7.6.2 Validation

Requests are validated at two levels: at the pub/sub level to ensure that a request conforms with the relevant schemata, and at the data control layer, which checks the request against IC rules.

The procedure of validating a request is previously described (§5.5.1). Fig. 7.10 presents a similar sequence diagram that details interactions between system components.

Publish/Subscribe Layer Validation

The initial validation phase is inherited from PostgreSQL-PS. A transaction is started and a request is removed from the request queue. If the request is a subscription, a filter plan is created. This ensures that the filter predicates are of the correct syntax, and that the user has sufficient privileges to evaluate the predicates.

Upon successful validation by the pub/sub layer, the *request validator* hook fires, executing the function to validate the request with respect to IC policy.

Data-control Layer Validation

The process is initiated by the execution of the validation function through the *request validator* hook rule. The function is passed the request tuple in order to validate, analyse and manipulate the request, and effect the appropriate restrictions. This validation function has the following responsibilities:

Authorise Channel Establishment Evaluates the predicates of request authorisation policy rules to determine, after conflict resolution, whether the request is authorised in the circumstances. If no rules are found to authorise the request, the request is denied; though the (negative) acknowledgement message can suggest the inclusion of particular permission attribute values. The decision is recorded in the `authorised` attribute of the request tuple.

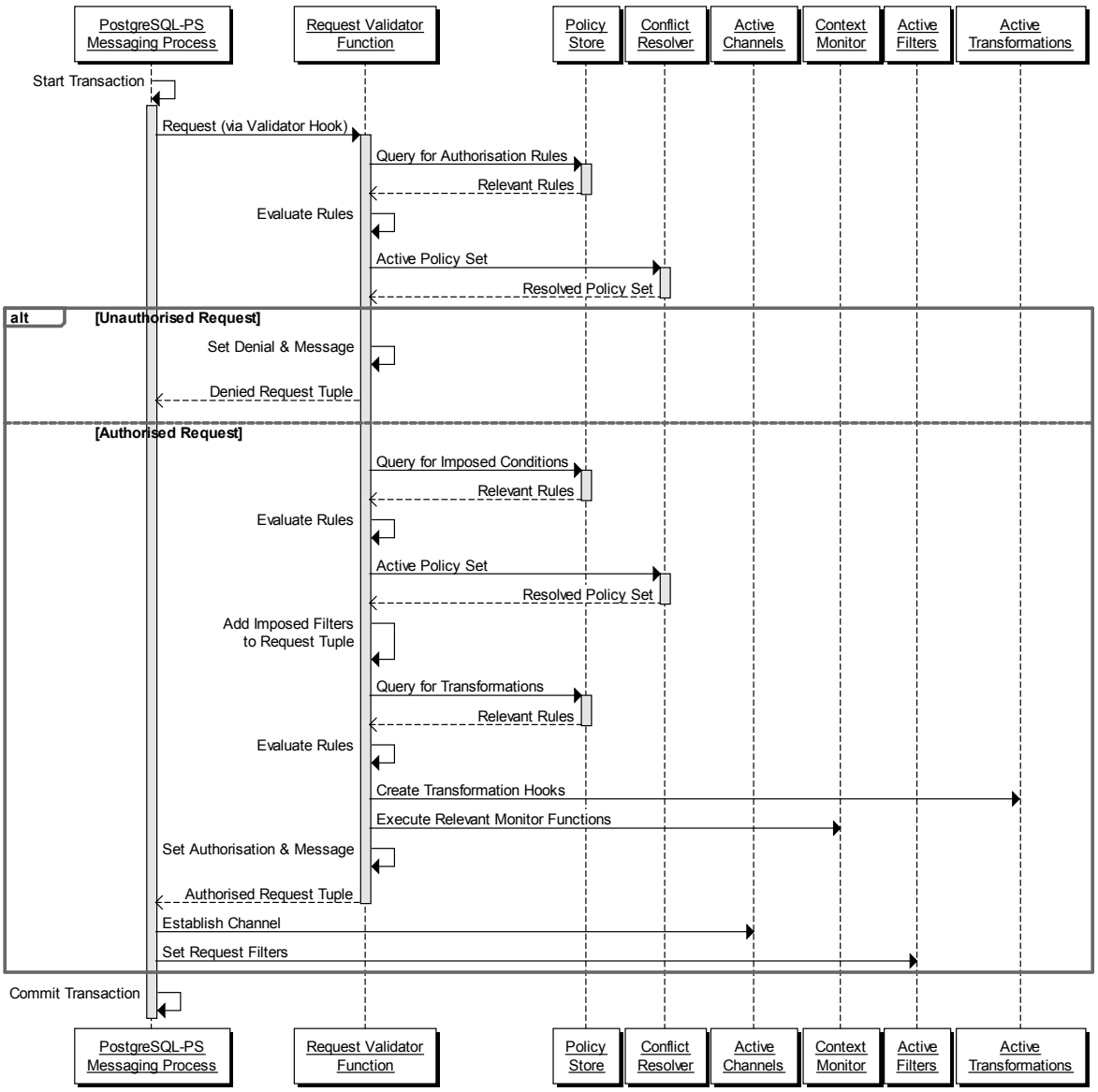


Figure 7.10: Sequence diagram for the request validation process.

Determine Imposed Conditions Evaluates the predicates of imposed policy rules to determine, after conflict resolution, the filters applicable to the event channel. If multiple rules apply, the restrictions are conjoined (ANDed) with each other, along with any permission attribute filters. This filter is stored in the `impconds` attribute of the request tuple.

Effect Transformation Policy Evaluates the predicates of event transformation rules to determine whether any transformations apply for the requested event type. An *event transformation* hook rule is created for each applicable rule. Conflict resolution does not occur at this stage, as transformation conflicts are resolved and applied in response to an event instance.

Implement Monitoring Calls the relevant `monitor` functions, passing the request ID, to trigger request re-evaluation when appropriate.

Provide Feedback Produces feedback relevant for the acknowledgement, e.g. to suggest retrying the request with particular permission attributes, or to inform that there are restrictions imposed on the request channel.

The validator function returns the request tuple, which the pub/sub layer uses to deny the event channel, or establish the event channel with the relevant filters, before sending the appropriate acknowledgement. As this process occurs within a single transaction, the channel and associated restrictions are constructed atomically.

7.7 Request Propagation

Validated requests are propagated to other brokers, through links, in accordance with request forwarding restrictions. This section describes the specifics of routing operations.

7.7.1 Routing Table Extensions

IC introduces rules to control request propagation. To effect this routing functionality, it is necessary to modify the routing catalogues of PostgreSQL-PS. The updated catalogues are presented in Fig. 7.11.

The main modifications concern the advertisement catalogue, which is split into two: *Advertisements_In* and *Advertisements_Out*. The *Advertisements_In* catalogue records information of advertisements received. The principal issuing the request is stored in the `from` attribute and the ID assigned to the request is recorded in `reqid`. The `islink` attribute records whether the advertisement was received through a link. This is used to ensure that subscriptions are only forwarded to other brokers. The `pubfilter` attribute records the filters imposed on published (incoming) event instances through the advertisement channel. Any filter on forwarding subscription requests (to brokers) is recorded in `subfilter`.

The *Advertisements_Out* catalogue records the advertisement requests propagated to each remote broker, whose ID is stored in the `to` field. The `issuedby` array is used to record the (locally) active channels that caused the request to be forwarded. An array is used to manage the propagation of an advertisement request, i.e. to avoid the transmission of duplicates. The `ackid` attribute records the ID assigned by the remote broker to the event (publication) channel. This is returned with the acknowledgement message to facilitate channel closure. The *Subscriptions* catalogue is also modified to record the request ID assigned to the event channel by the remote broker.

Advertisements_Out			
eventtype	to	issuedby[*reqid*]	ackid

Advertisements_In					
eventtype	from	islink	pubfilter	subfilter	reqid

Links			
connid	brokername	authorisingpol	advfilter

Subscriptions				
reqid	eventtype	filter	from	to[(*broker, ackid)*]

Figure 7.11: Structure of the IC routing catalogues.

The *Links* catalogue was extended to account for contextual link definitions. The attribute `authorisingpol` records the identifier of the authorisation policy allowing the connection. The inclusion of the `advfilter` allows filters to be defined on advertisement requests.

This filter is stored with link information because advertisements are not transmitted in response to a request.

7.7.2 Link Establishment

A broker attempts to create a link by connecting to the remote broker.[7] At this point, both brokers execute the *link validator* hook to authorise the establishment of the link.[8]

The process of validation involves determining the credentials of the remote broker, then searching the policy store for any applicable link authorisation rules. Any (defined) conflicts between the rules are resolved, where the first authorisation rule in the applicable set authorises the link. For safety, *monitor* functions watch the broker's credential allocations, though as brokers form part of a domain's infrastructure, we expect these to change infrequently. If no authorisation rules apply, the link is denied.

If the link is authorised, the relevant advertisement restrictions are established—as described in §7.7.3—and the function returns the ID of the link authorisation rule permitting the link. The pub/sub layer then sends an acknowledgement to the remote broker. Both brokers must authorise the link for it to be established. If authorised, brokers update their routing tables by forwarding, subject to any restrictions, the advertisement and subscription requests relating to the active event channels.

Conflict resolution is important to link establishment, as there are likely to exist general policies, e.g. those covering all registered surgeries, as well as broker-specific policies, e.g. those applying to specific surgeries. Resolution definitions enable the most relevant policy to apply.

7.7.3 Loading Forwarding Restrictions

In order to describe the process of enforcing forwarding restrictions, we first describe the mechanisms for loading/activating restriction policy.

Advertisement Restrictions

Advertisement restrictions are loaded by the *link validation* hook function, which fires when brokers interconnect. Advertisement restrictions for the remote broker are established on connection, as unlike subscriptions, advertisements are sent without a prior request. The policy store is queried to determine any relevant (after resolution) conditions imposed on forwarding advertisements to the remote broker. These are stored in the `advfilter` attribute of the *Links* catalogue. The `CanSubscribe(brokername,eventtype)` fluent is initialised and its status monitored, operating as an additional filter to ensure that advertisements are only forwarded to brokers with the propensity to subscribe to the event type. The relevant advertisement *request transformation* hook rules are then created for the link and particular event types.

Subscription Restrictions

A valid advertisement request establishes a publication channel, which is reflected in the *Advertisements_In* catalogue. If the advertisement is received through a link, i.e. is

[7] Connections for clients and links occur on different ports.

[8] Both brokers execute their respective versions of this hook as a) links create a bi-directional conduit for the establishment of event channels, and b) the hook function facilitates the establishment of advertisement forwarding restrictions.

forwarded through a broker, the *link advertisement processor* hook executes a function to establish any subscription forwarding restrictions relevant to the advertisement. The function creates a subscription *request transformation* hook for the link, if one is defined for the link and event type. It also generates a filter from any imposed conditions applicable to subscription requests for the broker and the event type. This filter is recorded in the `subfilter` property of the *Advertisements_In* catalogue, before the advertisement is forwarded.

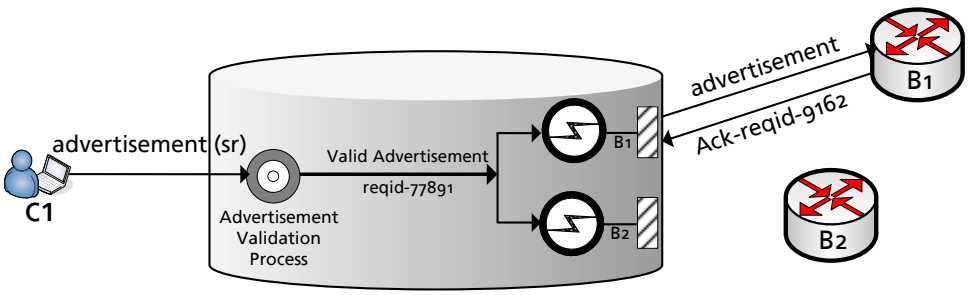
7.7.4 Request Forwarding

Only authorised requests, which result in the establishment of an event channel, are forwarded to remote brokers. We now describe the forwarding of the requests by type:

Advertisements

The process of forwarding a validated advertisement to a connected broker is as follows. First, the advertisement *request transformation* hook rule, if defined for the remote broker and event type, is executed. The resulting (or original) request is then validated against the advertisement filter for that link. If the filter is satisfied, the *Advertisement_Out* catalogue is examined to determine whether the advertisement should be propagated. If the remote broker has already received an advertisement for the request type, the request ID is appended to the `issuedby` array. This assists in request revocation. Otherwise, the advertisement request is transmitted to the remote broker. If the remote broker accepts the request (by establishing an event channel—§7.6), a row is added to the *Advertisement_Out* table, recording the local request ID in the `issuedby` array and the request ID assigned by the remote broker, via acknowledgement, in `ackid`. At this stage, advertisement processing is complete.

Fig. 7.12 provides an example illustrating the advertisement forwarding process, where the advertisement request issued by C1, after transformation, satisfies the forwarding filters specified for each broker. The advertisement is not forwarded to B2, as it has already received an advertisement for the event type `sr`; instead, the request ID is recorded in the `issuedby` attribute. As B1 has not previously received an advertisement for `sr`, due to an advertisement filter preventing B1 from receiving requests issued by C99, the advertisement request is forwarded B1. B1 accepts the advertisement, whose acknowledgement is recorded in the `ackid` field.



Advertisements_In					
eventtype	*from*	*islink*	*pubfilter*	*subfilter*	*reqid*
sr	c99	n			33211
sr	c1	n			77891

Advertisements_Out			
eventtype	*to*	*issuedby[reqid]*	*ackid*
sr	B2	[33211, 77891]	4551
sr	B1	[77891]	9162

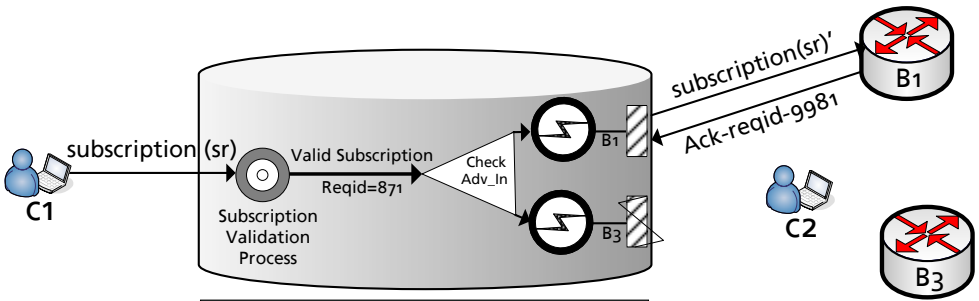
Figure 7.12: An illustration of advertisement forwarding.

Subscriptions

Once a subscription is validated its details are recorded in the *Subscription* catalogue. It must then be forwarded to brokers advertising the particular type, subject to any forwarding restrictions established as part of advertisement processing.[9]

The forwarding process begins by querying the *Advertisements_In* catalogue for remote brokers serving the event type. For each link, (a copy of) the subscription is subjected to the relevant subscription *request transformation* hook (if one is defined) before validation against the subscription filter (`subfilter`) imposed on the link. If the request satisfies the filter, it is forwarded to the remote broker. The remote broker may accept or deny a request (§7.6). If accepted, the request ID returned as part of the acknowledgement along with the remote broker ID is persisted in the `to` array of the *Subscription* catalogue. If a request is denied by the broker, the catalogue does not change.

Fig. 7.13 illustrates the process of subscription forwarding. Here the client C2 and the brokers B1 and B3 are advertising the event type `sr`. The subscription request, after being established in the local broker, is then considered for forwarding. The request, after the application of B1's subscription *request transformation* hook, matches the subscription filter (`subfilter`) defined for the link and thus is delivered to B1. However, the subscription, after the transformation, is *not* transmitted to B3. This is because B3 is not involved in the care of the patient, and thus request fails to satisfy the `subfilter` associated with B3's advertisement.



Advertisements_In				
event_type	*from*	*pubfilter*	*subfilter*	*reqid*
sr	B3		treats(att.patient_id,B3)	88123
sr	C2	sr.rate>5		901
sr	B1		treats(att.patient_id,B1)	71

Subscriptions				
reqid	*event_type*	*filter*	*from*	*to[(broker, ackid)]*
860	sr	ev.patient_id = NHS_1234 AND sr.rate>65	C42	[(B1, 9006) , (B3,422)]
871	sr	ev.patient_id = NHS_44912	C1	[(B1, 9981)]

Figure 7.13: An illustration of the subscription forwarding process.

Forwarding: Imposed Conditions

As part of the authorisation process, conditions may be imposed on an event channel to filter the flow of events. Although imposed filters are applied in conjunction with the filter specified in the request, the imposed conditions are not forwarded to remote brokers: they do not flow with the request. This is because imposed conditions are defined to control the flow of events between the requestor and the broker, with reference to *local* conditions.[10]

[9]Note that on receipt of a new advertisement, the requests pertaining to the active subscription channels must be forwarded to initialise the routing tables. The forwarding process is the same.

[10]The predicates of a request can be modified through a request transformation rule.

Revocation

On the closure of an event channel, either at the request of a client/broker or due to re-evaluation, the routing tables of local and remote brokers must be updated. If a subscription channel is closed, the relevant row in the *Subscription* catalogue is determined. Each remote broker is issued a revocation message for the request ID, as determined by the `to` array. The subscription is then removed from the catalogue.

The revocation of an advertisement involves removing its entry from the *Advertisement_In* catalogue. The *Advertisement_Out* catalogue is updated, where the request ID is removed from all `issuedby` rows in which it appears. If this renders the array empty for a particular row, the broker is sent a revocation for the appropriate request ID. The row is then removed from the *Advertisement_Out* catalogue.

7.7.5 Re-Evaluation

Request re-evaluation is triggered by some change in context. Active rules monitor the values of particular fluents, calling the appropriate re-evaluation function when certain conditions occur.

Link Re-Evaluation

Active rules are created by the link validation function to monitor the credentials of a remote broker. Should the credentials change, the *link validator* hook is re-executed, where any changes in restrictions are effected. If the link is no longer authorised, the requests for all active channels are revoked and the link closed.

Request Re-Evaluation

As part of establishing the event channel, *monitor* functions are used to associate a request with the value of a fluent. These functions trigger the request re-evaluation function when a value changes. The request ID is passed as a parameter to reconstruct and re-evaluate the original request tuple through the same validation function described in §7.6.2. This validates a request against authorisation rules, establishing any restrictions relevant to the request in the current context. Rules effecting restrictions and monitored conditions are removed where no longer relevant.

Re-Forwarding Requests

If a request tuple changes as part of re-evaluation, it may be necessary to update routing tables and inform remote brokers (§5.6.3). This process involves querying the catalogue for an existing entry representing the request for that link; if one exists, a revocation is issued. The updated request is then forwarded (if appropriate), and catalogues are updated according to the procedures previously described.

It is also necessary to forward requests pertaining to active event channels. For instance, all appropriate requests (per event type) for active subscriptions must be forwarded when a new advertisement is received or a new subscription authorisation policy is defined, and all appropriate advertisements must be forwarded to a newly connected broker. Forwarding involves transmitting the serialised request tuples for the active event channels in the manner previously described.

7.8 Event Processing

We have just described the details of request processing and the establishment of event channels. This section describes how policies are enforced for particular event instances, as they flow through event channels. Previously in Fig. 5.5 we presented the stages of policy enforcement as an event moves through a broker; here we detail this process. Again, enforcement is the same regardless of whether the publisher or recipient is a broker or general client.

The procedure for processing an event instance is as follows:

- Ensure the event complies with any publication filters
- Apply publication transformations
- Match resulting events against active subscriptions (by type)
- Apply notification transformations
- Deliver events if they match the subscription and restriction filters

7.8.1 Transactional Event Processing

We begin by describing the use of transactions and queues as part of the messaging process. An event on a queue signifies the completion of a particular stage of processing. Transactions are used to ensure integrity; the failure of a transaction allows recovery operations to take place. Queues and transactions are interrelated: events are added to queues within a transaction to ensure persistence; the addition of an event to a queue triggers processing, which involves the creation of new transactions; and, failure/recovery operations involve moving events between queues.

Section 4.3.3 describes how PostgreSQL-PS uses queues to manage event delivery. Processing begins when an event is received and added to the `in` queue. Events are matched against all active subscriptions, after which the event is placed on the `out` queue for delivery to each matched subscription. Once processing is complete, the event is removed from the `in` queue. Any failure in executing a function on an event instance for an internal subscription involves moving the event to the `exception` queue.

In PostgreSQL-PS, the input events are the same as the ones delivered. IC, however, introduces transformation rules, which raises additional considerations. Transformation functions produce new/modified events. This means that there is not necessarily a one-to-one mapping between the input event instance and the one delivered. In terms of reliability, merely adding an event that has already been internally validated to a queue has a relatively small scope for error, compared to IC, which involves executing user-defined functions for transformation and conflict resolution operations at various stages of the messaging process. As such, the queue management operations are changed to that described in Fig. 7.14.

To deal with this, we introduce a `match` queue, which operates as an intermediate queue before the application of notification hooks and filters. This queue stores a copy of the

```
For each event on queue1
     Begin Transaction:
          Remove event from queue1
          Determine applicable transformation rules
          Resolve conflicts between rules
          For each (resolved) transformation:
               Execute transformation function
               Place result on queue2
     End Transaction

     On Transaction End:
          If Error:
               Rollback transaction
               Copy event to exception_queue1
               Remove event from queue1
```

Figure 7.14: Pseudocode for queue management. `queue1` and `queue2` are the `in` and `match` queues for the publication stage, and the `match` and `out` queues for the delivery stage. The transformation rules are those relevant to the stage in the messaging process.

event instance, the output of a transformation hook,[11] for each active subscription to the event's type.

The `match` queue stores events for propagation to each connection, which as shown by Fig. 7.15, are added to the queue after the execution of each publication transformation. This queue acts as a marker denoting that publication processing is complete—that the event is ready for delivery to the particular subscriber. Once all applicable publication transformations are executed, the event is removed from the `in` queue and the transaction is committed.

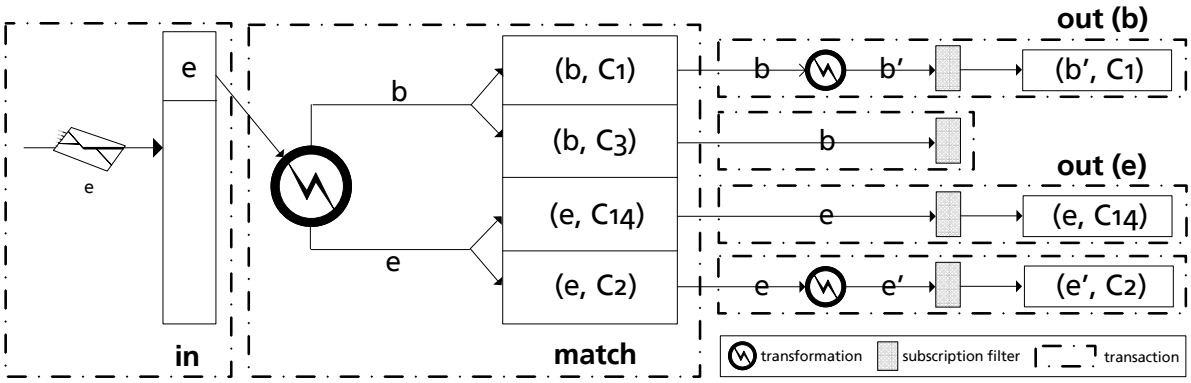


Figure 7.15: The transactions involved in processing a publication.

Each entry (event and connection-id) in the `match` queue is processed in its own transaction. This ensures that any failure in the execution of notification transformations and application of filters for a subscriber does not affect delivery to others. Events are moved

[11] This can be the original event instance, if no transformation function is defined or if the transformations do not consume the original (input) event.

to the `out` queue for delivery. After successful delivery, an event is removed from the queue. Note that PostgreSQL-PS simply ignores subscriptions where the filter fails. In IC, we force matching operations to occur within transactions, in case a filter fails or its evaluation impacts on state.[12] Further, each processing queue (for each type) is defined with a *dedicated* `exception` queue, to enable analysis and reprocessing of an event from the particular point of failure (see §10.1.3).

Our approach allows recovery operations to be focused on the particular event and phase of the messaging process. A failed publication can be retried by replaying the event from the stage in which processing failed *without* affecting delivery to other connections. Storing events on the `out` queue in separate transactions, without a `match` queue, complicates the recovery process if some subscribers receive the event before the failure occurs.

A transformation function changes the semantic of an event instance. It follows that in a channel, multiple transformations applicable to an event instance may, in some way, be related. This is evident from the need for conflict resolution mechanisms. In Ch. 9 we present a case study concerning prescriptions involving publication transformations, where a `drug_audit` event should not be created if the transformation creating the `prescription` event fails. Notification processes are subscriber-specific, concerning the delivery of an event instance to a particular subscriber. Our approach binds together related operations for a particular event instance into the same transaction, where failures are recorded in dedicated exception queues. The goal is to ensure that semantically related operations—those that affect an event instance in the same channel at the same enforcement point—apply and fail together. This renders failures visible, and aids recovery processes.

7.8.2 Publication Processing

The publication aspects of the messaging process concern the input (receipt) of an event by a broker, and its removal from the `in` queue. The process is summarised as follows:

- Receive and deserialise the message
- Validate it against the advertisement/filters
- Perform publication transformations
- Place events on the relevant `match` queue for notification processing

After the receipt and deserialisation of an event message into a tuple, the event instance is compared with the *Advertisement_In* catalogue. An event instance is only accepted for processing if it arrives on a channel for which there is a defined advertisement, and if the event satisfies any filter (`pubfilter`) imposed on that channel. If the event instance does not conform, it is denied;[13] otherwise the event is placed on the `in` queue. This functionality occurs within a single transaction.

[12] Transactions protect only local (database) state. Operations outside of the database, e.g. calls to external services, must be manually rolled-back (if required) in the case of exception. See [Var09] for details of distributed pub/sub transactions.

[13] A failure at this stage occurs before the event enters the `in` queue. Such failures are recorded by auditing processes (Ch. 10).

Event processing begins by starting a transaction and dequeueing an event from the `in` queue. The applicable transformation functions, loaded as active hook rules for the connection, are determined in the context of the current event instance. Conflicts are resolved through the *resolver* hook function, which returns the applicable set of transformations. Each function in the set is executed, in order, on a copy of the original event instance. A copy of the resulting event instance is placed on the `match` queue: a single entry for each connection maintaining one-or-more subscription channel(s) for the event type. The incoming (original) event is also added to the queue, if it is not consumed by any applied transformation rule. At this point the transaction is committed, completing the publication processing stage.

If there is some failure in the process, most likely due to an issue with a function or an unresolvable (cyclic/incompatible) policy set, the transaction fails. The transaction is rolled-back and the incoming event instance is moved from the `in` queue to the `exception_in` queue, along with a description of the error.

7.8.3 Notification Processing

The notification stage concerns the movement of an event from the `match` queue for delivery to the subscriber. The process is summarised as follows.

- Take an event from the `match` queue
- Perform the notification transformations
- Validate the results against the relevant subscription and restriction filters
- Serialise the events and deliver them to the subscriber

The process of event delivery is similar to that of the publication stage, but occurs in reverse. A transaction is started as the next event in the `match` queue is removed for processing and delivery. Each item in the queue is relevant for a specific connection. In the same manner as for publications, the relevant notification transformations are determined and applied. Transformation hooks are defined per recipient, so they are executed once regardless of the number of subscription channels for the connection. Each resulting instance is validated against the subscription filters and the imposed conditions on each event channel (of the relevant type) for the connection. These are applied in conjunction. If an instance satisfies the filter, it is moved from the `match` queue to the event type's `out` queue; there is at most one `out` queue entry for each event instance per connection. After comparison of all event instances, the transaction commits. The process repeats for subsequent items in the `match` queue.

Again if processing fails, the transaction is rolled-back and the event is removed from the `match` queue and placed in the `exception_match` queue for the type, along with details of the failure.

Events on the `out` queue are serialised and delivered to the subscribers. If the subscription is internal, the subscribing function is executed inside a transaction. An event is removed from the `out` queue when a receiver acknowledges receipt of the event, or the internal subscription function is successfully executed.

Type Transformations on Notification

If a notification transformation produces an event of a different type, it should be delivered subject to any relevant restrictions pertaining to the recipient and type. If delivery channels exist for the subscriber and the output type, the event is validated against those filters and placed on the corresponding `out` queue. If not, the transaction fails and the event is moved into the `exception_match` queue. This situation highlights the issues concerning type transformations on notification. If no delivery channel exists, it may be possible to establish one on demand through analysing the authorisation rules. This would involve determining whether any request authorisation rule for the new type can be satisfied given the information of the current request. If so, the channel could be temporarily established, subject to any imposed conditions relevant to the channel. However, as we do not advocate the use of type transformations at the notification stage (Appx. A), this was not explored in our implementation.

7.9 Summary

IC is integrated as a layer of control above an integrated database-pub/sub broker. This layer exploits and extends database functionality to implement broker-level control of information flows. Database languages, functions and active rules all facilitate the representation and monitoring of context. Hook rules are active rules that are sensitive to message processing operations, enabling the installation of controls at specific parts of a pub/sub service. These are used by the data control layer to effect IC functionality. This chapter demonstrates the feasibility and design decisions of realising an IC implementation.

8

Healthcare Integration Summary

The previous chapters describe the details of Interaction Control, a method for context-sensitive data disclosure control within pub/sub networks. This chapter revisits the information requirements of healthcare presented in Chapter 2 with reference to IC. We summarise its key features and reiterate the justifications previously presented for the model in supporting health processes, describing its place in an environment of federated control.

8.1 Healthcare Collaboration

Modern healthcare is becoming increasingly collaborative. As care moves to a preventative model, which involves the provision of remote and/or ongoing care, a greater number of care providers are becoming involved in primary and secondary health services. Each has specific information requirements relating to the service they provide, which can vary depending on the circumstances. Health infrastructure must support controlled information sharing to enable entities to perform their tasks as part of the care process.

Modern healthcare is highly data-driven. From a communication standpoint, both acute and preventative care models are reactive in that actions are taken in response to incidents. Patient monitoring is considered to be an integral part of future healthcare [EC08], where information must flow to various parties to alert of anomalous situations, inform of actions, update workflow processes, provide patient feedback, and for secondary uses such as auditing, billing and research. The technical infrastructure must enable entities to access the information required to perform their duties, so that they may react and respond to events as they occur, regardless of the severity of the health incident.

A push-based dissemination model is important for healthcare as it ensures that those providing care services are kept aware of the current situation. Patient monitoring in an assisted living situation highlights the need for an event-based infrastructure: to inform and alert of particular incidents as they occur, enabling an appropriate response.

IC is a model for controlling the dissemination of information. It is built on pub/sub, which is shown to be an efficient, scalable mechanism for event delivery [BV06, CRW01, EFGK03]. Consumers are able to specify their interests in receiving particular information. Clients are not burdened with addressing specifics, as delivery details are left to the middleware. This decoupling of information producers and consumers is suitable for healthcare, where many clients, applications and systems interact.

Database technology is already commonplace in the NHS; IC integrates into database systems. The coupling of messaging and storage substrates facilitates data replication. Information stores can subscribe to events as they occur in other environments. This enables information stores to maintain an up-to-date representation of state, which can be directly forwarded to local subscribers, and/or accessed through subsequent queries. An event might pertain to a patient, or represent some other aspect of context that may affect data control decisions, such as a staff member ending their shift. Such functionality is required to support an environment of federated data and policy.

Interoperability

This dissertation describes IC in the context of data protection. However, transformation rules also assist replication [SBS08, Fie04, CABB04], enabling interoperability between the databases of health service domains—including those proposed by the NPfIT. Transformations may involve the conversion of attribute values to different scales/measures, the addition or removal of data, and/or the production of another event in a format applicable for the remote system. Without transformation capabilities, the burden lies with producers to publish information in a manner appropriate for each possible recipient.

8.2 Data Sensitivity

Healthcare information is sensitive. Those who use and hold personal information are legally and ethically responsible for maintaining its confidentiality. Legislative acts and codes of practice acknowledge the need for sharing health information (§2.3), providing guidelines to assist in formulating sound sharing policy. The primary driver is patient consent, which may be explicit, implied, or in exceptional circumstances, irrelevant.

We have described the paradox between sharing personal health information and protecting its confidentiality. To balance this, information is best shared according to the need-to-know principle [NHS02]: concerning what information must be shared in the particular set of circumstances. Communication infrastructure supporting healthcare must allow those responsible for information to meet their data management obligations. IC targets this by allowing fine-grained control over the circumstances for data disclosure.

Typically, pub/sub concerns open communication in an anonymous environment. However, disclosure must be controlled when information is sensitive. The encryption and liberal distribution of information (at the broker-level e.g. [PEB07a]) for healthcare is inappropriate as data is perpetually sensitive.[1] Aside from issues of key-management, a compromised key, or a failure in an encryption algorithm at any time in the future can reveal sensitive information [BESP08, BES+09].

[1] We have mentioned that encryption plays a role at a lower-level: we assume transport layer security. Here, we are concerned with pub/sub security with regards to application(broker)-level routing.

IC addresses these concerns by building control mechanisms into a broker. The premise is that a broker should only release information as necessary in the particular circumstances. The goal is to enable those responsible to meet their data management obligations, through specification of broker-specific policy. IC brings four levels of control to a pub/sub service:

1. Request authorisation rules govern the establishment of typed event channels. These permit principals to deal with information of a particular semantic.
2. Imposed conditions act as a restriction, guarding certain event instances from being transmitted or processed.
3. Transformation rules control event content, perhaps enriching or degrading an event instance, or producing one of a different type. This enables precise control over the information contained in an event instance.
4. Link establishment and request forwarding restrictions allow control over distributed routing.

The NPfIT controls access to health information through RBAC, considering the relationship between an entity and a patient record. However, such an approach is rigid; there exists little scope for providers to customise access policy to suit the concerns of the local environment. This runs against the broader NHS goal of local control [DoH09b]. IC is context-aware, and thus provides greater flexibility by allowing granular control over the content of an event and its circumstances for transmission. Rule definitions can account for a number of concerns including consent preferences, patient state (e.g. emergency), location of staff and patients, client credentials, and so forth.

The middleware enforcement of IC rules ensures client adherence to policy. This reduces the potential for data loss due to error or negligence, as there are fewer policy definition and enforcement points. Clients are not burdened with the policy specifics concerning each patient, domain and recipient. Without IC, subscriptions must be partitioned along the lines of different access control rights. This requires the producer to publish multiple events pertaining to an incident, with varying levels of visibility. Apart from issues of practicality, information producers must be trusted to be aware of, maintain, and keep confidential the disclosure policy of domains and the consent preferences of patients. IC rules are context sensitive. If disclosure policy is enforced by clients, each producer requires knowledge of all interactions between context and event visibility, and must have the ability to access all relevant contextual information. Clearly events cannot be controlled with data outside the publisher's view.

8.3 Broker Interactions

Brokers interconnect to form a distributed broker network, forwarding advertisement/subscription requests to brokers, and event instances to brokers and other clients. IC introduces data disclosure policy into a pub/sub broker. Decisions by a broker to disclose information are based on local policy and accessible state. In this way, each broker maintains local control over its information, enforcing policy as data moves to/from directly-connected principals.

The difference between brokers and general clients is that brokers not only consume, but also forward requests and events. From a policy enforcement perspective, a broker is characterised as itself producing the forwarded request or publication. Policy is enforced on requests or events received through a link, subject to the rules applicable to the remote broker in the circumstances. In this way, clients and brokers are subject to similar enforcement processes.

8.4 Domains and Responsibility

We have defined a domain as a unit, governed by independent administrative policy, that provides particular services. In line with NHS notions of local control [DoH09b], each domain operates with a degree of autonomy. As part of the care process, a domain collects, stores and forwards information relevant to the service it provides. Each domain must appropriately share information with principals in the local environment, and with (principals from) other domains. It follows that a domain is responsible for protecting personal information. With local responsibility comes accountability: health domains are accountable, as they are subjected to auditing and monitoring procedures [DoH09c].

To meet this responsibility, health domains should specify an information sharing protocol defining the circumstances in which information is shared (§2.3.2).[2] This policy should be appropriate to the local environment, and should consider legal/ethical responsibilities, business practices, clinical processes, service contracts with other domains, NHS directives, and of course, consent. Any general protocol must be qualified by any patient-specific requests (consent).

8.4.1 Trust

Entities in the health service trust others to act appropriately with the information they receive. However, this trust is not implicit nor absolute. Taking a *pass-the-buck* approach, a domain meets its data responsibilities by passing information to a connected client in accordance with (sound) local policy. Given the overarching legal requirements for confidentiality, at this point the recipient becomes responsible for the information. That is, the responsibility for data passes with its transmission.

Generally pub/sub security models consider trust with respect to the broker network. IC assumes *no* broker-level trust, even between intra-domain brokers; instead, brokers enforce local policy against all connections. Here, trust exists at a higher level. We assume that a domain has control over, and therefore trusts, its (own) technical infrastructure. As such, a domain implements policy to ensure that information leaves its trusted broker network,[3] transferred to principals in local or remote domains only when appropriate—in line with the domain's data management obligations. In this way, the policy of a domain reflects its level of trust with other entities in the system.

[2] We do not imply that a domain and its clients (employees) are completely autonomous: employees are involved in the administration of a domain, just as employees are bound to act in accordance with domain practices.

[3] Trusted in the sense that infrastructure is under their direct control.

8.4.2 Realising Domain Policy

A domain implements IC policy in its broker network to realise its information sharing protocol. Policy need not, and often should not, be uniformly distributed amongst brokers within a domain. Instead, policy is defined to control the flow of information as appropriate to the topology of the network, controlling disclosure within and between domains. In addition, domain policy can also direct information flows to deal with system-level concerns, such as data replication and load balancing.

8.5 Federation

A federated healthcare environment consists of numerous domains of local control, who manage their own practices and procedure. That is, service providers define policy as relevant to their local operation. There is much support for federated healthcare environments, where domains hold local data, sharing it only when necessary, accounting for consent and legislation [HMF04, ABD+09, LKHT09]. This is in line with the explicit NHS goal of affording health providers a greater degree of freedom to manage their services [Dar08, DoH09b]. The arguments for federation pertain to the degree of data aggregation and issues of responsibility and accountability, i.e. who controls the data and associated policy, and the independence of the care provider. From a technical perspective, federated domains with cross-domain interaction form the natural architecture of the health space [Moo01], bringing *scalability* and manageability through limiting policy scope to that of the local environment.

Domains interconnect through links between local and external brokers. Domains formulate policies to enable sharing with others, taking account of particular information as previously described. A domain's policy is enforced by the brokers under its control.

As IC mechanisms are enforced at the broker-level, information flows can be controlled regardless of the size of the domain. That is, IC is agnostic to a domain's granularity, and thus can regulate information flows in both highly federated or centralised environments. Even in an extreme case where all information is stored in, and communicated through, a centrally managed *Information Service*, an internal broker-network is necessary for reasons of scalability. Fine-grained control mechanisms are still required to 1) control the propagation of information throughout the internal network, and 2) manage disclosure to entities with varying levels of privilege.

8.5.1 Point-to-Point Control

A key motivation for the use of a pub/sub middleware is that clients need-not have knowledge of disclosure policy, nor that of every information source/sink. This is important in healthcare—a large-scale environment of federated policy in which many thousands of systems operate [Bre05].

IC essentially overlays a point-to-point security model over a distributed pub/sub service. This supports information dissemination, in line with the overarching responsibility for protecting information, as disclosure controls are granular, enforced at each connection.

There is no metadata, such as the credentials of the originating requestor and their grounding domain, passed between brokers with forwarded requests/events. This is because a

broker is not qualified to make decisions regarding information flows pertaining to other brokers, especially if the broker is grounded in another domain. For a broker to make an informed decision concerning a request, it requires not only information of the requestor and the local processes, context and policies of its hosting broker, but also that of each broker along the dissemination path. Clearly, such decisions cannot be made at the broker-level without centralised policy definitions and a complete sharing of state. This would bring issues of policy management and scalability, especially when considering national-level services. It is more practicable for a domain to manage disclosure policy concerning only its direct connections.

Enforcing policy only at the publisher hosting broker in the worst case causes a separate version of an event to be routed for each subscription. This muddles notions of responsibility and accountability: should an intermediate broker merely forward events, thereby diminishing local control, or should it apply its own disclosure policy on top of that already applied by the publishing broker? Such an approach may not account for privacy aspects concerning brokers along the routing path.[4] Further, this runs against the general pub/sub notion of routing a single event instance as far as possible.

The local enforcement of policy can in some circumstances lead to a *Chinese-Whispers* type effect, where the content of an event or request changes as it moves through the network. This is a natural side effect of local control, existing at both the broker and domain level.

Each broker has policy controlling the information disclosed to connected principals, just as a domain maintains an information sharing protocol to manage its data responsibilities both within and across domains. A broker deals with particular information to support specific operations of a domain. Similarly, at a higher level, a domain holds information for a particular purpose, related to the care services it provides. As such, connections should be formed with (or close to) the relevant broker/domain for the particular information source,[5] which may not necessarily be the same location as the original publisher.

Such an infrastructure occurs naturally, given there is some reason for authorising each broker interconnection (link). That is, a connection exists between particular domains and brokers to share specific information. A local domain operates autonomously, and will maintain (relatively) persistent connections to the services most relevant to their local environment, forming ad-hoc connections when required. Such a structure allows scalability to the national level. Consider the example represented in Fig. 8.1. If a doctor requires information from the hospital, this information should come from the hospital itself. The *Pathology Register* requires information from firms that process biopsies—this is obtained directly from the laboratory.

Point-to-point security is a product of (higher-level) trust between care domains. The IC middleware is responsible for receiving a publication, transforming event content and delivering information to subscribers in accordance with information disclosure policy of each broker.

[4]These issues are similar to those raised in the vanilla-pub/sub implementation of the distributed prescription scenario as described in Ch. 9, except that rather than the publisher, here the broker hosting the publisher produces the multiple related event instances with varying levels of visibility.

[5]Note that this discussion concerns middleware, regarding broker interconnections. General clients communicate through the broker(s) of the domain(s) in which they operate. Of course, clients should not directly communicate, as to do so would circumvent the middleware and its protection mechanisms.

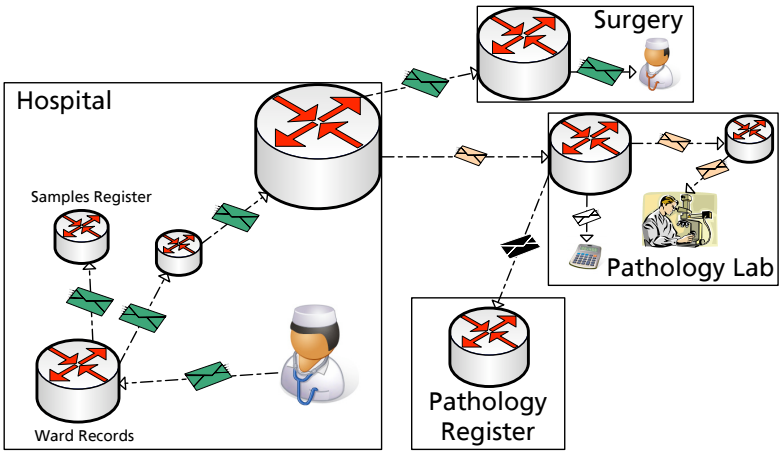


Figure 8.1: Illustration of point-to-point policy enforcement.

8.5.2 Central Services

Although federated healthcare entails local control over data and procedures, those advocating federated architectures acknowledge the need for centralised standards [HSH+09]. Some central services are also required to support national-level health processes. However, such services need not involve the complete centralisation of clinical data nor RBAC privilege allocations (as per §2.5). Instead, we introduce such services to assist with interoperability in a federated environment.

A health domain will frequently interact with other services. The number of domains will be far fewer than the number of clients operating within the system. As described, often the connections will be implicit, for example, existing through service contracts with a particular provider in a particular geographic locale. Directory services can assist in less-frequent interactions by directing connections to the appropriate domain(s). For instance, each SHA might define a lookup service to point providers from other regions towards the local provider(s) treating a patient. A domain can use this information to establish a link with a provider in another SHA, before forwarding subscription requests. This enables care for patients across the country.

Healthcare is *not* an environment of anonymous interaction. Identity plays an important role in defining the targets of an IC rule. As such, we assume that domains will register with some service to allow identification of a domain and its role in the health process. Similar processes already exist in the NHS, where organisations involved in care must register with the Care Quality Commission [DoH09b]. Clients must be registered to communicate through the N3 Network, enabling the unique identification of principals throughout the entire network. The Electronic Staff Record provides extra information on employees, such as their grounding domain. A similar approach can be used to identify brokers and their associated domains. Identity management need not be solely managed through centralised services, it is possible that a (verified and trusted) domain can assert some characteristics about a client for which they are responsible; for example, through certificate chains (§3.4.1). IC provides only the enforcement mechanism, and thus is agnostic to the method of credential management employed. By providing flexible mechanisms for representing context, we allow domains responsive dissemination control over the information for which they are responsible.

Our work also assumes globally defined event types and fluents, to allow communication between administrative domains. The NHS is already engaged in standardising definitions and terminology [NHS09c]. Also required is a mechanism to identify patients across domains. For this, the NHS is advocating the use of a global *NHS Number* [NHS08c].

8.5.3 Shared Policy

Although each domain maintains an information sharing protocol that is enforced by its brokers, some policies will be relevant to multiple domains. This might be due to a particular consent request, or perhaps some global NHS directive. Each domain, and thus each (appropriate) broker, needs to internalise such policy for local enforcement. The event infrastructure can be used to inform a domain of a particular restriction, where policy is encapsulated in an event, consumed and implemented by brokers.

Patient requests are typically restrictive, confined to particular sets of circumstances. A domain's information sharing protocol, implemented in local brokers, provides general protection that is qualified by any patient-specific restrictions. Such restrictions can be addressed through filters (imposed conditions) that prevent certain events from propagating. For example, a patient concerned about a particular staff member accessing their information can impose conditions in the domains in which the staff member operates. However, a data control restriction does not necessarily entail the definition of a new rule. The sharing protocol itself will often directly consider patient consent. Consent preferences can be generalised (e.g. opt-in/out), and thus can be encoded in (local/global) fluent state (see §9.5). For instance, the Legitimate Relationship Service could preclude the formation of an association between an employee and a patient, thereby encoding a 'negative' relationship.[6] Another example concerns the Summary Care Record, which is a central register of patient information updated by care providers, subject to patient consent. This consent can be represented in a fluent, which policy rules reference to allow or prevent data flow to the service.

Clearly, the best method of implementing a restriction is a question of design in the particular circumstances.

8.6 Summary

Healthcare information is sensitive, but must be shared to afford proper care. Those responsible for health information have a duty to protect it, sharing data on a need-to-know basis. IC brings to the pub/sub paradigm context-sensitive control over the type and content of the events transmitted. The approach is agnostic to the political concerns of health infrastructure: rules effect data control regardless of the size of a domain. IC seeks to provide the infrastructure to support federated healthcare, by facilitating data replication and the active delivery of information in a controlled manner. This allows each domain to meet their data management responsibilities, by giving fine-grained, context-aware, local control over data dissemination. Local responsibility brings accountability, mitigating the impacts of failure.

[6] The method for accessing such information must be carefully considered.

9

Case Studies

This chapter describes the application of IC mechanisms to healthcare scenarios. To emphasise the benefits of middleware enforcement, we present the IC rules necessary for restricting the data-flows of scenarios based on real-world requirements. We show that while the enforcement of IC mechanisms entails extra processing, in addition to ensuring compliance it can actually improve the overall performance of the messaging process. Results are presented to acknowledge implementation concerns, highlighting the need for careful consideration as to the manner in which data control policies are effected. We also describe how the middleware enforcement of policy facilitates policy evolution.

9.1 Prescribing Scenario

To test the expressiveness of our model in managing health information, we consulted nursing manuals, medical codes of practice and legislation, adapting the requirements to a home healthcare environment. In this scenario, we focus on data management aspects regarding the administration of prescriptions.

A key aspect of homecare is pain management. Care in the home tends to concern *palliative care*, which involves support of an ongoing, perhaps terminal condition; or that of *post-operative care*, which often involves pain and wound/infection management [McN00]. In these scenarios, typical medications include painkillers, or antibiotics. Generally, nurses are authorised to prescribe a licenced medicine for any condition within their competence [DoH07c]. In specific cases, including those of acute post-operative pain and palliative care, a nurse may prescribe certain *controlled drugs*, such as morphine derivatives. Controlled drugs have strict requirements, both in form and handling [RPS07], where designated entities are responsible for monitoring their supply [UK 06]. This monitoring focuses on the suppliers/prescribers—the audit process should, where possible, maintain patient confidentiality [UK 06].

9.1.1 Adaptation to Homecare

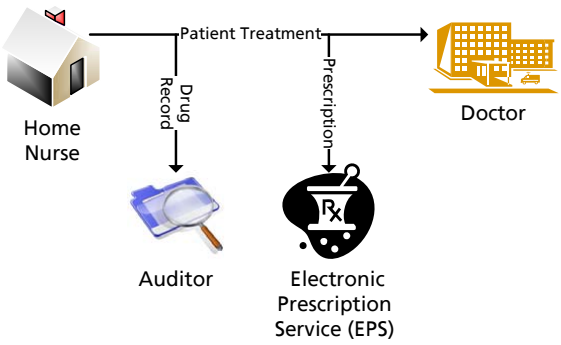


Figure 9.1: The providers interested in prescription information

A nurse caring for a patient at home might prescribe a drug. This involves recording information such as symptoms, patient complaints and observations, in addition to information regarding the drug and dosage. The act of prescription bears a significance that depends on the role of the provider. Firstly, the doctor responsible for a patient requires all information regarding the treatment process, including notes, observations and the reasons for the medication. This is relevant for the ongoing care of the patient. A pharmacist requires a valid prescription in order to legally dispense drugs. This includes information of the patient's personal details, the prescriber and drug information, but does *not* include information from the medical record, such as symptoms or observations [JBP05].[1] The auditor requires general information regarding the supply of controlled drugs; however, greater detail may be required for investigations, particularly when such matters pertain to public health and safety [UK 06].

9.1.2 Data Flows and Policy

Here we identify three domains that interact as part of the prescribing scenario:[2] 1) the *surgery* directly responsible for the care of the patient, 2) the *Electronic Prescription Service* (EPS) that is involved in the dispensing of medication, and 3) the *auditor*, responsible for monitoring the supply of controlled drugs for the particular surgery. The high level information flows between these entities are represented in Fig. 9.2.

There are two clients in this scenario: the homecare *nurse* and the patient's physician (*doctor*), both grounded (holding credentials) in the surgery domain. We model this scenario so that the surgery releases data to other principals, because it is the domain directly responsible for the care of the patient. The nurse, whilst caring for the patient at home, may decide to prescribe medication—perhaps a controlled drug if the situation warrants. This involves the nurse publishing (e.g. through a smartphone) a `prescribe` event, which includes information about the drug, dosage, symptoms, notes and observations. The doctor responsible for a patient may request to be notified, via a subscription, when drugs are prescribed for their patients, as this might indicate a condition of concern.

The body auditing the surgery must be informed of all controlled drugs issued as part of the care process. Rather than providing the auditor with complete details of the cir-

[1] There is pressure to allow pharmacists access to patient clinical data—see §9.2.

[2] In this example, we do not explicitly consider the home as we are concerned only with the actions of the nurse that are transmitted to the *surgery*.

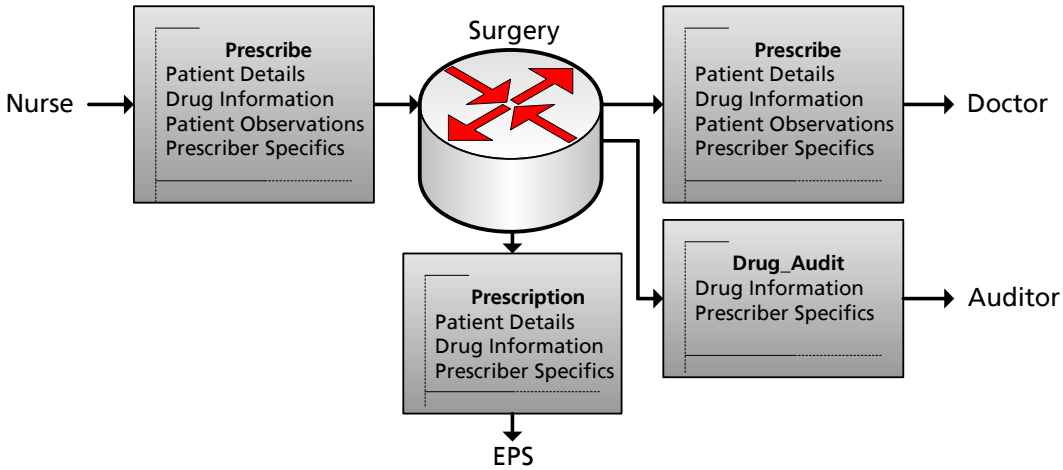


Figure 9.2: Data flows for `prescribe` events.

cumstances surrounding the prescription of a controlled drug, the auditor is notified by a `drug_audit` event that a controlled drug was issued. To protect patient confidentiality, the event does not include patient information. In exceptional cases, such as where the prescriber is under suspicion, the auditor may receive `prescribe` events, where extra information assists in the investigation. The surgery has a duty to protect the confidentiality of patient information, and thus must control disclosure, despite the fact the auditor is also bound to respect the confidentiality of any information received. The Controlled Drug regulations [UK 06] refer to the circumstances in which the auditor shares prescribing information with others. Our implementation goes further than the minimum requirements, by not releasing patient information to the auditor except where warranted, such as during the investigation of suspicious behaviour, and even then, in line with patient consent.[3] The EPS receives information concerning the allocation of a drug through a `prescription` event, which includes all information necessary to constitute a legal prescription.

9.1.3 Data Control Policies

There are two transformation rules defined for this scenario. The first concerns the conversion of a `prescribe` event into a `prescription`, where the `prescription` event contains general patient details, but does not include health specifics. The conversion process involves a transformation function which copies across the relevant details from the `prescribe` event, augmenting it with extra patient information, such as the patient's address and DOB, and the surgery details.[4] This transformation applies to all `prescribe` events, where the EPS subscribes directly to the `prescription` (output) event.

The other transformation function concerns the removal of sensitive information from controlled drug allocations. The `drug_audit` event is derived from the `prescribe` event, by filtering various fields. Fig. 9.3 shows the audit transformation rule, and the associated

[3]The appropriate procedures will vary depending on the processes of the auditor and local practice. Here we illustrate one such scenario of how they may be implemented. The approach presented is more protective than current practice, given that the legislation implies that the auditor has access to *all* sensitive information.

[4]Rather than presenting all attributes, Fig. 9.2 represents the event flows at a conceptual-level.

functions for both transformations are presented in Appx. E.[5] This transformation applies only when the drug prescribed is classified as controlled.

```
<transformation>
     <rule_name>createdrugaudit</rule_name>
     <event_type>prescribe</event_type>
     <output_event>drug_audit</output_event>
     <interaction_point>p</interaction_point>
     <consumable>f</consumable>
     <function>prescribe_to_audit</function>
     <conditions>controlledDrug(prescribe.drug_id)</conditions>
     <notes>Converts prescribe to drug_audit events</notes>
</transformation>
```

Figure 9.3: The prescribe/audit transformation rule.

Authorisation Rules & Restrictions

The authorisation rule of Fig. 9.4 allows a doctor to subscribe to `prescribe` events, but only for patients that they treat. The EPS may only subscribe to `prescription` events, while the auditor may subscribe to `drug_audit` events—these authorisation rule definitions are found in Appx. E.1.

```
<request_authorisation>
     <rule_name>drprescribe</rule_name>
     <event_type>prescribe</event_type>
     <request_type>s</request_type>
     <credentials>NHSCred(usernm, 'doctor')</credentials>
     <permission_attributes>patient_id:int8</permission_attributes>
     <mon_conditions>treatsPatient(usernm, att.patient_id)</mon_conditions>
     <notes>Allows a doctor to subscribe to prescribe events for patients that they treat.</notes>
</request_authorisation>
```

Figure 9.4: The authorisation rule for the doctor.

For reasons of public safety, we allow auditors to receive to `prescribe` events, only when the particular prescriber is under investigation. This is because an investigation is a serious matter, and as such it is in the interests of public safety to ensure that the particular prescriber is supplying all types of drugs correctly. This is, however, only done in line with patient consent, which might be obtained when providing care, or perhaps in reference to the individual investigation. We account for this condition through an imposed condition, as shown in Fig. 9.5.

9.1.4 Single Broker Implementation

Given this set of data control rules, we performed some trials concerning a single broker representing the surgery domain. The surgery domain is connected, via links, to the EPS and its local drug auditor. All tests described in this chapter, including the single and multi-broker scenarios, involved executing the brokers on hardware consisting of a Intel Core 2 Duo 2.4Ghz CPU with 2GiB of RAM. The clients were written in Python, distributed amongst a number of machines running on a different subnet from the brokers. Brokers were slightly modified for instrumentation purposes: system calls (`gettimeofday()`) and counters were added to respectively measure the time taken to

[5] The complete set of data control policies driving this scenario is presented in Appx. E.

```
<imposed_condition>
     <rule_name>auditorprescribeinvestigation</rule_name>
     <event_type>prescribe</event_type>
     <interaction_point>n</interaction_point>
     <credentials>NHSCred(usernm,'drug_auditor')</credentials>
     <restrictions>underInvestigation(prescribe.prescriber_id)
          AND givenAuditorConsent(prescribe.patient_id)</restrictions>
     <notes>Prescribe events can be delivered to the auditor when the prescriber
          is under investigation and consent has been given.</notes>
     <hidden>f</hidden>
</imposed_condition>
```

Figure 9.5: Auditor-specific subscription rules for the prescribe event.

perform various operations, and to record the bytes transmitted. We present the mean values over 10 trials.[6]

Our experiments consist of a surgery with 1,000 active patient records. Four doctors work at the surgery, who manage 250 patients each. Five nurses visit patients at home, raising `prescribe` events for the drugs they prescribe. Each doctor subscribes to `prescribe` events pertaining to the top 25 (10%) most critical patients they treat.[7]

We characterise the workload as each nurse publishing 1,000 messages, 200 of which concern patients to which doctors are subscribed.[8] Each `prescribe` event is transformed into a `prescription` for transfer to the EPS, and each controlled drug into a `drug_audit` event for the auditor. The auditor receives all `prescribe` events published by the single nurse that is under investigation.[9] Fig. 9.6 illustrates the environment.

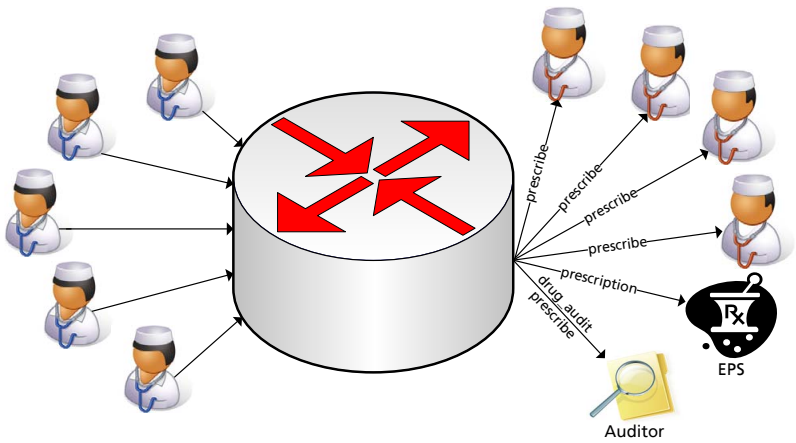


Figure 9.6: The single-broker prescription scenario.

[6]Unless otherwise specified, the error-bars represent the 95% confidence intervals for the mean.

[7]Clearly the doctors will generally be interested in prescriptions pertaining to other, less-critical, patients. However, all information is stored in the database-broker of the surgery.

[8]In practice prescriptions occur relatively infrequently, i.e. several per day. However, a large number of publications were used in this scenario to provide a *general* indication of performance, accounting for database optimisations such as write buffers and caching, and to ensure that nurses publish simultaneously to interleave processing.

[9]To reduce the number of variables, in this experiment we assume that patient consent has been obtained, where `GivenAuditorConsent(pid)` holds for all patient IDs. As this condition is imposed and thus evaluated on every instance, the imposed evaluation overhead is fixed.

9.1.5 Controlled Drug Implementation

Naturally, we expect transformations to introduce processing overheads. Fig. 9.7 presents the average time for complete processing of a `prescribe` event publication. The percentage of prescriptions for controlled drugs was varied, as each controlled drug requires more processing, to create and deliver `drug_audit` events. A 20% increase in the number of controlled drugs results in 1,000 more audit events produced and delivered.

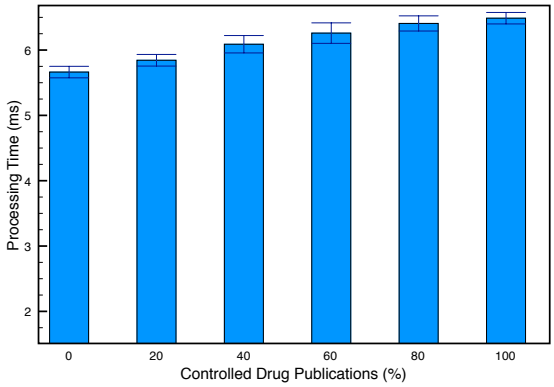


Figure 9.7: Processing time per `prescribe` publication.

The figure shows that although there are variations in the processing time at the various percentage points, overall the differences in processing times were relatively insignificant when compared with the total processing time per event.

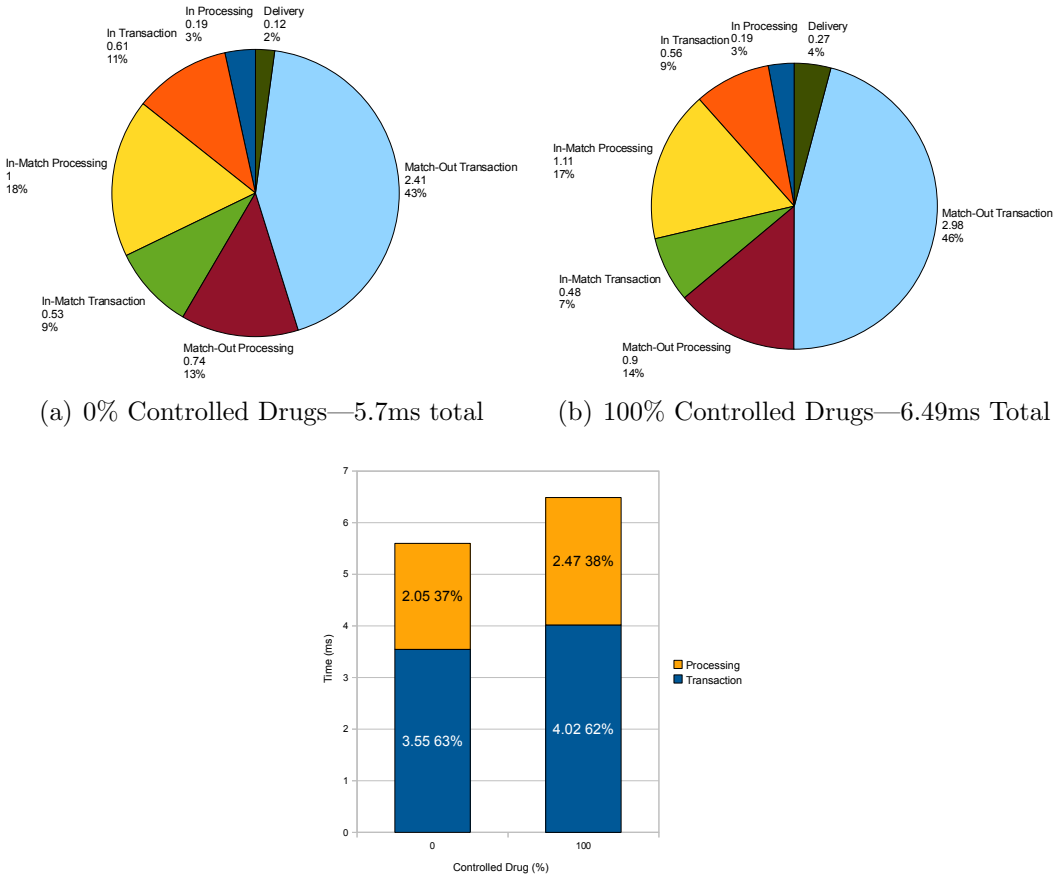
In §7.8.1 we described the transactional processing of events, involving the movements of events between processing queues. Transactions are necessary for the reliable processing of events; however, transactions also impose an overhead. The breakdown of message processing operations and the associated transaction times is presented in Fig. 9.8, for the two extreme cases—where no and all drugs are controlled. Figs. (a) and (b) show the stages in which the transactional (particularly commit) operations apply, where the operations as a proportion of processing time are relatively similar. As the creation of a `drug_audit` event entails more processing, Fig. 9.8(b) presents a greater processing time for each operation. Fig. 9.8(c) shows that transaction operations, in total, take close to two-thirds of event processing time.

In this example, the transactional overheads of message processing outweigh those imposed by the transformations. A workload in which all drugs are controlled results in a ∼14% increase in message processing time over the case where no drugs are controlled, but the number of messages delivered increases by ∼71%.[10]

9.1.6 Comparison to Vanilla Pub/Sub

To better gauge the message-processing overheads of IC, it is necessary to compare that to an implementation of a vanilla pub/sub model; that is, one *without* event processing (transformation) capabilities. Without IC rules, the publishers become the sole source of information, who must deal with confidentiality concerns by publishing a separate event for each level of visibility. This involves publishing two or three events for each

[10] 12,000 events are delivered in the workload with controlled drugs, 7,000 in the workload without.



(a) 0% Controlled Drugs—5.7ms total

(b) 100% Controlled Drugs—6.49ms Total

(c) Transaction Aggregation

Figure 9.8: Transactional overheads on event processing.

prescription: a `prescribe` event, relevant for the doctors/surgery, a `prescription` event as relevant for the EPS, and if the drug is controlled, a `drug_audit` event.

We implement the vanilla pub/sub scenario using the same database-broker infrastructure, to ensure that the information is reliably recorded, audited, processed and delivered. The difference is that the vanilla implementation lacks any IC rules. To enable a like comparison we assume that subscribers are honest, issuing filters in line with the restrictions otherwise imposed by the IC implementation.

Fig. 9.9 shows that the total processing time for the vanilla implementation involves a significantly greater overall processing time than the IC model for each workload. This is because the vanilla approach involves a broker receiving 2–3 times the number of publications in order to account for varying levels of data visibility, where each publication is subject to all message processing operations.

To give a clearer indication of the overheads, Fig. 9.10 presents a breakdown of processing time per event type. This shows that the `drug_audit` and `prescription` events for the IC approach involve a significantly lower processing time than their vanilla counterparts, due to the fact that the transformed events are subject to fewer messaging processing operations.

In the Figure, the *match* category refers to the time the query engine spends in evaluating subscription filters. This is negligible where a subscription is filterless, as is the case for

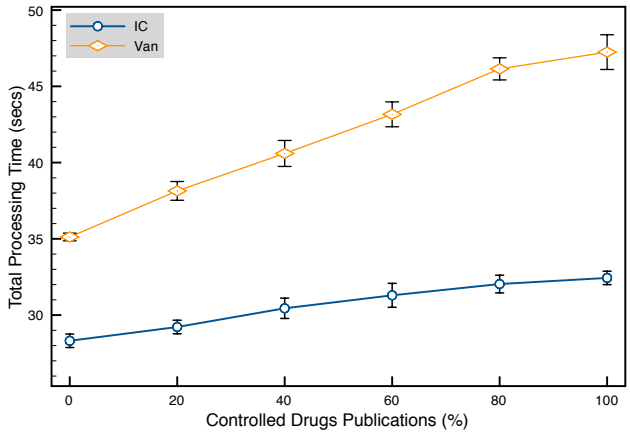


Figure 9.9: Workload processing comparisons.

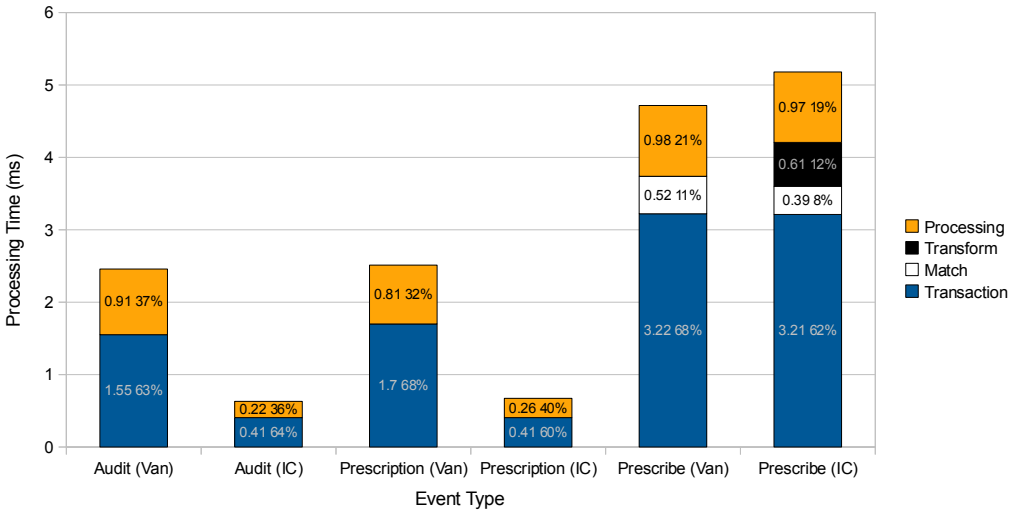


Figure 9.10: Processing time per event type.

the `prescription` and `drug_audit` subscriptions. Matching operations occur within a transaction, the overheads (begin/commit) recorded in the *transaction* category.

`Prescribe` events take greater processing effort as they are subject to more subscriptions, and thus involve more processing and transactions. Although the vanilla implementation processes a `prescribe` event faster due to the extra operations (transformations) of the IC approach, the difference is relatively small with respect to the overall processing time.[11] When considering the total processing time for a prescription *incident*, i.e. accounting for all information flows for the action of prescribing, the IC implementation is shown to be more efficient, taking 6.48ms compared to 9.82ms for the vanilla approach.[12]

Intuitively, one expects event processing to impose overheads. This example based on real-world requirements demonstrates that there are situations where transformations can actually result in an improved workload performance, where middleware data processing serves to reduce redundant publications and the amount of message processing required.

[11]Interestingly, we consistently observed faster matching in the IC implementation, despite both approaches using the same subscription filters. This appears due to PostgreSQL caching functionality, where a transformation function executed on an event instance reduces the time for subsequent queries on the event. Thus, the IC implementation incurs such overhead with the publication transformations.

[12]The total time for the vanilla approach is calculated by summing the mean time per event type.

9.1.7 Distributed Environment

The previous discussion considered the overheads of IC from the perspective of a single broker. In this section we consider the impacts on a distributed broker network.

As in the previous example, information concerning prescriptions must flow to the EPS, regional auditor and doctors who are concerned with a particular patient. Here we modify the example to apply in a hospital.

Fig. 9.11 depicts the topology of the network, where a number of nurses are assigned to work in various wards. Each *ward* maintains its own broker, which is connected to the *central* (ward) broker. The central broker manages all ward information, and thus requires information of (subscribes to) all events raised within the wards. The hospital *dispenser* acts much like a pharmacy, in that they receive all prescription information, but without clinical details. The dispenser is responsible for distributing information to the EPS (for billing purposes), and forwarding audit events for controlled drugs. The *local trust* is a broker that (exclusively) manages the information flowing to health bodies in the locale. In this example, there are two *surgeries* who have doctors subscribing to information on their patients, as well as the regional *auditor* who receives all `drug_audit` and `prescribe` events for nurses under investigation.

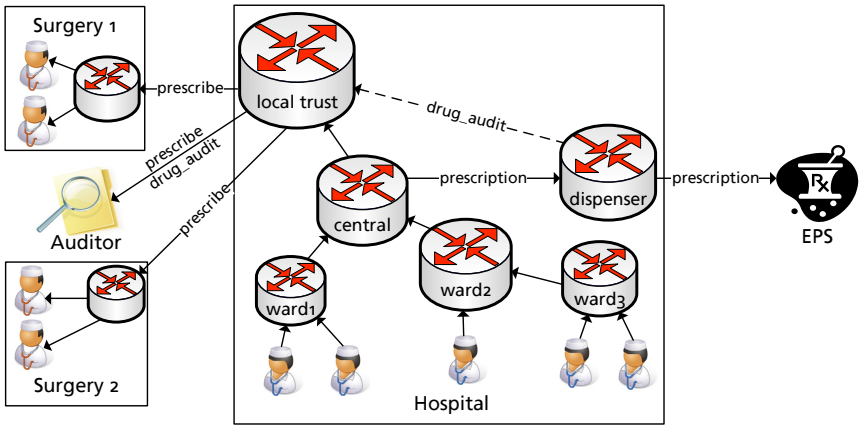


Figure 9.11: The prescribing scenario in a distributed broker environment.

Again, we compare a vanilla implementation against an IC implementation, to show the effect of IC on event delivery within the network infrastructure. There is a slight variation in the routing paths between the approaches.[13] As the dispenser is responsible for all drug related information, we use transformations to cause the central ward to pass the prescription events to the dispenser, who forwards them to the EPS service. The dispenser, being the hospital's authority on medicines, is also responsible for producing the audit events. In the vanilla approach, routing paths are determined solely by (unrestricted) subscription propagation. As `drug_audit` events are produced by the nurses (publishers), they do not come from the dispenser, but instead pass directly from the central broker to the local trust broker for distribution to the auditor. In the vanilla approach, the dispenser only receives prescriptions.

We use the same workload as previously described, where each nurse publishes 1,000 messages, except the number of controlled drugs is fixed to 40% of total publications

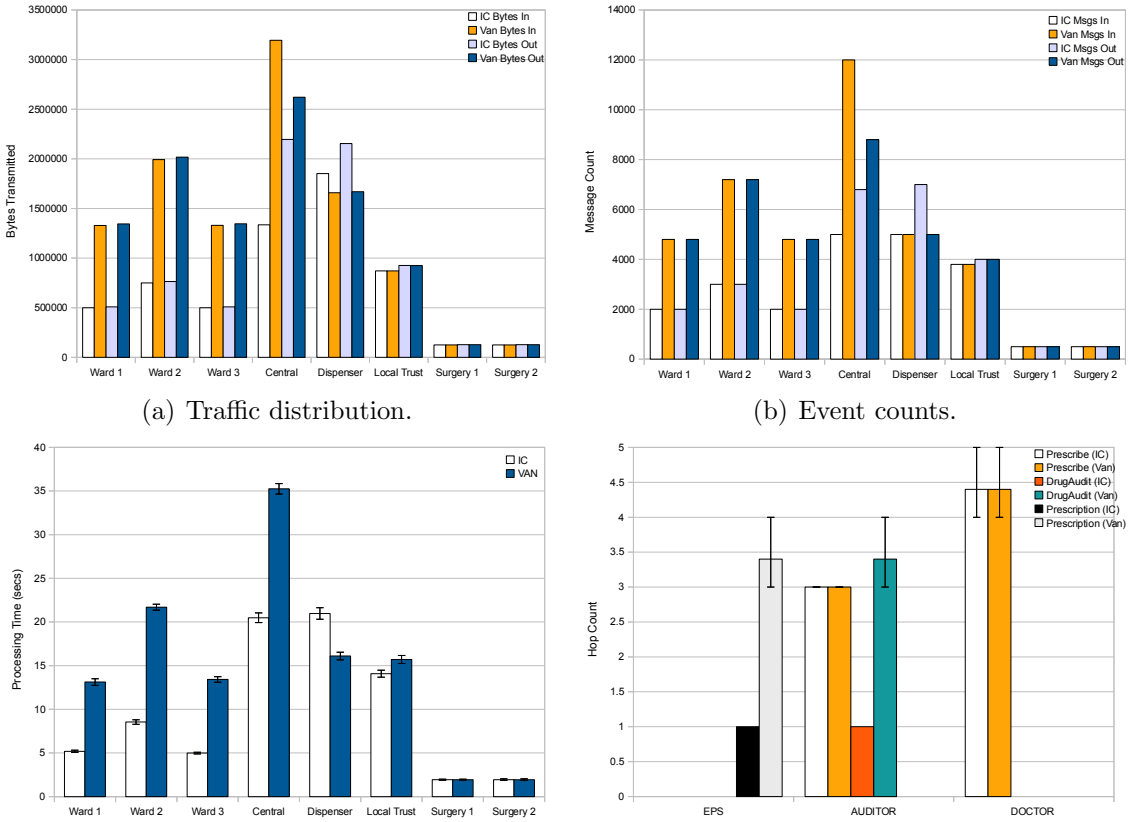
[13]This is indicated in Fig. 9.11 by the dashed line for the `drug_audit` event.

(400/1,000 per nurse). The nurse in Ward3 is under investigation. As in the previous scenario, the timing results are averaged over ten trials.

Fig. 9.12 presents results concerning the overall processing of the scenario. We see that the IC approach results in a ∼36% reduction in bytes transmitted[14] and a one-third reduction in overall processing time. To show the effects of distribution, Fig. 9.13 presents a breakdown of processing for each broker in the topology. Each IC broker incurs equal, or significantly less overhead than its vanilla counterpart, except for the dispenser whose auditing responsibilities require additional operations in the IC implementation.

Measurement	IC Total	Van. Total	Difference	% Reduction
Bytes Transmitted	13,370,183	20,800,872	7,430,689	36%
Events Transmitted	47,600	74,200	26,600	36%
Processing Time (ms)	79,072	118,511	39,438	33%

Figure 9.12: Overall total resource values for the workload.



(a) Traffic distribution.

(b) Event counts.

(c) Total processing time.

(d) Hop counts.

Figure 9.13: Performance comparisons between the IC and Vanilla implementations for the distributed prescription scenario.

Generally, pub/sub models aim to reduce multiplicative event fan-out by pushing subscriptions as close to the publisher as possible. Similarly, transformations can improve event distribution by routing a single copy of an event as far as possible, before trans-

[14]Note that in this experiment the size of an event instance for each type is similar. This is because many attributes are of fixed length, such as identifiers like the NHS Number.

forming it as relevant to the subscriber. This is reflected in Fig. 9.13(d),[15] where the prescription and audit events resulting from a transformation travel only one hop, as opposed to the vanilla implementation where events travel the whole path from the publisher to the subscriber. The other subfigures show that this reduces the event count, byte count and processing time. Where transformations do not impact on propagation, such as in the delivery of `prescribe` events, the IC approach presents similar statistics to the vanilla implementation.

9.1.8 Scenario Discussion

As IC entails event processing, it is useful to consider its overheads. Although this dissertation focuses on confidentiality rather than on performance, the purpose of this section is to quantify the overheads through a scenario based on real-world data requirements.

Our pub/sub implementation uses transactions as part of the delivery process, which is particularly important for healthcare, as all information must be reliably stored, audited, processed and delivered. As such, the use of transactions imposes a base overhead. In this scenario, transformation functions, despite including queries on stored data, did not greatly impact overall performance.

This example shows that transformation functions can, in some situations, result in an overall performance gain. In this scenario, the use of transformations is more efficient than separate event publications that vary in levels of visibility. The effects are pronounced in the distributed example, where IC reduces network traffic and processing time by approximately one-third. Such reductions tie in with data control, as the event is transformed as *appropriate* for the recipient. This reduces message fan-out: transformations more naturally occur closer to the subscribers.

These results are presented for a number of workloads concerning a particular scenario. Clearly the results will vary depending on the complexity of the operations (e.g. transformation functions), and the representation of context. These are all application-level concerns, where performance characteristics depend on the specific requirements of the scenario and the implementation environment. That said, this scenario shows that enforcing data control policy in the middleware does not necessarily degrade performance, particularly when considering the overall workload (total events delivered).

Finally, although a vanilla implementation is useful for performance comparisons, it is important to consider the security aspect. In a generic pub/sub system, clients must be trusted to publish events with the appropriate level(s) of visibility for the potential recipients, and to (only) subscribe to information for which they are authorised. Information is routed through brokers in an uncontrolled manner—brokers are also trusted to behave appropriately. This is unsuitable for environments where data is highly sensitive. The enforcement of policy in the middleware abstracts policies away from clients, leaving the middleware to perform event processing and delivery. This ensures client compliance and facilitates audit, while removing the burden on clients of managing disclosure policy. The next-hop enforcement of policy is in line with overarching notions of responsibility. This section demonstrates that IC policies, in addition to controlling data dissemination, can in some circumstances also improve efficiency and scalability by reducing system load.

[15] The bars represent the average number of hops, where the error-bars represent the maximum and minimum number of hops travelled.

9.2 Regulation Change

There will often be situations where regulations change. Such changes need to be reflected in the policies controlling information flow. The advantage of defining and enforcing policy in middleware is in terms of abstraction and isolation: clients are not burdened with maintaining current policy sets, nor need they be trusted to comply; while a change in policy is effected at specific enforcement points, rather than at each client. In this section we give some examples of proposed alterations to health policy, describing how the changes can be implemented through IC rules.

Following from the previous example, pharmaceutical interest groups are lobbying for pharmacists to have access to health record information [McC07]. Currently, there is a government investigation and some trials attempting to discern the benefits/trade-offs [DoH08]. In the prescribing scenario (§9.1.2), this would require information of observations to be included with the prescription. Such functionality can be realised by altering the `prescription` event type and the transformation function to include notes and observations. Alternatively, an authorisation rule can be defined allowing the EPS (or a particular pharmacy) to subscribe directly to `prescribe` events. This allows such a regulation change to be implemented through the addition/replacement of a single rule and/or function.

Another prescription-related concern regards the use of generic drugs. A National Audit Office report into the prescribing costs of the NHS states that over £200 million a year could be saved by systematically prescribing lower cost, or generic alternatives [NAO07]. A domain could define a transformation rule to transform prescriptions, as per Fig. 9.14, replacing a drug by an (exact) cheaper alternative if one exists.

```
<transformation>
     <rule_name>cheaperdrug</rule_name>
     <event_type>prescription</event_type>
     <output_event>prescription</output_event>
     <interaction_point>n</interaction_point>
     <consumable>t</consumable>
     <function>substitutecheaper</function>
     <conditions>exactGenericAlternative(prescription.drug_id)</conditions>
     <notes>Substitutes drugs for cheaper alternatives</notes>
</transformation>
```

Figure 9.14: Transformation function to automatically prescribe cheaper alternatives.

The NHS is currently piloting *position-based access control* (PBAC) [NHS08b] where privileges are assigned to positions (organisational-roles) within a health institution. PBAC is an attempt to simplify RBAC privilege allocations, by providing local organisations with the ability to define local roles to exploit commonalities between staff members, e.g. that all nurses operating in the *Mulberry Ward* have the same information privileges. IC easily integrates extra considerations, where rule predicates can account for local roles, job functions and other contextual concerns.

It has been suggested that confidentiality can be improved through *anonymous billing*. This involves removing patient details from information flows to health authorities (government) for the purposes of payment [FfIPR05]. Such functionality can be implemented through a transformation function that removes all patient identifiers from event flows to

the billing domain, so only the service provider information and billing codes are transmitted.[16]

These real-world examples highlight the advantage of middleware policy enforcement, where a change in policy occurs only at the brokers, as opposed to every application managing policy specifics. This simplifies policy updates and ensures a consistent enforcement regime within the administrative domain. Context-aware rules provide the flexibility necessary for controlling dynamic environments.

9.3 Sensor-Based Remote Care

Future healthcare scenarios envisage remote care services, where sensor technologies measure environmental and physiological state. This information is forwarded to care staff, who may not be physically present, to make care decisions. Such measurements provide insight as to the patient's well-being [McN00], where health incidents trigger appropriate responses.

We implement a scenario where patients are connected to a range of sensors measuring aspects of physiological state, as well as their present location. This information is published to a broker, which records the data for subsequent query/analysis. Carers may subscribe to the event streams of their patients, to obtain information of the patient's current state of health.

This scenario involves the propagation of patient information, measured through sensors, to subscribers as appropriate to the circumstances. In the general case, summary information is provided, transmitted at regular intervals. In an emergency situation, access restrictions are less stringent to allow subscriptions to raw sensor streams. This provides more detailed information to assist in emergency treatment. We use this scenario to show how changes in context affect information flows, both in terms of the data received by subscribers and in terms of server-load.

9.3.1 Sensor Middleware

Our implementation uses the Hidalgo Equivital sensor module [Hid09]: a wearable device with a number of sensors measuring various aspects of physiological state, including heart and respiration rates, movement and orientation, as well as providing technical feedback, such as sensor failure or disconnection. The module propagates sensor readings via *Bluetooth* [BSIG09] to a sensor-middleware, which manages the sensor module, receives the sensor data and serialises it for publication to the relevant broker.

In this example we consider location data, with regards to global positioning as well as room sensors, if the patient is at home. Although such technologies are readily available [PCB00, HHS+02], location data was simulated due to lack of hardware.

9.3.2 Data Control Rules

The data streams are propagated to the broker managing the particular patient, which stores the data to give a detailed picture of the patient's recovery process. Carers can

[16] This operates in a similar manner to the `drug_audit` transformation previously described.

Event Type	Producer	Publication Rate	Description
`sensor_snapshot`	Sensor MW	2 secs	A periodic event providing a snapshot of the patient's current physiological state.
`movement`	Sensor MW	1–5secs	Informs of the detection of a change in patient state concerning movement, orientation or location. Events are delivered on a change in state, or every 5secs if no change is detected. Publication rate is approx. 1 event per second in situations of constant change.
`status`	Sensor MW	On Event	Records patient status. Indicates the start/cessation of perceived emergencies, where readings exceed a safe threshold. This event is used to initiate/terminate emergency fluents
`ecg_reading`	Sensor MW	16/sec (62.5ms)	This encapsulates the live ECG stream. An event containing a buffer of 16 ECG readings. The reading timestamp is used to account for messaging delays.
`vitalsigns`	Broker	On Sensor_Snapshot	A combination (mashup) of sensor data, aggregating the last received snapshot, movement and status data for the patient, executed through a transformation function.

Figure 9.15: Event type descriptions.

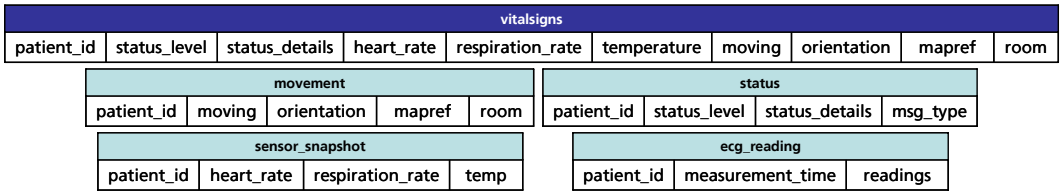


Figure 9.16: Event type schemata for the sensor scenario.

subscribe to event streams, subject to policy constraints. Figs. 9.16 and 9.15 present the event types used in this scenario.

There are several restrictions imposed on the data flows of this model, the definitions for which are shown in Fig. 9.17. Firstly, subscriptions are only authorised if the subscriber has a treating relationship with the patient. This requires both a permission attribute, and a monitored condition. As in Fig. 9.17(b) clients can generally subscribe to the `vitalsigns` event, which provides a summary as to the patient's current state, and the `status` event type, which informs of perceived emergency situations. Clients (doctors) can subscribe to the other event streams to obtain more detailed information, but only when the patient is an emergency.

The `vitalsigns` event is used to control the level of information disclosure. Firstly, the summary itself serves to remove some less pertinent information, such as the patient's orientation. Location is useful for interpreting readings; if a patient is at home, they are less likely to be subject to external stimuli, and thus should be more relaxed/stable. For reasons of privacy, their exact location is perturbed, except for emergency situations in which the location is relevant for emergency care. This perturbation is implemented as a notification transformation (Fig. 9.17(c)) applying only to doctors.

In situations where readings fall outside particular ranges or exceed thresholds, the event infrastructure is used to alert the relevant parties of emergencies, through `status` events. A perceived emergency represents a significant change in context, thus affecting the access control rules: the access restrictions are relaxed (Fig. 9.17(a)) and the perturbation transformation does not apply in an emergency (Fig. 9.17(c)). Note that `emergency` is *not* a monitored condition because it may be useful to continue monitoring the ECG stream for some period after an emergency is considered resolved, e.g. in case of a relapse.

```
<request_authorisation>
     <rule_name>ecgsub</rule_name>
     <event_type>ecg_reading</event_type>
     <request_type>s</request_type>
     <credentials>NHSCred(usernm,'doctor')</credentials>
     <permission_attributes>patient_id:int8</permission_attributes>
     <mon_conditions>treatsPatient(usernm, att.patient_id)</mon_conditions>
     <conditions>emergency(att.patient_id)</condition>
     <notes>A doctor can subscribe to their patient's ECG stream in an emergency</notes>
</request_authorisation>
```

a) `ecg_reading` Authorisation Rule. The rule is similar for the `movement` event type.

```
<request_authorisation>
     <rule_name>vitalsignssub</rule_name>
     <event_type>vitalsigns</event_type>
     <request_type>s</request_type>
     <credentials>NHSCred(usernm,'doctor')</credentials>
     <permission_attributes>patient_id:int8</permission_attributes>
     <mon_conditions>treatsPatient(usernm, att.patient_id)</mon_conditions>
     <notes>A doctor must treat the patient to subscribe</notes>
</request_authorisation>
```

b) `vitalsigns` Authorisation Rule. The rule is similar for the `status` event type.

```
<transformation>
     <rule_name>perturbvitalsigns</rule_name>
     <event_type>vitalsigns</event_type>
     <output_event>vitalsigns</output_event>
     <stage>n</stage>
     <consumable>t</consumable>
     <function>removesensitivedata</function>
     <credentials>NHSCred(usernm,'doctor')</credentials>
     <conditions>not emergency(vitalsigns.patient_id)</conditions>
     <notes>Perturb location for Doctors</notes>
</transformation>
```

c) `vitalsigns` Transformation Rule.

Figure 9.17: IC rules for the sensor scenario.

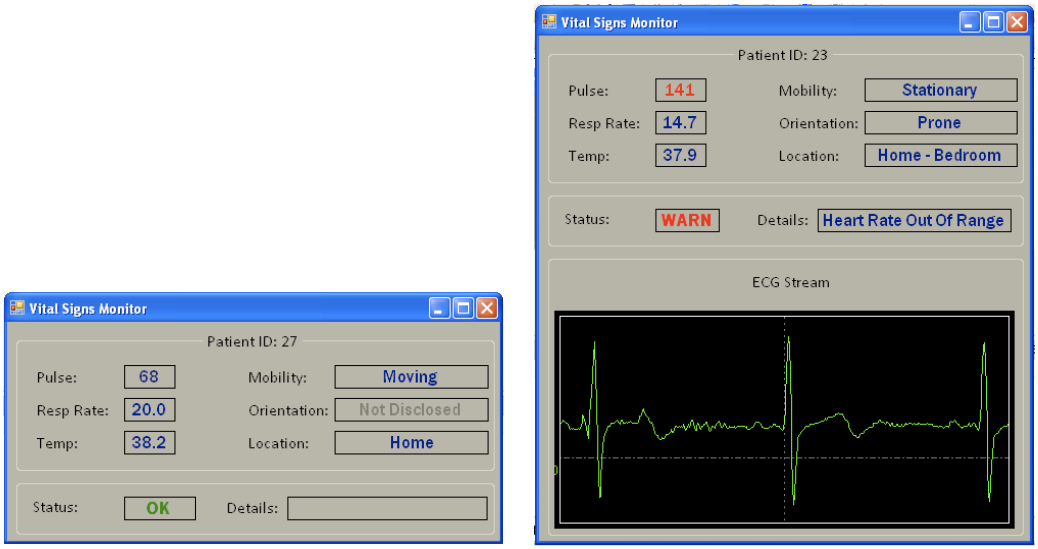
Overall, this helps to bring a context-aware level of privacy for patients. In a perceived emergency it is important that information is made available. However, in the general case, events are perturbed to provide less detail.

9.3.3 Experimentation

The sensor middleware publishes a number of events per second for each patient, which must be stored and in some cases transformed and forwarded. In practice, we expect that a surgery provides the infrastructure to handle numerous patients at a time. Here, the focus is on the impact of contextual change, rather than on the numbers themselves. As such, our experiment consists of a single broker monitoring the streams for 10 patients, as this balances the storage and processing requirements of the scenario. As we have access to only one sensor module, we simulated multiple patients by publishing a number of pre-recorded event streams.[17] We use this scenario to show the effect of context on data flows, where an emergency situation alters event visibility and impacts system load.

Screenshots from our client application are shown in Fig. 9.18. The application subscribes to a patient's streams, presenting a dashboard of patient statistics. The application automatically subscribes to the `ecg_reading` stream when a `status` event indicates an

[17] The timing intervals between events are preserved.

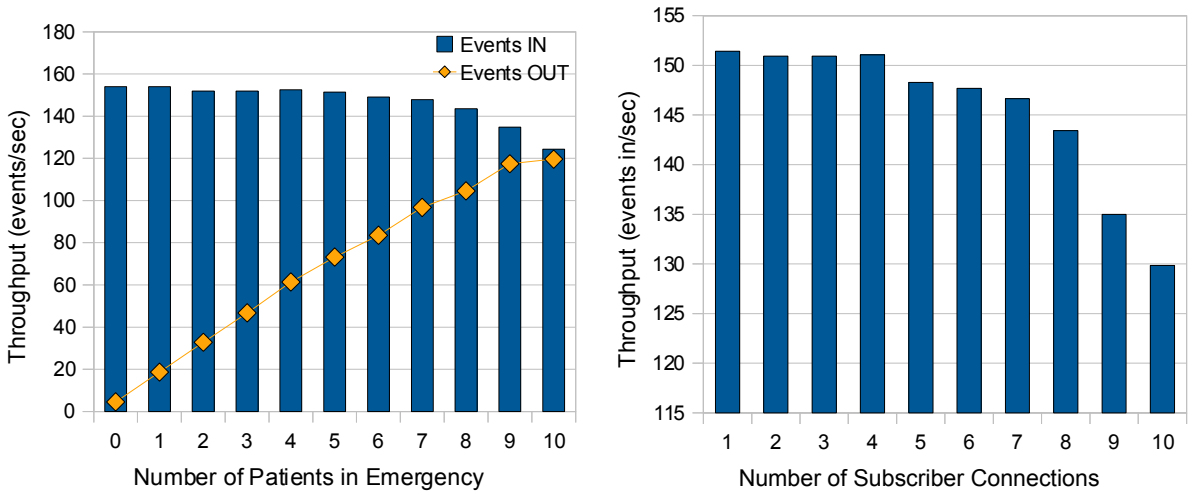


(a) Non-Emergency Patient Stream (b) Emergency Patient Stream

Figure 9.18: Screenshots from the monitoring application.

emergency. Fig. 9.18(a) shows the general dashboard, where some information is restricted and perturbed. Fig. 9.18(b) shows the application when a patient is in an emergency, presenting complete location information as well as a plot of the ECG.

To experiment with the effects of context on the broker, we altered the number of patients in a perceived emergency situation. As clients automatically subscribe to `ecg_reading` events on detection of an emergency, the number of events delivered increases with the number of patients in an emergency. This is depicted in Fig. 9.19(a).



(a) Scenario throughput (b) Throughput per connection, 10 patients in an emergency

Figure 9.19: Results for the sensor scenario.

We notice that the rate of incoming events stays relatively constant until about 7 patients are in emergency. The degradation in performance is due to the subscription matching operations, where Fig. 9.19(a) assumes that each subscriber maintains a separate connection when subscribing to the `ecg_reading`. That is, 10 separate subscribers to each

`ecg_reading` results in approx. 1,600 (160 events $*$ 10 subscriptions) subscription matching operations executed per-second of processing. Subscription filters on an event type are evaluated in aggregation (conjunction) for a subscriber: there is a single evaluation per event-type, for all subscriptions of a client's connection. Fewer evaluations involve fewer transactions and calls to the query engine, thus reducing the effects on throughput. This is reflected in Fig. 9.19(b), which divides the 10 subscriptions to the `ecg_reading` amongst a varying number of connections. In this Figure, all 10 patients are in an emergency. This is significant for topological design, given that brokers that interconnect through links can maintain several subscriptions for an event type (through subscription forwarding). In this scenario, Fig. 9.19(b) shows us that if the clients were to subscribe through up to four brokers, that there is no performance degradation even in the unlikely case where all patients are simultaneously in an emergency.

There are various design considerations that can improve performance in this scenario. The most obvious is to have the sensor middleware produce fewer messages, by buffering a greater number of ECG readings, or to have a broker manage fewer streams. We observed that the CPU load in our experiments did not exceed 54%. Given that our proof-of-concept implementation involved a single message-processing thread, we would expect performance to improve with multiple worker-threads.

Our implementation performed reasonably in the general case, degrading in the unlikely situation wherein most patients are in a critical state and all subscribers are separately connected to the broker. Despite the performance impact, we did not observe a difference from the client perspective even when all patients were in an emergency situation.[18] The `ecg_reading` events include the time of the reading to enable clients to account for variable delivery rates. In line with this, our client application buffers `ecg_reading` for 2 seconds before displaying the plot. In our experiments, the results appear reasonable given that this scenario requires 'human-speed' rather than 'machine-speed' response times.

9.3.4 Scenario Discussion

The purpose of this scenario is to present an example implementation in which sensor data is reliably stored, processed and delivered with respect to context-sensitive data control rules. This involved using our middleware to feed real sensor data to several instances of a monitoring application. We show the effects of a change in context on the system, both in terms of the data the client receives, with regard to perturbed results and visualisation of the ECG stream, and in overall system load.

A broker's performance requirements depends on the particular situation. In practice, we expect that contextually significant events require storage and propagation, such as a change in patient status (emergency). However, it can be argued that general monitoring data, such as the ECG data stream, or other high-volume data streams such as video, need not be stored in a relational database nor processed transactionally; or if so, stored in batches/summaries. That said, we found that in our implementation a single broker could, in general, appropriately store, propagate and control data streams for 10 patients; with performance degrading in the extreme case where all patients are in a perceived emergency situation and all subscribers are directly (separately) connected to the broker hosting the sensor stream.

[18] Over time the effects may become noticeable, depending on broker load.

9.4 Contextual Complexity

IC is context-sensitive. Contextual predicates (fluents) are evaluated throughout the messaging process: credential predicates, the filters on events, the guards for transformation rules, and the monitoring of conditions all require evaluation. It follows that the method of contextual representation impacts system performance.

To illustrate this point, we take the sensor scenario where `movement` events define the patient's current location. These events occur sporadically; though the rate is in the order of one event every 1–5 seconds. A patient's location can be maintained in a fluent: `atHome(patient-id)`. The manner in which this fluent state is derived/maintained can greatly affect performance.

We take four approaches to implementing the fluent, the query for each is presented in Fig. 9.20. The first approach makes use of a materialised view, which uses a table dedicated to storing the current state of the fluent—whether the patient is currently at home. The second approach determines the fluent state by performing the query on demand, through use of a join between the last known location of the patient and their registered home-location. The third approach is similar to the join, but instead uses a subquery, which is typically expensive to evaluate. The fourth executes a call to a remote server (through a socket) on an external subnet, which maintains an in-memory hash table of the patient-home status.[19]

```
select home from fluent_athome where patient_id = $1;
```

a) Materialized View

```
select pg_log_in_movement.location = patient_home.location
     from pg_log_in_movement left join patient_home on pg_log_in_movement.patient_id =
          patient_home.patient_id where patient_home.patient_id = $1
               order by processed_time desc limit 1;
```

b) Join

```
select count(patient_id)=1 from
     (select patient_id as pid, location as loc from pg_log_in_movement where
          patient_id = $1 order by processed_time desc limit 1) as A
               inner join patient_home on a.pid = patient_id and location = A.loc;
```

c) Subquery

Figure 9.20: Queries for the different definitions of the `atHome` fluent.

To illustrate the performance differences, each fluent representation was used as a guard for a transformation rule. The average fluent evaluation times for 2,500 queries over 10 trials are presented in Fig. 9.21.

Our results show that in this scenario, the materialised view is clearly the most efficient fluent implementation. However, when considering materialised views, it is important to account for the overhead in maintaining its state. To quantify this, we measured the time taken to update the fluent on receipt of a `movement` event. Our trials under similar conditions found the update to take on average ∼0.21ms.

[19] We omit the code which calls the remote server, as it involves a procedure using sockets.

Fluent Type	Average Processing Time
Materialised View	0.25ms
Left Join	1.98ms
Subquery	3.79ms
Remote Server	3.31ms

Figure 9.21: Processing times per fluent type.

Although the numbers are in the order of milliseconds, the effects show up in aggregation, where a broker may perform thousands of evaluations per second. In the sensor scenario, the materialised view is the favoured approach, as `movement` events occur relatively infrequently (in the order of seconds, cf. the ECG stream every 62.5ms) and the cost of maintaining the view is comparatively low. However, if the scenario is one such that the state is expected to change more frequently than queries on the condition itself, it may be beneficial to query the relevant tables on demand to avoid the performance and storage overheads of maintaining the view. The appropriate method for accessing context is clearly a question of circumstance.

The representation of context is an important consideration for IC policy designers, to ensure that the implementation meets the requirements of the scenario.

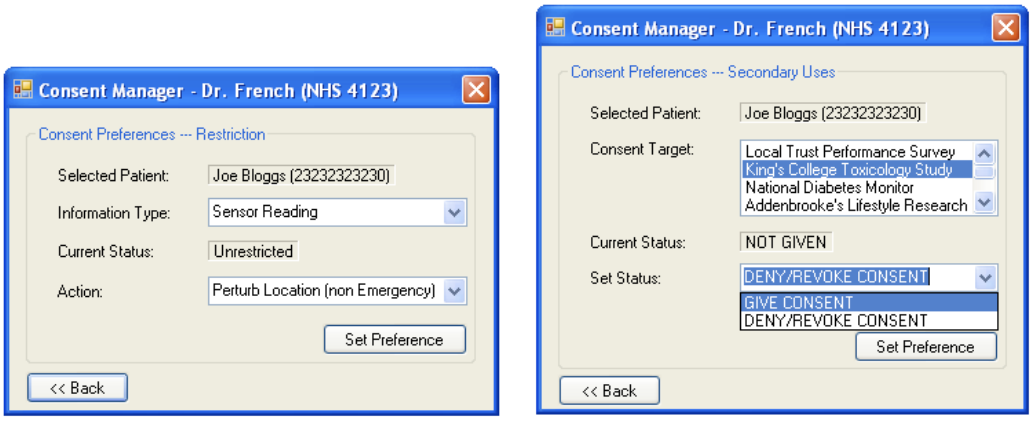
9.5 Policy Interface

Much of this dissertation focuses on policy defined by a system administrator. However, patient specific requests and aspects of consent will also be specified by non-expert users. Clearly the definition process must be simple. To ease authoring, restriction templates can be defined by domain administrators for use by policy authoring applications. Given that IC policy is stored as data, rules can be defined through similar interfaces to other data manipulation applications. Further, restrictions do not necessarily entail the creation of a rule; often a restriction can be implemented through manipulating fluent state, which is referenced by a rule of the domain's sharing protocol. As an example, we use a simple interface to illustrate a mechanism to effect the consent preferences of patients.

Fig. 9.22(a) presents an interface that controls the perturbation of a patient's sensor data stream. The process involves selecting the particular patient, followed by the type of information and the operation. This generates a `consent` event that encapsulates the details of the preference, which is routed through the local domain, consumed by the broker(s) maintaining the `perturbOutput(patient-id)` fluent. This fluent is referenced by the rules effecting the perturbation—the value of the fluent determines whether the transformation degrading the event applies.

Fig. 9.22(b) presents an interface for controlling data flows to an external service. The relevant secondary services are populated in a list, in which the user, in this case a doctor acting on behalf of a patient, sets the appropriate levels of consent. This, again, generates an event which is consumed by the relevant broker(s) to allow/prevent information transmission to the particular service.

These examples, although simple, illustrate that policy authoring can be facilitated through a database frontend—much like many enterprise applications. As such, similar techniques can be used to develop convenient interfaces for specifying complex policy preferences, including the definition of IC rules.



(a) Setting perturbed preferences (b) Screenshot for setting SUS preferences

Figure 9.22: Consent authoring screenshots.

9.6 Summary

This chapter describes the application of IC to particular healthcare scenarios. We do not attempt to provide a performance analysis of IC, as our implementation forms merely a proof-of-concept. Instead, we take specific scenarios based on real healthcare requirements to justify the properties of IC and to highlight various policy formation and implementation considerations.

IC introduces extra functionality to the pub/sub messaging process. The prescription scenario is an attempt to quantify this overhead. We observe that a significant proportion of processing time relates to transactions. We advocate the use of transactions to enable reliable event processing. Given that health information must be audited, some transactional and storage overhead is likely to be incurred regardless of the implementation.

Our comparison of an IC implementation to a vanilla pub/sub approach illustrates that IC does not necessarily degrade overall performance, but in some circumstances actually improves scalability. Intuitively, processing overheads can be reduced given that restrictions and transformations can limit information flows, thereby reducing event fan-out. This, of course, depends on the nature of the scenario, there will be situations where IC policy imposes an overhead. Our experiments indicate the scalability of IC, while providing useful information for policy authors.

The two approaches, however, differ significantly in terms of information protection: IC controls disclosure by enforcing policy on all connections, as opposed to the uncontrolled vanilla approach, which must trust clients and brokers to act appropriately. Even without malice, there are still issues of negligence and curiosity. Policy management concerns aside, the vanilla approach is insufficient even in an environment of overarching responsibility, as accountability is diminished. This chapter highlights the advantages of middleware enforcement, where changes in disclosure policy are abstracted away from clients, easing management and ensuring client adherence.

IC is context-sensitive. We use the sensor application to show how policy rules interact with contextual changes to impact on information flows. This in turn affects system load, thus it is important to consider data flow requirements when structuring the topology. We also demonstrate that the method for implementing contextual state affects performance. Such issues must be considered when realising an IC implementation.

IC provides a framework for controlling information flow. We have applied IC to health scenarios derived from health policy and procedure documents. As policy is user-defined, system evaluation depends on the particular requirements of the situation. Given that data protection is a fundamental healthcare concern, the performance characteristics of the control mechanisms must be considered in the context of complete, real-world scenarios. Indeed an area for further work is to analyse IC with respect to real(istic) healthcare workloads, accounting for the number of clients, data load and required processing/response rates, information types, and policy constraints of an actual healthcare deployment.

10
Audit and Event Replay

Lessig [Les99] describes four categories of constraints that influence and regulate behaviour: legal, social, economic and architectural. In healthcare, constraints operate to control the use and influence the protection of sensitive information. There are legal ramifications for negligence or malice in healthcare, be it the failure to protect information or to adhere to standards. Medical professionals take an oath, and are bound by codes of conduct. Reputation is important. Knowledge of possible audit and peer-review instills social pressure on staff to behave in a certain way. Economic factors are involved, considering the costs and benefits of taking particular actions and precautions: a failure to respect confidentiality might entail fines or termination of employment, while for organisations, negative publicity deters business. The architecture refers to what is (physically) possible; for instance, data must be stored in order to be accessed. IC is an information disclosure architecture, where policy defines the circumstances for data release. Audit provides information (evidence) important for regulating the use of health information.

The purpose of audit is to ensure that medical professionals and organisations adhere to the appropriate standards of care and conduct. In other words, audit brings about accountability. While confidentiality underpins the health service, healthcare is not an environment of anonymous interaction. Instead, the actions of (non-patient) users must be audited. From an information standpoint, patients must feel confident that their information is handled correctly; patients may withhold sensitive information from practitioners for fear of inappropriate disclosure [Gol98, WHEH06], which may impact on their well-being. The NHS Constitution [DoH09b] states that health organisations are responsible for meeting safety and quality standards. Organisational policies set the goals served by the technical infrastructure. This involves defining appropriate access privileges, and creating mechanisms for preventing and detecting violations, backed by sanctions [MY04, HSH$^+$09]. The NHS defines *arms-length-bodies*: independent organisations that review and hold to account health providers at all levels of the health service, from the executive to that of local practice [DoH09c, DoH09a].

Most large scale systems are designed to include audit capabilities. Audit logs provide visibility, reflecting past state and processes. At a system level, audit is used for safety

and recovery, and to monitor the effectiveness of system processes. Audit involves more than merely recording data: audit brings about accountability. It provides the means for influencing behaviour, producing the evidence on which to investigate system, higher-level (organisational) issues and individual conduct. Further, audit mechanisms can alert of situations as they arise, and assist in the discovery of policy errors.

This chapter begins by considering the system-level aspects of auditing, which involves recording system state at various stages of the messaging process. We then consider *active audit*, where the event infrastructure can be used to alert of particular situations as they occur. In an environment of dynamic control, historical information brings interesting considerations, where a change in context can alter privilege in some way, or make relevant a previously insignificant or unauthorised event. As such, we extend IC to deal with historical event dissemination *(event replay)*, before concluding with a discussion of the approach and areas for future work.

10.1 Recording Information

Information is the "lifeblood of NHS organisations" [DoH07a]. It follows that information governance is a fundamental healthcare concern. To enable proper inspection, monitoring and investigation, information surrounding health data and user actions must be recorded. This is reflected, for example, in the NHS Care Records Guarantee [NHS07b] which describes the need for audit trails that record access and modifications to patient information. As the middleware enables communication, it has the capability to monitor and record actions and occurrences. The Security Audit Requirements document by the US Department of Veteran Affairs [HIA06][1] summaries the requirements for healthcare auditing as specified by various political, legal and industrial instruments. Requirements relevant for a healthcare middleware include that:

- The audit trail provides sufficient information: events, time of occurrence and the cause (who did what, when);
- Content is recorded both before and after the performance of actions (queries, views, insertions, deletions, conversions);
- Access requests and privilege assignments are recorded;
- Capabilities exist to alert of data changes, e.g. anomalies, security breaches, etc.;
- Audit requirements can be customised for specific applications and environments;
- Audit traces can be distributed and transferred.

These requirements make it clear that it is insufficient to merely log the events sent and received; it is also necessary to record the policies authorising (or denying) the request, as well as the context for the decision [Bak07, Bak04]. This is particularly important for IC as policies are context-sensitive. Alerting is facilitated by an event infrastructure (§10.4). As IC operates within the database space, database infrastructure is used to implement the audit. This section outlines how audit mechanisms are integrated into a IC broker.[2]

[1] Who incidentally collaborates with the NHS on information sharing and governance issues [NHS06d].

[2] The volume of audit information will likely exceed that of the live, currently used data. This issue, however, is common to many enterprise applications. To deal with this, data might be archived at regular

10.1.1 Message Receipt/Delivery

It is necessary to record the information that enters and leaves a broker. The structure for storing this information, as presented in Fig. 10.1, consists of separate tables to store the requests and the events sent and received for each event type.[3]

log_[event-type]_received					
msg_id	in_queue_id	username	ackmsg	errorcode	logtime

log_[event-type]_sent					
delivery_id	out_queue_id	username	ack	errorcode	logtime

log_[request-type]_received						
msg_id	request_id	username	eventtype	ackmsg	errorcode	logtime

log_[request-type]_sent						
delivery_id	request_id	username	eventtype	ackmsg	errorcode	logtime

Figure 10.1: Tables recording the the inputs/outputs of a broker.

The `username` references the unique ID of the relevant connected principal. The `msg_id` and `delivery_id` refer to the ID of the serialised (XML) message. Internally, a tuple is assigned a request/event ID, to identify it throughout the messaging process. These tables do not directly store the information contained within the event/request tuple; such information is accessed through `joins` between the ID and the the appropriate event/request history store—as described in the next section.

10.1.2 Event Auditing

As described in Ch. 7, message processing involves moving events between queues. For the purposes of audit, we define a separate history table (or *log*) for each queue. These tables record a history of queue entries supplemented with additional information, such as timestamps and applied policies. Fig. 10.2 presents the structure of the queues and the associated logs for the event type `ev`, consisting of two integer attributes `A` and `B`.

Transformations work to alter an event instance, or create a new one. This information must be recorded, as the event is no longer the same as that originally published. As shown in the Figure, transformation outputs are stored along with the function and hook rule identifiers, and references to the event instance before transformation.

The result of evaluating a subscription/restriction filter is not explicitly recorded. This information can be derived through a query, where an event on the `match` log has no corresponding entry on either the `out` or `exception` log. Publications that fail to satisfy the applicable advertisement filters are stored in a separate log because such events are prevented from reaching the `in` queue. This log takes the same form as `pg_log_in_<type>`.

intervals, perhaps distributed through the event infrastructure. We do not consider these issues as part of this dissertation, as these are a question of design, related to the particular organisation, its technical infrastructure, the volume of data processed, the information stored and available resources.

[3]Forwarded requests are audited through triggers on changes to the *Advertisement_Out* and *Subscription* routing catalogues. PostgreSQL requires modification to allow triggers to be defined on catalogues.

pg_queue_in_ev			
eventid	*publisher*	*A*	*B*
401	nhs_321	4	9

pg_log_in_ev				
eventid	*publisher*	*logtime*	*A*	*B*
401	nhs_321	2009-01-03 11:21:41.002	4	9
399	nhs_7162	2009-01-03 11:21:37.668	2	8

pg_queue_match_ev			
eventid	*subscriber*	*A*	*B*
401	nhs_223	4	9

pg_log_match_ev										
eventid	*subscriber*	*publisher*	*hkoutput*	*hkname*	*hkfunction*	*inevid*	*inevtype*	*logtime*	*A*	*B*
401	nhs_223	nhs_321	n					2009-01-03 11:21:41.003	4	9
400	nhs_223	nhs_7162	y	perturb halfdbl	99313	399	ev	2009-01-03 11:21:37.671	4	4

pg_queue_out_ev			
eventid	*subscriber*	*A*	*B*
402	nhs_223	8	9

pg_log_out_ev									
eventid	*subscriber*	*hkoutput*	*hkname*	*hkfunction*	*inevid*	*inevtype*	*logtime*	*A*	*B*
402	nhs_223	y	perturbdbl	81002	401	ev	2009-01-03 11:21:41.006	8	9
400	nhs_223	n					2009-01-03 11:21:37.672	4	4

Figure 10.2: Queue schemata and their associated audit tables.

10.1.3 Exception Queues

Exception queues are used for recording system errors, where a failure in processing occurs due to some unforeseen circumstance or coding/function error. Such information is aimed at system administrators to resolve system defects. We define the following exception queues for different points of failure.

Incoming_Exception Stores information regarding the messages received that could not be deserialised. This might be due to malformed XML, or some type mismatch in constructing a tuple.

Request_Processing_Exception Records the request tuples for which validation processing failed.

Request_Forwarding_Exception Records the request tuples that failed as part of the forwarding process.

Exception_In* Records a failure in processing an event from the `in` queue. This is likely due to a failure in publication transformation rule resolution or function execution.

Exception_Match* Records a failure in processing an event from the `match` queue. This is likely due to a failure in notification transformation rule resolution, function execution, or delivery filter evaluation.

Exception_Out* Records a failure in processing an event from the `out` queue. This might be due to some error in serialisation or a network anomaly.

The * denotes that the queue is defined for each event type. The schema of an exception queue is the same for that as its associated log, except that it includes the transaction ID and a field to store additional information about the failure.

Triggers defined for the queues can alert of the exception.

10.1.4 Conflict Resolution

In addition to auditing the data sent and received, the rules authorising privilege are recorded [Bak07]. This is important not only for reconstructing prior circumstances, but also for policy evolution.

Conflict resolution information is particularly important as it details the rules applicable and applied at an enforcement point. This, in turn, indicates the state of affairs at the time of evaluation. The conflict resolution function, which is called by all functions requiring policy resolution, is extended to persist conflict resolution information in a table of the following structure:

id The identifier of the connection/request/event in which the resolution occurs.[4]

user The principal ID in the context of evaluation (publisher/subscriber/broker).

input_policies[ ] The active set of policies to resolve.

output_policies[ ] The resolved set of policies, or `null` if incompatible.

resolution_policies[ ] The set of applied conflict resolution policies.

description Any additional notes, derived from the policy definitions.

Log_Link_Authorisation_Resolution					
conn-id	*broker-id*	*input_policies[]*	*output_policies[]*	*resolution_policies[]*	*description*
22	NHS_6614	[authorise-surgery, NHS_6614_auth]	[NHS_6614_auth]	[6614priority]	Local surgery NHS_6614 takes priority
23	NHS_104	[authorise-surgery]	[authorise-surgery]	null	Authorise any surgery

Log_Transformation_Resolution_[event-type]_Notify					
event_id	*subscriber*	*input_policies[]*	*output_policies[]*	*resolution_policies[]*	*description*
845	NHS_4412	[polov,original]	[polov]	[consume-polov]	polov consumes original
906	NHS_4412	[polov2,diffov, original]	[polov2,diffov, original]	null	null
907	NHS_4412	[polov,diffov original]	null	[fail-polov-diffov]	polov and diffov are incompatible

Figure 10.3: Conflict resolution audit tables.

A table exists to store the resolution information for each type of rule: link authorisation, request authorisation, imposed conditions or transformation. The `id` and `user` attributes are passed to the resolution function as a parameter. Fig. 10.3 presents example resolution histories for transformation and link authorisation rules.

10.1.5 Request Auditing

Request auditing functionality is implemented within the request validation function, as previously described in §7.6.2. Active requests, i.e. requests pertaining to an active event channel, are maintained in a catalogue reflecting the structure of the request tuple. An insertion to this table represents the creation of a channel, and deletion a closure of the channel. Modifications to an event channel are recorded in a history table, along

[4]When resolving request authorisation rules, the two identifiers are passed: the request ID and the evaluation ID (see §10.1.5).

with the operation and associated timestamp. This occurs through a trigger (Fig. 10.4). Unauthorised requests that never become active are manually written to the history table as part of the validation process. The *Links* catalogue uses similar mechanisms to record the history of broker interconnections.

```
CREATE OR REPLACE FUNCTION logreq_sub() RETURNS TRIGGER AS $$
BEGIN
IF (TG_OP = 'DELETE') THEN
       INSERT INTO requesthist_sub SELECT 'CLOSED', now(), OLD.*;
       RETURN OLD;
ELSIF (TG_OP = 'UPDATE') THEN
       INSERT INTO requesthist_sub SELECT 'RE-EVALUATED', now(), NEW.*;
       RETURN NEW;
ELSIF (TG_OP = 'INSERT') THEN
       INSERT INTO requesthist_sub SELECT 'CREATED', now(), NEW.*;
       RETURN NEW;
END IF;
RETURN NULL;
END;
$$ language plpgsql;

CREATE TRIGGER aug
AFTER INSERT OR UPDATE OR DELETE ON activesubcriptions
FOR EACH ROW EXECUTE PROCEDURE logreq_sub();
```

Figure 10.4: A trigger to record the history of the active request table.

The request validation process generates auditable information, which is persisted in a number of dedicated log tables. Given that requests can be re-evaluated, the request tuple is extended to include an extra attribute (`eval_id`) to distinguish each evaluation. This attribute is assigned a unique value each time a request is evaluated, enabling identification of the audit information generated from each validation process.

Fig. 10.5 presents the relationships between the request history and the logs. The conflict resolution process (§10.1.4) records the applicable/applied request authorisation and imposed condition rules. The transformation hooks applicable to a channel are manually inserted into a separate table recording the request ID, evaluation ID, transformation policy ID and the name of the hook rule. The `operation` field describes whether a hook was created or removed from an event channel: a transformation may no longer be relevant after re-evaluation.

Request reprocessing involves executing the request validation function on an active request. A record of the re-evaluation is maintained in a separate table storing the request and evaluation IDs and time of reprocessing, along with the `description` parameter, which describes the monitor function triggering the re-evaluation. The auditing process for request re-evaluation is essentially the same as that for the initial validation, except that a request is removed from the active requests catalogue if it is no longer authorised.

Request Forwarding

Forwarded requests must be recorded as they pass information to another broker. Such information is useful; for example, describing why a certain domain did not receive a particular advertisement. Auditing is relatively straightforward. The table maintaining the

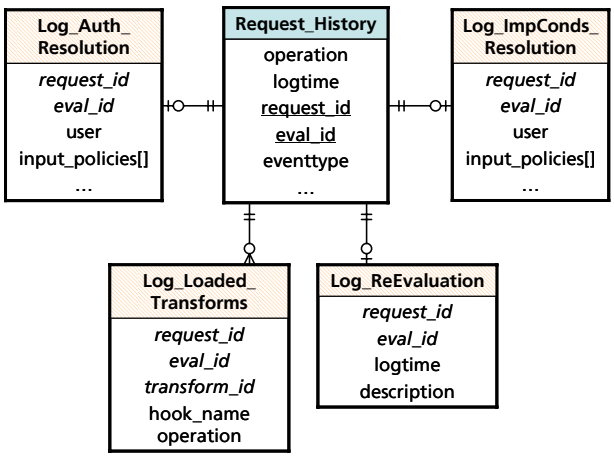


Figure 10.5: Relationships between request log tables.

history of broker connections is associated with logs maintaining the history of forwarded requests. These record the details of the request tuples, the transformations and filters applied, and broker acknowledgements.

10.1.6 Policies

The policy definitions for each rule type are stored in separate catalogue tables (§7.4.2). Each policy catalogue has associated with it a history table, as shown in Fig. 10.6 where all modifications to policy definitions are recorded. This log stores the author, modification time, update type and a description regarding the change (if provided), along with the values of the rule definition (represented by the ellipses). The history is maintained through triggers, similar to that of Fig. 10.4.

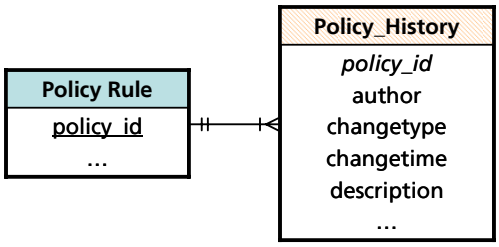


Figure 10.6: Relationship between a policy definition and its history table.

10.2 Contextual History

As described, IC audit tables record the rules previously activated and applied. This implicitly provides information of historical state, as rule predicates must have held for it to apply. However, given that IC is context-sensitive, it is generally useful to maintain a historical record of contextual changes to provide a view of the system at a particular point in the past [Bak07, Bak04]. This provides extra information for accountability purposes, and assists in investigating errors—an incorrect disclosure might not be due

to defective policy, but perhaps due to a flawed contextual definition. Further, having a historical representation of context is particularly relevant for event replay (see §10.5).

Fluents are inherently time-based [Sha99]. The state of a fluent can be derived through the `HoldsAt(f,t)` predicate, which searches through the event log to determine the fluent value at a particular point in time. As IC implicitly maintains event histories, contextual state is visible at a time in the past. However, when querying prior states of affairs, one must account for any subsequent changes in fluent initiation/termination definitions.

The enforcement of IC policy concerns the current state of a fluent. Earlier, we described how the current value of a fluent can be maintained through a materialised view to avoid searching the event history on each evaluation. In this case, the active rules/functions maintaining the view can be extended to record the time and even the reason for the change in the fluent value. This creates a historical log of state changes, explicitly recording the times in which a particular fluent held.

10.3 Accessing the Log

Audit logs provide much information useful for investigative purposes and for policy/process refinement. There exists work in determining possible data leaks based on analysis of the audit log [AEKV07]. It is said that the sensitivity of a log is equal to that of the most sensitive information it holds [HIA06]. In fact, it is arguable that audit information is more sensitive, given that a log often consolidates information. It follows that access to audit information must be carefully controlled.

Given that audit information is recorded within a database system, we can exploit database functionality to control and monitor access to audit information. Audit tables must be read-only. Access to audit-related tables, views and functions can be controlled using the standard database access control mechanisms. Privileges should be restricted to specific roles. Activation of these privileges and execution of audit functionality should also be monitored—to audit the auditor. Audit information can be shared through the event infrastructure, controlled through IC rules (§10.4, §10.5).

Rather than assigning privileges directly to the audit logs, views can be constructed to limit disclosure. For instance, types rather than values may be sufficient [Bak07]: it may be enough to know that a user received a particular event-type, without revealing the information contained in the instance itself. Stored procedures can provide information regarding common auditing operations, such as the recipients of information pertaining to a particular patient. Again, this can be implemented without revealing specific details. In addition to limiting data exposure, these procedures help to consolidate audit information into a more useful form. Fig. 10.7 presents some example queries within a specified time range, where stored procedures provide a view over historical information. The first example shows the recipients of information concerning a patient at the type-level, without revealing any content; the second presents the information flows for a particular publisher with respect to event types; and the third displays the prescriptions for controlled medications issued by a specific prescriber without revealing patient specifics.

Here we consider only the technical aspect; stringent (organisational) protocols must be defined and implemented to control access to audit information. Ultimately, it is a political question as to who has access to audit information. This is likely to depend on

```
transmitted_patient_information(2323232320, '2009-01-01', '2009-02-30')
```

Patient_Specific_Events		
Recipient	*Event Type*	*Count*
NHS_4126	Prescribe	5
NHS_4126	Treatment	11
NHS_499	Prescription	5

```
event_flow('NHS_993', '2009-01-03 00:00:00', '2009-01-03 23:59:59')
```

Event_Flow_View				
Publisher	*Event Type*	*Recipients[]*	*Transform Types*	*Time*
NHS_993	Prescribe	[NHS_4126, NHS_8813]	[Prescription, Drug_Audit]	2009-01-03 14:40:01
NHS_993	Treatment	[NHS_4991]		2009-01-03 14:44:07

```
controlled_prescriptions('NHS_716', '2009-04-01', '2009-04-10')
```

Drug_Audit_Events						
Event_ID	*Managing Domain*	*Time sent*	*Prescriber*	*Drug*	*Dosage*	*Date*
44412	NHS_4412	2009-04-03 11:16:24	NHS_716	Morphine	1 x 10mg	2009-04-03
44455	NHS_4412	2009-04-03 12:00:04	NHS_716	Diazepam	2 x 2mg	2009-04-03
44472	NHS_4412	2009-04-03 13:45:41	NHS_716	Morphine	1 x 25mg	2009-04-03

Figure 10.7: Views over the audit logs.

circumstance. One expects a patient to have access to their own information [UK 98], and the details of others accessing their information [NHS07b]. The Figure indicates that queries over audit logs could allow patients to visualise the data flows concerning their information. At a system level, some access to the exception logs is likely required by system administrators to resolve technical issues, e.g. to investigate the failure of a transformation function; though access should be restricted to resolving the particular issue, and consent should be obtained if appropriate.

10.4 Active Audit

The previous sections describe traditional auditing, where local data and processes are recorded within a local broker for subsequent query and analysis. *Active audit* is proactive, in the sense that it involves raising alerts when certain situations arise to allow for an appropriate response. Alerts are useful at the system level, e.g. to inform of a failed transaction (an entry to the exception log), or to warn of policy incompatibilities. At the application-level, alerts can warn when a patient's sensor is disconnected, when a carer omits a task, or when a patient's information is accessed, as well as notify of suspicious circumstances, e.g. when a surgeon requests data while officially on vacation. In addition to triggering workflows or automated responses, active audit can also bring humans into the system, enabling direct intervention in exceptional circumstances.

IC infrastructure inherently provides distributed audit capabilities. PostgreSQL-PS brokers have the ability to publish events, meaning that internal subscriptions, active rules and triggers can be used to raise events when certain situations occur. These events can propagate throughout the broker network, subject to data flow controls.

10.5 Event Replay

The pub/sub paradigm involves delivering information to those interested *as it occurs*. However, there are situations, often triggered by some incident, where clients require historical information. This might be due to a break in information flow, perhaps due to a period of disconnection; or more generally, where an incident triggers some application-level need for historical data, rendering previously 'uninteresting' events relevant.

In a distributed broker network, events from a number of sources can satisfy a subscription. The decoupled nature of the environment, along with notions of local control, makes it impractical for a client to uncover each source of historical information [LCH+07]; even then, a client may not hold sufficient privilege to directly access the datastore. By handling historical requests in the middleware, the requesting principal is not burdened with the task of discovering and directly querying every potential information source.

There is some work concerning historical events in pub/sub systems. This tends to focus on the use of history buffers and event retransmission to deal with issues of bootstrapping and disconnection in mobile systems [CFH+03, FGKZ03, BZA03]. Mühl et al. [MUHW04] also consider mobility, mentioning that event histories can be managed either by producers or brokers. Li et al. [LCH+07] take a general approach, in which databases are connected to various brokers, each associated with a filter to store particular information. The broker connected to the database holding the relevant information republishes historical events on receipt of a subscription query with a historic time-based parameter.

Previous work demonstrates the value of event replay in pub/sub, but overlooks issues of access control. Clearly, event replay mechanisms require control, where policy defines the circumstances in which historical information is released. In multi-domain environments it will often be infeasible, if not inappropriate, for a client to directly query a remote broker for historical information. Instead, historical requests and events should traverse the broker network to enable the proper enforcement of dissemination policy. Further, because access control policy is context sensitive, the content of and/or the restrictions on a historical event may differ due to a change in circumstance. For instance, an event originally subjected to a restriction filter, may now, instead, be subject to a transformation function. Replay mechanisms need not only restrict, but can also enrich information flows, e.g. where a particular event (such as an emergency) triggers the release of a set of unperturbed historical sensor readings. As all events use the same infrastructure, it is possible that the replayed events may impact on system state, e.g. when auditing the auditor. When dealing with historical events, it is necessary to consider the circumstances in which replay is appropriate, and how a change in state alters information flow.

Here, we extend IC to support event replay, exploiting the fact that brokers are database systems that maintain a log of historical events and contextual information. Our approach unifies the management of current and historical events, focusing on context and access control specifics.

10.5.1 Replay Requests

Historical information is delivered in response to a *replay request*: a query over historical information. Such a request is defined in a message, the schema for which consists of the following attributes:

Event Type The event type to be replayed.

[From/To] (timestamp) Timestamps that bound the time line of the historical query on one or both sides.

During Conditions that held at the time of the original event publication.

Filter A conditional clause acting as a filter on the event instances delivered—akin to a subscription filter.

Permission Attributes Additional information to assist in the replay authorisation decision.

Scope [global/local] Defines the scope of the replay request: local applies only to the remote broker, global is propagated throughout the broker network.

The `during` attribute acts as a filter applicable at the time of the original publication. It functions to replay historical events that were published while particular fluents held. To implement this functionality, it is necessary to have access to the values of the contextual predicates at the time of the original publication—see §10.2.

Recall that the scoping attributes were removed from PostgreSQL-PS, so that subscription propagation is controlled by the forwarding policy of the domain (§7.2.1). However, client-specified scoping is relevant for event replay as the semantics of a replay request differs—a replay request is one-off, where the client might be interested only in a particular broker's event history, as opposed to that of the whole network. Here the specification of scope is used to avoid duplicate historical notifications (§10.5.3).

A replay request must, at the very least, specify the event type to be replayed and the scope of the request; however, we expect that replay messages will typically bound the historical search through specification of time-range and/or a filter. Like subscription requests, replay requests are validated subject to authorisation policy. Permission attributes assist in this respect.

Unification of Replay Requests and Subscriptions

A replay request involves a one-off query for historical events. Thus, a replay request is essentially a subscription to past information, while a general subscription forms a query over future events. As such, subscriptions and replay requests can be unified.

It follows that replay requests take two forms. The first is an explicit replay-only request, in which the event channel persists until all matching historical events are delivered. An example replay-only scenario concerning prescriptions is discussed in §10.5.5. The second is where a replay request forms part of a general subscription request, signified through the addition of (some) replay specific attributes: `to`, `from`, `during` and `scope`. In this situation the event channel remains after historical delivery, functioning as a (general) subscription to deliver events as they occur. An example scenario concerns A&E services: in emergency situations, paramedics might be authorised to connect to the event stream. To provide some background to the situation, a paramedic might request the patient's sensor readings for the last two hours as part of their subscription request. After receiving this historical information, the subscription persists to deliver further readings as they are taken.

10.5.2 Replay Request Validation

The dissemination of historical information requires control. As a replay request is essentially an instance of a subscription, they undergo similar validation processes.

Replay authorisation rules, such as the one in Fig. 10.10(a), are defined to permit replay of a particular event type. They take a similar form to request authorisation rules (Appx. C.2), except that they do not include monitored conditions. Instead, replay authorisation rules include an attribute (`select_filter`), which allows policy administrators to place a filter on the query determining the (original) event instances selected for republication.

Conditions may be imposed on a client's replay request, adding extra predicates to the `filter` or `during` clause, or limiting the date range of the request. These constraints are set by imposed condition rules defined specifically for replay requests. Transformation rules do not require any special definition for replay scenarios. The restrictions apply to replay channels in a similar manner to subscription channels as described in Ch. 7.

10.5.3 Replay Request Processing

Brokers process a replay request in a similar manner to a subscription request, where a subscription (delivery) channel is created with the appropriate filters and transformation hooks. In this way, historical event instances are subject to the same restrictions as a freshly published event instance.

Global replay-only requests are forwarded to brokers who have, at some stage, advertised the particular event type. This information is maintained in the logs of the routing catalogues. If the remote broker is no longer connected, reconnection is attempted. If the remote broker currently does not advertise that type, a temporary advertisement is (locally) created and processed to forward the replay request. The creation of this request-specific channel enables the application of the relevant forwarding restrictions, through the same procedures described in Ch. 7.

The forwarding process where a replay request is issued as part of a subscription is more involved. If the replay scope is local, the subscription is forwarded to relevant remote brokers with the replay aspects removed. Otherwise, the complete request is forwarded to brokers advertising the event type, in the manner described in Chs. 5 and 7. Brokers that do not currently advertise the event type, but have at some point in the past (within the time-range specified by the request), are forwarded only the replay aspects of the request. This is to obtain historical information without attempting to establish a subscription. Of course, forwarding is subject to any request transformations and filters imposed on the channel. Fig. 10.8 illustrates how an example global subscription request (`s`), which includes replay aspects (`r`), is forwarded throughout the network. Brokers marked with an `a` currently advertise the event type, and thus receive the complete subscription request.

On the successful validation of a request, a broker selects the appropriate historical events it stores by querying its local event type's audit table (`pg_log_in_<type>`). This query is qualified by any filters included/imposed as part of the request, in accordance with the scope of the request. If the replay scope is global, events are replayed only if published by a directly connected client, not if forwarded through a link.[5] Locally-scoped replay

[5]Such information is determined through a join between the event log and connection histories.

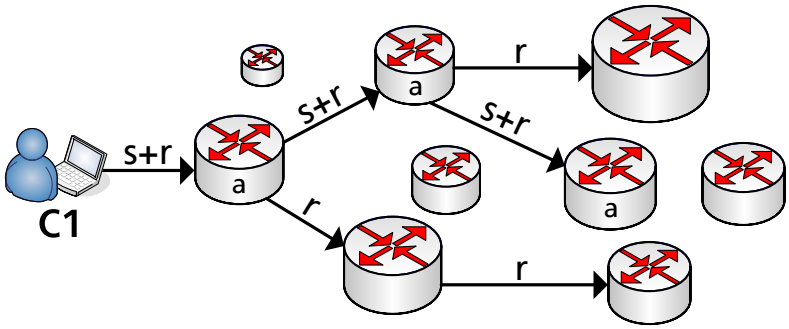


Figure 10.8: Forwarding of a global subscription request including replay.

requests, however, apply to all publications regardless. This distinction is important, as duplicates can occur if both local and remote brokers were to replay all events. The event instances are subject to the applicable transformations and filters as per the usual IC process. Event properties are used to tag an event as historical, allowing clients and brokers to distinguish between historical and current events. This includes the ID of the delivery channel to ensure that the events are only added to the `match` queue for the client that issued the historical request. Historical events also include a property containing the timestamp of the original publication.

A broker executes a replay request as a one-off query to select the relevant historical events to replay from its local store. After transmission of the events, a broker delivers a system-message to signal the completion of the replay process. A broker closes the request channel when a) all relevant local historical events have been transmitted; and b) in the case of a global request, when all the event channels created for the replay request have closed (i.e. recursive closure). If a request is issued as part of a subscription, the channel remains to operate as a general subscription.

10.5.4 Automatic Replay

We have considered the replay of events in response to a client's request. There may, however, be situations where a broker should automatically republish historical events. This might be for quality of service, e.g. to minimise loss in cases of disconnection, or in situations where a change in context interacts with the access control mechanisms to alter the flow of information. For instance, a change in context might cause a certain imposed condition rule or transformation to no longer apply, and thus prior events should be republished and delivered without restriction.

The definition of an *automatic replay* rule is essentially a combination of a request authorisation rule and a (policy-specified) replay request. An example rule is shown in Fig. 10.10(b). The `triggering_event` and `triggering_condition` attributes are used to create the active rules (internal subscriptions) that trigger the replay. The definitions for the other attributes are as previously described. Automatic replay rules are implicitly local in scope.

Automatic replay rules are loaded as part of the subscription validation process, and are evaluated in the same manner as request authorisation rules. Automatic replay rules operate much like monitor functions. On receipt of a triggering event, if the triggering conditions hold, the replay process begins: historical events are selected, according to

the replay definition, and delivered to the subscriber subject to any transformations and filters.

10.5.5 Application Examples

We illustrate the usage of event replay functionality in the following two scenarios.

Historical Event Replay: Prescribing

We have used a prescribing scenario to demonstrate how IC works to restrict information flow. The scenario involves the auditor receiving prescription information for controlled drugs with patient specifics removed. If the prescriber is under investigation, the restrictions do not apply: the auditor receives a `prescribe` event that includes patient details.

Consider a nurse whose employer contracts with many health institutions to provide care services. Each institution holds the information pertaining to their patients. It comes to light that a number of her patients have been falling ill, and that this nurse has been prescribing controlled drugs far more frequently than other nurses. As such, she is officially placed under investigation; though as she is only under suspicion she continues to practice. It is in the interests of the public and patient safety that the auditor investigate the matter. To undertake the investigation the auditor requires detailed information regarding all the nurse's prescriptions, including patient specifics and the reasons for prescribing, from all institutions for which she has worked. This data is required not only from her future prescriptions, but also those previously issued.

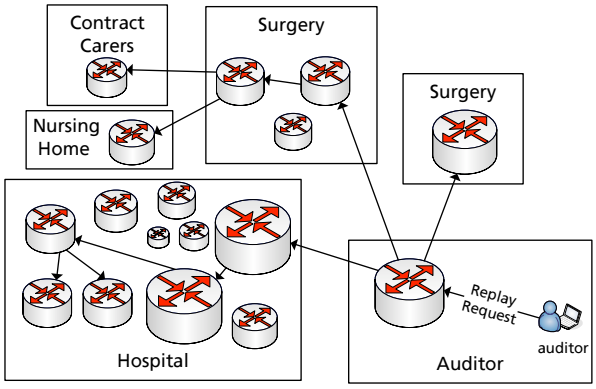


Figure 10.9: The propagation of a replay request for prior prescriptions issued by the regional auditor.

As described, an imposed condition prevents `prescribe` events from propagating to the auditor unless the prescriber is under investigation.[6] Now that the nurse is under investigation, the auditor requires information of her previous prescriptions—information that was previously restricted. To obtain this information, the auditor issues a replay request for all prescriptions issued by the nurse.

As shown in Fig. 10.9, the replay request propagates through the broker network to those brokers currently and/or previously advertising `prescribe` events. This causes each broker receiving the request to republish the `prescribe` events issued by the nurse.[7]

[6] Again, we assume a fluent that represents consent for information to flow to the auditor.

[7] Although policy is broker specific, we assume that domains will enact similar policies given the nature of the scenario.

The authorisation rule in Fig. 10.10(a) allows the replay of `prescribe` events, if the requestor is an auditor and the request targets a specific prescriber. As the nurse is under investigation, the replayed events satisfy the filter and thus pass through the event channel. This provides the auditor with detailed information concerning the nurse's prescribing history.

```
<replay_authorisation>
  <rule_name>auditorreplay</rule_name>
  <event_type>prescribe</event_type>
  <credentials>NHSCred(usernm, 'drug_auditor')</credentials>
  <permission_attributes>prescriber_id:varchar(10)</permission_attributes>
  <condition />
  <select_filter>prescribe.prescriber_id=att.prescriber_id</select_filter>
  <notes>Allows an auditor to request historical prescribe events,
    but they must target a specific prescriber</notes>
</replay_authorisation>
```

(a) The replay authorisation and restriction rules concerning `prescribe` events.

```
<automatic_replay>
  <rule_name>replaysensoremergency</rule_name>
  <event_type>sensor_reading</event_type>
  <credentials />
  <condition />
  <permission_attributes>patient_id:int8</permission_attributes>
  <from>now() - interval '2 hours' </from>
  <to>now()</now>
  <during />
  <select_filter>sensor_reading.patient_id=att.patient_id</select_filter>
  <triggering_event>emergency</trigger_event>
  <triggering_condition>emergency.patient_id = att.patient_id</trigger_condition>
  <notes>Any subscription to the sensor stream will receive 2hrs of previous
     readings when the patient enters a perceived emergency situation</notes>
</automatic_replay>
```

(b) The automatic replay rule for emergency situations.

Figure 10.10: Policy fragments driving the replay scenario.

Sensor Obfuscation

In the previous chapter, we described a homecare scenario where sensor data is perturbed for reasons of privacy. In sensor environments, transformations might involve altering attribute values, perhaps transforming location co-ordinates into regions, as opposed to providing the precise GPS or room co-ordinates; or summarising physiological information, e.g. bucketing respiratory readings into a 'dashboard' representation: [Stable, Concern or Emergency]. In emergency situations, however, it is important that care staff receive unperturbed events.

Given the importance of an emergency situation, a domain might decide to automatically replay sensor events from several hours before the detection of the incident, to provide background information. Such functionality is effected through the automatic replay rule of Fig. 10.10(b), which attaches to `sensor_reading` subscriptions to trigger an unperturbed (the transformation no longer applies) stream of historical events on detection of an emergency. This provides care staff with more details of the situation, to assist emergency response.

10.5.6 Further Work

Our work in event replay provides a base for further exploration into historical event republication. Clearly, republication rates and the effects on system load are important concerns [MUHW04], as replaying historical information involves the propagation of a number of events at once. While an area for future work, it may be possible to use fluents and IC rules for performance considerations, e.g. to control the rate of data transmission.

Overheads aside, the dynamic nature of the network brings complications, e.g. it is not possible to guarantee that all potential sources were queried if a prior routing path is no longer accessible. Investigation is required into the consequences of dynamically establishing links and event channels to provide replay functionality. Further, there are issues concerning the storage of historical data—it is unrealistic to assume that a broker perpetually stores all historical information. The effect on request semantics must be considered where historical events are no longer accessible by a broker. One approach might entail brokers producing *meta-events* to inform clients of any limitations in realising the request, such as a reduced date range due to archiving, or that a broker is no longer accessible.

Events are replayed for particular time ranges, optionally refined by the values of fluents. Both the requestor and policy author can filter the event stream. While flexible, scenarios are required to determine whether request semantics are sufficiently/overly expressive.

Given that brokers are databases, our investigation into event replay is a natural consequence of IC's auditing capabilities. As shown by the examples, replay functionality is intuitively useful for healthcare. However, further investigation is required into the use of event replay mechanisms in a real-world healthcare deployment. Consideration must be given to the types of information requested, the volume of data returned by the historical queries, the frequency of replay requests, a domain's storage requirements, and the associated impacts on overall system performance. Analysis is required into the dynamism of the environment (i.e. broker/policy/state churn), accounting for the frequency of change, to determine the impact on, and effectiveness of, event replay mechanisms.

10.6 Summary

Audit is a fundamental requirement for health information. An integrated database-pub/sub architecture simplifies auditing procedures. Message processing involves the use of queues, which audit mechanisms extend to persist data long-term. Given that IC rules are context-sensitive, both the data transmitted and any changes in context must be recorded. The active rules managing state can also maintain contextual history. Alternatively, such information can be obtained by directly querying the event logs. This provides visibility of system state at a particular time in the past. Audit information is useful not only for purposes of accountability, e.g. providing evidence for investigations, but can also assist in the detection of policy errors.

Active audit is facilitated by an event-based infrastructure, where events alert in situations of concern. We describe event replay mechanisms to unify the delivery and control of both past and current events. Event replay appears a useful feature of a pub/sub infrastructure, particularly in situations where there are multiple information sources and/or where changes in context affect data visibility. The infrastructure presented demonstrates mechanisms to control event republication, describing how context can be exploited by policy administrators to manage data disclosure.

11

Conclusions and Further Work

Health providers are responsible for protecting the confidentiality of personal health information. At the same time, healthcare information must be shared, where (aspects of) an incident may be relevant to a number of interested clients in various administrative domains. As such, there is a responsibility to ensure that information is shared appropriately. Healthcare is highly data-driven, involving interactions between patients, professionals, organisations, government, sensors, etc. Wide-scale event distribution paradigms, while effective for data dissemination, generally lack the rigorous disclosure controls required by health infrastructure. IC provides the means for dealing with these competing concerns, by introducing point-to-point, context-sensitive policy rules that are enforced by brokers to control information flows throughout a distributed pub/sub infrastructure.

The IC database-pub/sub integration results in an efficient connection between the substrates; facilitates persistence, replay and audit; and allows policy rule definitions access to a rich representation of state. Our implementation leverages much from the database system, in terms of languages and transactions, and through the extension of active rules. Given that health systems depend on the use of database systems, IC introduces a layer above technology already commonplace in NHS infrastructure.

Recent debate concerns the use of centralised storage and security infrastructure in healthcare. Our focus is on supporting the heterogeneous nature of the health service—where information flows across administrative domains.[1] The NHS aims to give a greater degree of freedom and control to service providers. It is argued that only standards and basic services (e.g. identity management) should be centrally managed, allowing local control over data and privilege. Given that providers hold and require information relevant to their service, it is natural that they manage the associated sharing concerns. With control comes responsibility. Our approach allows autonomous management to extend to the protection of health information, giving those responsible for data fine-grained control over the circumstances for its release. Brokers enforce local policy consistently across

[1] Although this work is presented in the context of supporting environments of multiple, autonomous administrative domains, it also applies to more centralised architectures (§8.5).

all connections. This gives a domain the ability to control information flows within and to/from the infrastructure it manages. By sharing information appropriately, providers meet their data management obligations: the recipient becomes responsible for the information. Federated environments are not only scalable, but mitigate against risks to confidentiality. Accountability is improved, by providing visibility as to those responsible for information misuse/mismanagement.

This dissertation presents IC: a layer of control over a pub/sub service to manage the distribution of sensitive information. Its goal is to provide an infrastructure appropriate for an environment of federated administrative control, to allow those responsible for information to meet their data management obligations. IC enables precise control over the content of, and circumstances for, event and request transmission, while facilitating data management and storage.

11.1 Further Work

This work raises a number of research issues:

Workflow Integration Many health procedures are workflow (*pathway*) oriented. Workflows are suited to an event-based infrastructure, as they are responsive to incidents. One area for investigation concerns the interplay of care pathways, such as those of [MoM], and information disclosure. While IC provides context-aware access controls, there is scope to tightly couple workflow procedures with privilege allocation. There exists some general work in area of workflow based access controls [RDD08, AC06], though these do not directly consider information dissemination. IC rules are evaluated given current contextual state. Health environments are complex, frequently involving task fragmentation and variance from standard procedures [SGI04]. Workflows involve enumerating and prescribing a particular sequence of states, and thus tend not to handle (unexpected) exceptions. As such, it remains an open question as to whether workflows are suitable for an access control regime, given the dynamic and complex nature of health processes.

Sticky Policies In IC, brokers enforce locally defined policy against directly connected clients/brokers. An interesting area for research concerns the attachment of policy to events and requests for enforcement in other parts of the network. Sticky policies, however, raise enforcement issues: concerning the capability of brokers to locally enforce remotely specified policies; concerning trust, whether remote brokers can be trusted to enforce the policy of another; and verification, whether it is possible to verify policy adherence outside of the domain of control. It is arguable that these concerns are mitigated in healthcare, given the overarching responsibility for health information. However, sticky policies raise questions of accountability. For example, who is liable for an ill-specified, under-specified, ambiguous or misinterpreted policy, the policy author and/or the enforcer(s)? If an attached (remote) policy conflicts with local policy, which takes preference? Can the policies be combined? If so, is this appropriate? It is important that such concerns can be reconciled with the notion of local responsibility. The *SmartFlow* project [CIC09] is exploring event tagging to control visibility within distributed infrastructure [PBE+09].

Event Replay As discussed in §10.5.6, there is scope for further work in the area of event replay. A real healthcare deployment is needed to evolve the approach, to investigate the use and requirements of such functionality, and to consider issues concerning storage, context and connectivity.

Event Composition Composite events not only react to, but also impact on system state, and thus can affect information flows. In this dissertation we have used fluents to encapsulate meaningful context, providing some examples of where fluents are sensitive to patterns of events. However, we did not explicitly consider the details of composite events, such as distributed detection or the integration of composition languages to provide a particular level of expressiveness. IC provides an attractive model for the integration of composite event functionality, as hook rules allow events to be consumed, processed and created by the brokers themselves. Research into composite event functionality has origins in the database world [CM94]. As brokers are also databases, event composition mechanisms have access to rich state and can exploit the inherent data management/processing capabilities of the underlying database system, which in turn feeds back into the messaging infrastructure.

Database Query Control This dissertation focuses on controlling event flows in a distributed infrastructure, through integrated database-pub/sub brokers. We do not, however, consider the control of SQL queries issued by local clients directly against the database system.[2] General database security is increasing in flexibility; for instance, PostgreSQL 8.4 [PGDG09a] allows privileges to be defined with field-level granularity. Hippocratic databases [IBM09a] enable control over the data disclosed by local queries, through query re-writing and by using statistical measures to restrict results. An area for research concerns the unification of distribution and query control mechanisms, subjecting SQL queries and results to similar filtering and transformation mechanisms as described for IC (and/or vice-versa). That said, given the stringent management requirements for health information, it may be preferable that clients issue queries through a middleware, rather than executing SQL queries directly against a database.

Formalisation The purpose of formalisation is to verify correctness, thus facilitating the detection of errors. However, the ability to reason about behaviour may come at the cost of the model's expressiveness. From a systems perspective, both PostgreSQL-PS and IC would benefit from rigorous formalisation. The validation of system behaviour for global healthcare infrastructure is challenging, considering its size and complexity. However, given that healthcare is an environment of federated control, it seems reasonable to analyse policy per administrative domain.

In this dissertation we describe the use of fluents to represent context. This was motivated by the fact that fluents provide a means by which to reason about policy. There is some work in using Deontic Logic and an Extended Event Calculus to monitor data obligations [EE08, RPG05] and for policy validation [BLR03]. Recording changes in fluent state and the events causing change serves as an evidence base for detecting failures in policy, and situations of non-compliance.

[2] Our work on audit and event replay touches on the query aspect.

Real-world Deployment The focus of this work is on data protection. The aim is to provide an alternative to centralised data stores and policy definitions, through a methodology for controlling information dissemination in an environment of federated administrative control. IC was developed with regard to requirements derived from legislation, codes of practice, research papers, executive statements, NHS (NPfIT) specifications, nursing manuals and other medical documents. Thus, IC is grounded on health policy instruments—which itself is a moving target.

A key area for future work is to consider the operational concerns of IC. That is, evaluating the run-time usage of the system in a live health deployment, considering real clinical data[3] and workloads. There are two main areas for further examination:

Scalability Pub/sub is generally well-regarded for its scalability. The results presented in this dissertation indicate that IC transformations can in some circumstances improve routing efficiency. However, it is important to measure load and performance based on real workloads. Data is required regarding an actual deployment: considering the number of clients, event types and policy rules, event load and sizes (e.g. text or X-Ray images), rule complexity, connection churn, required response times, and other system/environmental variables and constraints. Such information will allow for quantified statements of scalability, which in turn feeds back into the research and development process.

Policy Management IC rules are deliberately expressive, enabling authors to address a wide-range of healthcare issues. However, expressiveness increases complexity. The deployment of IC will provide visibility as to the number and type of rule definitions, enabling evaluation of the policy model. It may be that in practice policies are relatively simple, in which case the model can be constrained. This eases issues of policy definition, coverage and analysis, and aids formalisation.

System usability must also be investigated to consider the ease in which end-users (e.g. doctors, patients) can effect data control policy. This will often concern the definition of consent preferences. General restrictions can be encapsulated in fluent state, e.g. disclosure preferences concerning data flows to a research organisation or a centralised record service; however, specific concerns entailing granular or patient-specific restrictions might require rule definition. Familiar *wizard*-like interfaces, similar to those presented in Ch. 9, can guide users through policy authoring processes, manipulating policy definitions (data) as appropriate. Although this is a design issue, the ease of policy authoring/management will certainly impact on user-acceptance.

These issues are best explored with real-world experimentation. Clearly, an actual national-level deployment requires significant resources—health infrastructure contracts are in the order of hundreds of millions pounds, requiring consultation with patients, clinicians, service providers and the government [Bre05]. At the time of writing, we are improving the efficiency and robustness of our proof-of-concept IC implementation for use in a project entitled *Personal and Social Communication Services for Health and Lifestyle Monitoring* [CEBE].

[3] At least in terms of type, size and volume.

11.2 Concluding Remarks

The protection of health information is not just a technical issue. There are legal and social pressures regarding the use and management of health data. Infrastructure supporting health services must facilitate (real-time) information sharing, storage and protection, in line with higher-level concerns. IC builds on the inherent responsibility for health information, introducing a layer of control over a distributed pub/sub network to manage wide-scale event distribution. Unlike most (proposed) health infrastructure, IC is designed to account for the federated nature of healthcare policy. It is not a *panacea*, but instead is intended as a control mechanism, giving those managing data the ability to meet their responsibilities. Information protection is improved—responsibility brings accountability. We feel that the concepts described in this dissertation provide a realistic foundation for managing the heterogeneous nature of current and future health services.

Bibliography

[ABD+09] Ross Anderson, Ian Brown, Terri Dowti, Philip Inglesant, William Heath, and Angela Sasse. *Database State*. The Joseph Rowntree Reform Trust Limited, UK, 2009.

[AC06] Raman Adaikkalavan and Sharma Chakravarthy. How to use events and rules for supporting role-based security? (invited paper). In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 698–702, Washington, DC, USA, 2006. IEEE Computer Society.

[AE07] Alan S. Abrahams and David M. Eyers. Using annotated policy documents as a user interface for process management. In *Proceedings of the First International Workshop on Knowledge-based User Interfaces (KUI07)*, page 64, Athens, Greece, June 2007. IEEE Computer Society.

[AEKV07] Rakesh Agrawal, Alexandre Evfimievski, Jerry Kiernan, and Raja Velu. Auditing disclosure by relevance ranking. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 79–90, New York, NY, USA, 2007. ACM.

[AKSX02] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases*, pages 143–154, 2002.

[And96] Ross J. Anderson. A security policy model for clinical information systems. In *IEEE Symposium on Security and Privacy*, pages 30–43, 1996.

[ANSI92] American National Standards Institute. Standard x3.135-1992, 1992.

[AoMS06] Academy of Medical Sciences. Personal data for public good: using health information in medical research. `http://www.acmedsci.ac.uk/download.php?file=/images/project/Personal.pdf`, 2006.

[Apa08] Apache Software Foundation. ActiveMQ. `http://activemq.apache.org/`, 2008.

[BA05] Roberto J. Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 217–228, Washington, DC, USA, 2005. IEEE Computer Society.

[Bak04] A. Bakker. Access to EHR and access control at a moment in the past: a discussion of the need and an exploration of the consequences. *International Journal of Medical Informatics*, 73(3):267–270, 2004.

[Bak07] A. Bakker. The need to know the history of the use of digital patient data, in particular the EHR. *International Journal of Medical Informatics*, 76(5–6):438–441, 2007.

[BBHM95] Jean Bacon, John Bates, Richard Hayton, and Ken Moody. Using Events to Build Distributed Applications. In *Proceedings of the 2nd International Workshop on Services in Distributed and Networked Environments (SDNE '95)*, page 148. IEEE Computer Society, 1995.

[BCSS99] Guruduth Banavar, Tushar Deepak Chandra, Robert E. Strom, and Daniel C. Sturman. A case for message oriented middleware. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 1–18, London, UK, 1999. Springer-Verlag.

[Bec05] Moritz Y. Becker. *Cassandra: Flexible trust management and its application to electronic health records*. PhD thesis, University of Cambridge, 2005.

[Bec07] Moritz Y. Becker. Information Governance in NHS's NPfIT: A Case for Policy Specification. *International Journal of Medical Informatics*, 76:432–437, 2007.

[BEMP05] Jean Bacon, David Eyers, Ken Moody, and Lauri Pesonen. Securing publish/subscribe for multi-domain systems. In *Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, pages 1–20, 2005.

[BEP+03] András Belokosztolszki, David M. Eyers, Peter R. Pietzuch, Jean Bacon, and Ken Moody. Role-based access control for publish/subscribe middleware architectures. In *DEBS '03: Proceedings of the 2nd International Workshop on Distributed Event-Based Systems*, pages 1–8, New York, NY, USA, 2003. ACM.

[Ber08] Berkeley Database Research. TelegraphCQ. `http://telegraph.cs.berkeley.edu/telegraphcq/v0.2/`, 2008.

[BES+09] Jean Bacon, David M. Eyers, Jatinder Singh, Brian Shand, Matteo Migliavacca, and Peter Pietzuch. Securing event-based systems. *Information Technology. Special issue on Complex Event Processing*, 2009.

[BESP08] Jean Bacon, David M. Eyers, Jatinder Singh, and Peter R. Pietzuch. Access control in publish/subscribe systems. In *DEBS '08: Proceedings of the Second International Conference on Distributed Event-Based Systems*, pages 23–34, New York, NY, USA, 2008. ACM.

[Bha09] Neil Bhatia. Summary care records—No one has asked me. *British Medical Journal*, 338, June 2009. b2519.

[BHM+01] Jean Bacon, Alexis Hombrecher, Chaoying Ma, Ken Moody, and Walt Yao. Event storage and federation using ODMG. In *POS-9: Revised Papers from the 9th International Workshop on Persistent Object Systems*, pages 265–281, London, UK, 2001. Springer-Verlag.

[Bib77] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, MITRE Corporation, Apr 1977.

[BLR03] Arosha K. Bandara, Emil Lupu, and Alessandra Russo. Using event calculus to formalise policy specification and analysis. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, pages 26–40, 2003.

[BMA07a] British Medical Association. Confidentiality and disclosure of information to PCTs in primary care settings — Guidance for GPs. `http://www.bma.org.uk/ethics/confidentiality/confiddiscloseinfopcts.jsp`, 2007.

[BMA07b] British Medical Association. Electronic health records will fail unless public and professional confidence is restored, says BMA, 13 Sep 2007.

[BN89] D. F. C. Brewer and M. J. Nash. The Chinese wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[BOS+06] Mike Bilger, Luke O'Connor, Matthias Schunter, Morton Swimmer, and Nev Zunic. Data-centric security. Technical report, IBM, NY, 2006.

[BP73] David E. Bell and Leonard J. La Padula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corporation, May 1973.

[Bre05] Sean Brennan. *The NHS IT Project: The Biggest Computer Programme in the World... Ever!* Radcliffe Publishing Ltd, 2005.

[BRG00] Bolton Research Group. Patients' knowledge and expectations of confidentiality in primary health care: a quantitative study. *British Journal of General Practice*, 50:901–902, 2000.

[BSIG09] Bluetooth Special Interest Group. Bluetooth specification documents. `http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/`, 2009.

[BSNM07] Gaetano Borriello, Vince Stanford, Chandra Narayanaswami, and Walter Menning. Guest editors' introduction: Pervasive computing in healthcare. *IEEE Pervasive Computing*, 6(1):17–19, 2007.

[BV06] R. Baldoni and A. Virgillito. Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey (revised version). Technical report, MIDLAB 1/2006 - Dipartimento di Informatica e Sistemistica A.Ruberti, Universita di Roma la Sapienza, 2006.

[BZA03] Sumeer Bhola, Yuanyuan Zhao, and Joshua Auerbach. Scalably supporting durable subscriptions in a publish/subscribe system. *DSN '03: International Conference on Dependable Systems and Networks*, pages 57–66, 2003.

[Cab05] Cabinet Office (UK) - eGovernment Unit. e-Government Interoperability Framework Version 6.1. `http://www.govtalk.gov.uk/schemasstandards/egif_document.asp?docnum=949`, 2005.

[CABB04] Mariano Cilia, Mario Antollini, Christof Bornhö)vd, and Alejandro Buchmann. Dealing with heterogeneous data in pub/sub systems: The Concept-Based approach. In *DEBS '04: Proceedings of the 3rd International Workshop on Distributed Event-Based Systems*, Edinburgh, Scotland, May 2004.

[CB95] D Carman and N Britten. Confidentiality of medical records: the patient's perspective. *British Journal of General Practice*, 45(398):485–488, 1995.

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*, pages 379–390. ACM, 2000.

[CEBE] University of Cambridge, University of Essex, BT, and Ericsson. PAL:Personal and Social Communication Services for Health and Lifestyle Monitoring. `http://pal.dalore.net`.

[CFGR02] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. *The VLDB Journal*, 11(4):354–379, 2002.

[CFH+03] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A. P. Buchmann. Looking into the past: enhancing mobile publish/subscribe middleware. In *DEBS '03: Proceedings of the 2nd International Workshop on Distributed Event-Based Systems*, pages 1–8, 2003.

[Cha95] Sharma Chakravarthy. Early active database efforts: A capsule summary. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):1008–1010, 1995.

[Cha06] Ritu Chadha. A cautionary note about policy conflict resolution. In *Military Communications Conference (MILCOM))*, pages 1–8, October 2006.

[CHI] Canada Health Infoway. `http://www.infoway-inforoute.ca`.

[CHY96] Ming-Syan Chen, Jiawei Hun, and Philip S. Yu. Data mining: An overview from database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8:866–883, 1996.

[Cic90] Aaron V Cicourel. The integration of distributed knowledge in collaborative medical diagnosis. *Intellectual teamwork: social and technological foundations of cooperative work book contents*, pages 221–242, 1990.

[CIC09] University of Cambridge, Imperial College, and Clinical and Biomedical Computing Unit, National Health Service. Smartflow: Extendable event-based middleware. `http://www.smartflow.org/`, 2009.

[CL08] David W. Chadwick and Stijn F. Lievens. Enforcing "sticky" security policies throughout a distributed application. In *MidSec '08: Proceedings of the 2008 Workshop on Middleware Security*, pages 1–6, New York, NY, USA, 2008. ACM.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[CM94] S. Chakravarthy and D. Mishra. Snoop: an expressive event specification language for active databases. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):1–26, 1994.

[CM96] Chris Clifton and Don Marks. Security and privacy implications of data mining. In *In ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 15–19, 1996.

[CNF01] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.

[Cod83] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 26(1):64–69, 1983.

[Com06a] Douglas E. Comer. *Internetworking with TCP/IP Vol I. Principles, Protocols, and Architecture, 5th edition*. Prentice Hall, 2006.

[Com06b] OASIS Web Services Notification (WSN) Technical Committee. Web Services Notification (WSN) Standards v1.3. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn`, 2006.

[Con09] Conservatives (UK). Conservative Party Response to the Independent Review of NHS IT. `http://www.conservatives.com/~/media/Files/Downloadable%20Files/Conservative%20Response%20NHS%20IT.ashx?dl=true`, Aug 2009.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[CRW99] Antonio Carzaniga, David R. Rosenblum, and Alexander L. Wolf. Challenges for Distributed Event Services: Scalability vs. Expressiveness. In *Engineering Distributed Objects '99*, Los Angeles, California, May 1999.

[CRW01] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[CU07] House of Commons UK. The Electronic Patient Record: HC 422-I, Sixth Report of Session 2006-07, September 2007.

[CW91] Stefano Ceri and Jennifer Widom. Deriving Production Rules for Incremental View Maintenance. In *VLDB '91: Proceedings of the 17th International Conference on Very Large Data Bases*, pages 577–589, 1991.

[CZO+08] David Chadwick, Gansen Zhao, Sassa Otenko, Romain Laborde, Linying Su, and Tuan Anh Nguyen. PERMIS: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience*, 20(11):1341–1357, 2008.

[DA99] T. Dierks and C. Allen. *The TLS Protocol (RFC 2246)*. Internet Engineering Task Force (IETF), 1999.

[Dar08] Lord Darzi. *High quality care for all: NHS Next Stage Review*. Department of Health, 2008.

[Dat04] C. J. Date. *Introduction to Database Systems, 8th edition*. Addison Wesley, 2004.

[Dav07] Linda Davidson. After the security storm. `http://www.e-health-insider.com/Features/item.cfm?docID=228`, 12 Dec 2007. eHealth Insider.

[Dee89] Steve Deering. *Host Extensions for IP Multicasting (RFC 1112)*. Internet Engineering Task Force (IETF), 1989.

[DF03] Yanlei Diao and Michael Franklin. Query Processing for High-Volume XML Message Brokering. In *VLDB '03: Proceedings of the 29th International Conference on Very Large Data Bases*, pages 261–272, 2003.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DoH03a] Department of Health. Confidentiality: NHS Code of Practice. `http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_4069253`, Nov 2003.

[DoH03b] Department of Health. Primary care trust configurations. http://www.glospct.nhs.uk/pdf/professionals/procurement/map.pdf, Nov 2003.

[DoH04a] Department of Health. Chronic disease management: a compendium of information. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_062820, May 2004.

[DoH04b] Department of Health. Improving Chronic Disease Management. http://www.dh.gov.uk/dr_consum_dh/idcplg?IdcService=GET_FILE&dID=9015&Rendition=Web, 2004.

[DoH05a] Department of Health. A short guide to foundation trusts. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_4126013, Nov 2005.

[DoH05b] Department of Health. Supporting People with Long Term Conditions. A NHS and Social Care Model to support local innovation and integration. http://www.bjhc.co.uk/telecare/docs/SupportingPeopleWithLongTermConditions.pdf, 2005.

[DoH07a] Department of Health. Information Security Management: NHS Code of Practice. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_074142, Apr 2007.

[DoH07b] Department of Health. NHS information governance guidance on legal and professional obligations. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_079616, Oct 2007.

[DoH07c] Department of Health. Safer management of Controlled Drugs. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_074513, 2007.

[DoH08] Department of Health. Pharmacy in England: building on strengths - delivering the future. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_083815, April 2008.

[DoH09a] Department of Health. Categorisation of arm's length bodies. http://www.dh.gov.uk/en/Aboutus/OrganisationsthatworkwithDH/Armslengthbodies/Categorisationofarmslengthbodies/DH_063474, April 2009.

[DoH09b] Department of Health. The NHS Constitution for England (interactive version). http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_093419, Jan 2009.

[DoH09c] Department of Health. The statement of NHS accountability for England. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_093422, Jan 2009.

[DoHD05] Older People Department of Health and Disability Division. Building Telecare in England. http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_4115303, 2005.

[DoHHS] U.S. Department of Health & Human Services. The Office of the National Coordinator for Health Information Technology. http://healthit.hhs.gov.

[EC07a] European Commission. Accelerating the development of the eHealth market in Europe, Dec 2007.

[EC07b] European Commission. Personal health systems (conference report). `http://ec.europa.eu/information_society/newsroom/cf/document.cfm?action=display&doc_id=323`, 2007.

[EC08] Commission of the European Communities. Communication on telemedicine for the benefit of patients, healthcare systems and society (com(2008) 689). `http://ec.europa.eu/information_society/activities/health/policy/telemedicine/index_en.htm`, November 2008.

[Edi09] Editors. Healthcare data breaches burgeon in wake of new laws. *Computer Fraud & Security*, 7:2–3, 2009.

[EE08] David Evans and David M. Eyers. Deontic logic for modelling data flow and use compliance. In *MPAC '08: Proceedings of the 6th International Workshop on Middleware for Pervasive and Ad-hoc Computing*, pages 19–24, New York, NY, USA, 2008. ACM.

[EFGK03] Patrick Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[EGS03] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS '03: Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 211–222, New York, NY, USA, 2003. ACM.

[eHe05] eHealth Insider. No sealed envelopes for first summary records. `http://www.e-health-insider.com/news/item.cfm?ID=1562`, Nov 2005.

[eHe07a] eHealth Insider. Foundation trusts tender outside NPfIT. `http://www.e-health-insider.com/News/3091/south_edges_towards_'best_of_breed'_approach`, Oct 2007.

[eHe07b] eHealth Insider. South Warwickshire authorises shared smartcard use. `http://www.e-health-insider.com/news/item.cfm?ID=2449`, Jan 2007.

[eHe08a] eHealth Insider. Foundation trusts tender outside NPfIT. `http://www.e-health-insider.com/News/4077/foundation_trusts_tender_outside_npfit`, Aug 2008.

[eHe08b] eHealth Insider. Medics sceptical about government data security, 01 Feb 2008.

[eHe09] eHealth Insider. CfH jigsaw still 'missing pieces'. `http://www.e-health-insider.com/News/4930/cfhjigsawstillmissingpieces`, Jun 2009.

[EN04] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 4th Edition*. Addison Wesley, 2004.

[EPTS08] Event Processing Technical Society. Event processing glossary - version 1.1. `http://complexevents.com/wp-content/uploads/2008/08/epts-glossary-v11.pdf`, 2008.

[Eug01] Patrick Th. Eugster. *Type-Based Publish/Subscribe*. PhD thesis, EPFL, 2001.

[FfIPR05] Foundation for Information Policy Research. NHS Confidentiality Consultation - FIPR Response. `http://www.cl.cam.ac.uk/~rja14/fiprmedconf.html`, 2005. Press Release.

[FGKZ03] Ludger Fiege, Felix C. Gärtner, Oliver Kasten, and Andreas Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 103–122, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[Fie04] Ludger Fiege. *Visibility in Event-Based Systems*. PhD thesis, Technical University of Darmstadt, 2004.

[FJLM05] E. Fidler, Hans-Arno Jacobsen, Guoli Li, and Serge Mankovski. The padres distributed publish/subscribe system. In *Feature Interactions in Telecommunications and Software Systems VIII*, pages 12–30, 2005.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[FMMB02] Ludger Fiege, Mira Mezini, Gero Mühl, and Alejandro P. Buchmann. Engineering event-based systems with scopes. In *ECOOP*, pages 309–333, 2002.

[FZB+04] Ludger Fiege, Andreas Zeidler, Alejandro Buchmann, Roger Kilian-Kehr, and Gero Mhl. Security aspects in publish/subscribe systems. In *DEBS '04: Third International Workshop on Distributed Event-Based Systems*, pages 44–49. IEEE, 2004.

[GIP] GIP-DMP (Dossier Medical Personal). `http://www.d-m-p.org/`.

[GMCU06] General Medical Council UK. Good Medical Practice, 2006.

[Gol98] J Goldman. Protecting privacy to improve health care. *Health Affairs*, 17(6):47–60, 1998.

[GSB+08] Trisha Greenhalgh, Katja Stramer, Tanja Bratan, Emma Byrne, Jill Russell, Yara Mohammad, Gary Wood, and Susan Hinder. Summary Care Record Early Adopter Programme: An independent evaluation by University College London. `http://www.haps.bham.ac.uk/publichealth/cfhep/002.shtml`, Apr 2008.

[HHS+02] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. *Wireless Networks*, 8(2-3):187–197, 2002.

[HIA06] Office of the CIO: Health Information Architecture Office. Security audit requirements: Functional allocation. `http://hssp-security.wikispaces.com/PASS_Audit`, Feb 2006.

[Hid09] Hidalgo Ltd. Equivital. `http://www.equivital.co.uk/contact.html`, 2009.

[HMF04] Connecting for Health (Markle Foundation). *Achieving Electronic Connectivity in Healthcare: A Preliminary Roadmap from the Nations Public and Private-Sector Healthcare Leaders*. Markle Foundation, New York, NY, USA, 2004.

[HPS07] Jennifer Hursteld, Urvashi Parashar, and Kerry Schoeld. The costs and benets of independent living, 2007.

[HR00] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about computer systems (first edition).* Cambridge University Press, 2000.

[HSH+09] Glyn Hayes, Ian Shepherd, Richard Humphries, Gail Beer, Jonathan Asbridge, and John Williams. *Independent Review of NHS and Social Care IT.* Conservatives (UK), Aug 2009.

[HW04] Junzhe Hu and Alfred C Weaver. Dynamic, context-aware access control for distributed healthcare applications. In *Workshop on Pervasive Security, Privacy, and Trust (PSPT04)*, 2004.

[IBM08] IBM. DB2. `http://www-01.ibm.com/software/data/db2/`, 2008.

[IBM09a] IBM. Hippocratic databases. `http://www.almaden.ibm.com/cs/projects/iis/hdb/hdb_projects.shtml`, 2009.

[IBM09b] IBM. IBM Websphere MQ. `http://www.ibm.com/software/integration/wmq/`, 2009.

[IHTS] International Health Terminology Standards Development Organisation. SNOWMED CT. `http://www.ihtsdo.org/snomed-ct/`.

[Jar84] Matthias Jarke. Common Subexpression Isolation in Multiple Query Optimization. In *Query Processing in Database Systems*, pages 191–205. Springer, 1984.

[JBP05] Joint Formulary Committee (Great Britain), British Medical Association, and Pharmaceutical Society of Great Britain. British National Formulary 49th Edition, 2005.

[Khu05] Himanshu Khurana. Scalable security and accounting services for content-based publish/subscribe systems. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied Computing*, pages 801–807, New York, NY, USA, 2005. ACM.

[KSW02] Günter Karjoth, Matthias Schunter, and Michael Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. In *Privacy Enhancing Technologies*, pages 69–84, 2002.

[LAE+04] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 108–119, 2004.

[Lam71] Butler Lampson. Protection. In *Proceedings of the Fifth Annual Princeton Conference on Information Sciences and Systems,*, pages 437–443, Princeton University, 1971.

[Las] Louis Lasagna. Hippocratic Oath — modern version. `http://www.pbs.org/wgbh/nova/doctors/oath_modern.html`.

[LCH+07] G. Li, A. Cheung, Sh. Hou, S. Hu, V. Muthusamy, R. Sherafat, A. Wun, H.-A. Jacobsen, and S. Manovski. Historic data access in publish/subscribe. In *Distributed Event Based Systems*, pages 80–84, 2007.

[Les99] Lawrence Lessig. *Code and Other Laws of Cyberspace*. Basic Books, Inc., New York, NY, USA, 1999.

[Lev84] Henry M. Levy. *Capability-Based Computer Systems*. Butterworth-Heinemann, Newton, MA, USA, 1984.

[LKHT09] Helma van der Linden, Dipak Kalra, Arie Hasman, and Jan Talmon. Inter-organizational future proof ehr systems: A review of the security and privacy related issues. *International Journal of Medical Informatics*, 78:141–160, 2009.

[LLN03] Jim Longstaff, Mike Lockyer, and John Nicholas. The Tees confidentiality model: an authorisation model for identities and roles. In *SACMAT '03: Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, pages 125–133, New York, NY, USA, 2003. ACM.

[LPT99] Ling Liu, Calton Pu, and Wei Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[Luc02] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, 2002.

[McC07] Lindsay McClure. Evidence submitted by the Association of Independent Multiple Pharmacies, the Company Chemists Association, the National Pharmacy Association and the Pharmaceutical Services Negotiating Committee (EPR 22). `http://www.publications.parliament.uk/pa/cm200607/cmselect/cmhealth/422/422we08.htm`, March 2007.

[McK] McKesson Information Solutions — NHS Staff Record Project. Electronic Staff Record. `http://www.esrsolution.co.uk/`.

[McN00] Gloria J. McNeal. *AACN Guide to Acute Care Procedures in the Home*. Lippincott Williams & Wilkins, Philadelphia, PA, 19106, 2000.

[MFGB02] Gero Mühl, L. Fiege, F. C. Gartner, and A. Buchmann. Evaluating Advanced Routing Algorithms for Content-Based Publish/Subscribe Systems. In *Proceedings of the 10th International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'02)*, pages 167–176. IEEE Computer Society, 2002.

[MFP06] Gero Mühl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer, 2006.

[Mic08] Microsoft. SQL Server. `http://www.microsoft.com/sqlserver/2008/en/us/default.aspx`, 2008.

[Mic09a] Microsoft. Microsoft Message Queuing (MSMQ). `http://www.microsoft.com/windowsserver2003/technologies/msmq/`, 2009.

[Mic09b] Microsoft. Patterns and Practices: Publish/Subscribe. `http://msdn.microsoft.com/en-us/library/ms978603.aspx`, 2009.

[Mik02] Zoltán Miklós. Towards an access control mechanism for wide-area publish/-subscribe systems. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 516–524, Washington, DC, USA, 2002. IEEE Computer Society.

[MoM] Map of Medicine. http://www.mapofmedicine.com.

[Moo01] Ken Moody. Coordinating policy for federated applications. In *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security*, pages 127–134, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[MU00] Naftaly H. Minsky and Victoria Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering Methodologies*, 9(3):273–305, 2000.

[MUHW04] Gero Mühl, Andreas Ulbrich, Klaus Herrmann, and Torben Weis. Disseminating information to mobile clients using publish/subscribe. *IEEE Internet Computing*, 8(3):46–53, 2004.

[MY04] Jai Mohan and Raja Razali Raja Yaacob. The malaysian telehealth flagship application: a national approach to health data protection and utilisation and consumer rights. *International Journal of Medical Informatics*, 73(3):217–227, 2004.

[NAO07] National Audit Office. *Prescribing costs in primary care.* The Stationery Office, 2007.

[NAO08] National Audit Office. *The National Programme for IT in the NHS: Progress since 2006.* The Stationery Office, 2008.

[NEHTA] National E-Health Transition Authority. http://www.nehta.gov.au/.

[NHSa] National Health Service. http://www.nhs.uk/.

[NHSb] NHS Connecting For Health. http://www.connectingforhealth.nhs.uk/.

[NHS02] NHS Information Authority in conjunction with the Consumers Association and Health Which? Share with Care! people's views on consent confidentiality of patient information, 2002.

[NHS06a] NHS Connecting For Health. An introduction to Legitimate Relationships and Workgroups, July 2006. NPfIT Information Governance.

[NHS06b] NHS Connecting For Health. RBAC Statement of Principles, July 2006. NPfIT Access Control (Registration) Programme.

[NHS06c] NHS Connecting For Health. Sealed Envelopes Briefing Paper: 'Selective Alerting' Approach. http://www.connectingforhealth.nhs.uk/systemsandservices/infogov/confidentiality/sealedpaper.pdf, 2006.

[NHS06d] NHS: Ministerial Taskforce on the Summary Care Record. Report of the Ministerial Taskforce on the Summary Care Record. http://www.library.nhs.uk/HEALTHMANAGEMENT/ViewResource.aspx?resID=224460, Dec 2006.

[NHS07a] NHS. SNOMED CT — the language of the NHS Care Records Service, 2007.

[NHS07b] NHS Care Record Development Board. The care record guarantee—our guarantee for NHS Care Records in England. http://www.nigb.nhs.uk/guarantee, 2007.

[NHS07c] NHS Connecting For Health. RBAC Rationalisation Document v1.0, Feb 2007. NPfIT Access Control (Registration) Programme.

[NHS08a] NHS Connecting For Health. National RBAC Database v24, Feb 2008. NPfIT Access Control (Registration) Programme.

[NHS08b] NHS Connecting For Health. An introduction to Position Based Access Control (PBAC). http://www.esrsolution.co.uk/upload/file/IntroductiontoPositionBasedAccessControl(PBAC).pdf, 2008.

[NHS08c] NHS Connecting For Health. NHS number standard for general practice. http://www.connectingforhealth.nhs.uk/systemsandservices/nhsnumber/staff/documents/opinfogp.pdf, 2008.

[NHS08d] NHS Connecting For Health. The National RBAC Database User Guide v3.0, Feb 2008. NPfIT—Information Governance.

[NHS09a] NHS. Authorities and Trusts. http://www.nhs.uk/NHSEngland/aboutnhs/Pages/Authoritiesandtrusts.aspx, 2009.

[NHS09b] NHS 23. A dossier of concerns. http://www.nhs-it.info, 2009.

[NHS09c] NHS Connecting For Health. Data services. http://www.connectingforhealth.nhs.uk/systemsandservices/data, 2009.

[NHS09d] NHS Connecting For Health. N3 Factsheet. http://www.connectingforhealth.nhs.uk/systemsandservices/n3, 2009.

[NTBL07] Qun Ni, Alberto Trombetta, Elisa Bertino, and Jorge Lobo. Privacy-aware role based access control. In *SACMAT '07: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 41–50, New York, NY, USA, 2007. ACM Press.

[OAS05a] OASIS eXtensible Access Control Markup Language (XACML) Technical Committee. eXtensible Access Control Markup Language (XACML) v2.0. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2005.

[OAS05b] OASIS eXtensible Access Control Markup Language (XACML) Technical Committee. XACML RBAC Profile v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf, 2005.

[OP01] Lukasz Opyrchal and Atul Prakash. Secure distribution of events in content-based publish subscribe systems. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 21–21, Berkeley, CA, USA, 2001. USENIX Association.

[OPA07] Lukasz Opyrchal, Atul Prakash, and Amit Agrawal. Supporting privacy policies in a publish-subscribe substrate for pervasive environments. *Journal of Networks*, 2(1):17–26, 2007.

[Ora09a] Oracle. Oracle Advanced Queueing. http://www.oracle.com/technology/products/aq/htdocs/aq9i_overview.html, 2009.

[Ora09b] Oracle. Oracle Database. http://www.oracle.com/database/index.html, 2009.

[Pat06] Shyam Pather. *Microsoft SQL Server 2005 Notification Services.* Sams, 2006.

[PB02] Peter R. Pietzuch and Jean M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *DEBS '02: Proceedings of the 1st International Workshop on Distributed Event-Based Systems*, pages 611–618, Vienna, Austria, July 2002.

[PB05] Lauri I. W. Pesonen and Jean Bacon. Secure event types in content-based, multi-domain publish/subscribe systems. In *SEM '05: Proceedings of the 5th International Workshop on Software Engineering and Middleware*, pages 98–105, New York, NY, USA, September 2005. ACM Press.

[PBE+09] Ioannis Papagiannis, Jean Bacon, David Eyers, Matteo Migliavacca, Peter Pietzuch, and Brian Shand. Privateflow: Decentralised information flow control in event based middleware (demo). In *3rd ACM International Conference on Distributed Event-Based Systems (DEBS)*, Nashville, TN, USA, 07/2009 2009.

[PCB00] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom '00: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 32–43, New York, NY, USA, 2000. ACM.

[PD99] Norman W. Paton and Oscar Díaz. Active database systems. *ACM Computing Surveys*, 31(1):63–103, 1999.

[PEB06] Lauri I. W. Pesonen, David M. Eyers, and Jean Bacon. A capabilities-based access control architecture for multi-domain publish/subscribe systems. In *SAINT '06: Proceedings of the Symposium on Applications and the Internet*, pages 222–228, Phoenix, AZ, January 2006. IEEE.

[PEB07a] Lauri I. W. Pesonen, David M. Eyers, and Jean Bacon. Encryption-enforced access control in dynamic multi-domain publish/subscribe networks. In *DEBS '07: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems*, pages 104–115, New York, NY, USA, 2007. ACM.

[PEB07b] Lauri I.W. Pesonen, David M. Eyers, , and Jean Bacon. Access control in decentralised publish/subscribe systems. *Journal of Networks*, 2(2):57–67, April 2007.

[PGDG09a] PostgreSQL Global Development Group. `http://www.postgresql.org`, 2009.

[PGDG09b] PostgreSQL Global Development Group. Triggers. `http://www.postgresql.org/docs/8.0/interactive/triggers.html`, 2009.

[PIAG08a] Patient Information Advisory Group. PIAG Response to NHS CFH Consultation on Public, Patients and other interested parties views on Additional Uses of Patient Data . `http://www.dh.gov.uk/ab/PIAG/index.htm?IdcService=GET_FILE&dID=185043&Rendition=Web`, 2008.

[PIAG08b] Patient Information Advisory Group. PIAG Response to the Consultation on the NHS Constitution . `http://www.dh.gov.uk/ab/PIAG/index.htm?IdcService=GET_FILE&dID=185094&Rendition=Web`, 2008.

[Pie04] Peter R. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, University of Cambridge, 2004.

[PSC07] UK Parliament: Select Committee on Health Written Evidence. Evidence submitted by the Foundation for Information Policy Resarch (EPR 61). `http://www.publications.parliament.uk/pa/cm200607/cmselect/cmhealth/422/422we22.htm`, March 2007.

[PSI] PSIRP. Publish/Subscribe Internet Routing Paradigm. `http://www.psirp.org/`.

[RDD07] Giovanni Russello, Changyu Dong, and Naranker Dulay. Authorisation and conflict resolution for hierarchical domains. In *POLICY '07: Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 201–210, Washington, DC, USA, 2007. IEEE Computer Society.

[RDD08] Giovanni Russello, Changyu Dong, and Naranker Dulay. A workflow-based access control framework for e-Health applications. In *AINAW '08: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops*, pages 111–120, Washington, DC, USA, 2008. IEEE Computer Society.

[RHH+04] M Robling, K Hood, H Houston, R Pill, J Fay, and H Evans. Public attitudes towards the use of primary care patient record data in medical research without consent: a qualitative study. *Journal of Medical Ethics*, 30(1):104–109, February 2004.

[RI80] Daniel J. Rosenkrantz and Harry B. Hunt III. Processing Conjunctive Predicates and Queries. In *VLDB '80: Proceedings of the Sixth International Conference on Very Large Data Bases*, pages 64–72. VLDB Endowment, 1980.

[RM86] R.Kowalski and M.Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[RPG05] Mohsen Rouached, Olivier Perrin, and Claude Godart. A contract-based approach for monitoring collaborative web services using commitments in the event calculus. In *WISE '05: Web Information Systems Engineering*, pages 426–434, 2005.

[RPS07] Royal Pharmaceutical Society of Great Britain. Medicines, Ethics and Practice: A guide for pharmacists and pharmacy technicians. `http://www.rpsgb.org.uk/pdfs/MEP31s1-2b.pdf`, July 2007.

[RR06] Costin Raiciu and David S. Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. In *Securecomm '06: Proceedings of the Second IEEE/CreatNet International Conference on Security and Privacy in Communication Networks*, 2006.

[SAB+00] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content based routing with Elvin4. In *Proceedings AUUG2k*, 2000.

[SBC+98] Robert Strom, Guruduth Banavar, Tushar Chandra, Marc Kaplan, Kevan Miller, Bodhi Mukherjee, Daniel Sturman, and Michael Ward. Gryphon: An Information Flow Based Approach to Message Brokering. In *Proceedings of the 9th International Symposium on Software Reliability Engineering (ISSRE'98)*. IEEE Computer Society, 1998.

[SBM07] Jatinder Singh, Jean Bacon, and Ken Moody. Dynamic trust domains for secure, private, technology-assisted living. In *Proceedings of the the Second International Conference on Availability, Reliability and Security (ARES'07)*, pages 27–34, Vienna, April 2007. IEEE Computer Society.

[SBS08] Daniel Sturman, Guruduth Banavar, and Robert Strom. Reflection in the gryphon message brokering system. In *Reflection Workshop of the 13th ACM Conference on Object Oriented Programming Systems, Languages and Applications*, 2008.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[Sch03] Bruce Schneier. *Beyond Fear: Thinking Sensibly about Security in an Uncertain World*. Springer-Verlag, 2003.

[Sch04] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2004.

[SE05] Scottish Executive. Building a Health Service Fit for the Future (The Kerr Report). `http://www.scotland.gov.uk/Publications/2005/05/23141307/13348`, May 2005.

[SGI04] Scotland Government. Information services division. Effective interventions unit: Integrated care pathways guide 4: Analysis and review. `http://www.scotland.gov.uk/Resource/Doc/47237/0013492.pdf`, 2004.

[SHA] Office of the Strategic Health Authorities. `http://www.osha.nhs.uk`.

[Sha99] Murray Shanahan. The event calculus explained. *Artificial Intelligence Today*, pages 409–430, 1999.

[SL05] Mudhakar Srivatsa and Ling Liu. Securing publish-subscribe overlay services with EventGuard. In *CCS '05: Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 289–298, New York, NY, USA, 2005. ACM.

[SL07] Mudhakar Srivatsa and Ling Liu. Secure event dissemination in publish-subscribe networks. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 22, Washington, DC, USA, 2007. IEEE Computer Society.

[SM02] Sun Microsystems. Java Message Service Specification 1.2. `http://java.sun.com/products/jms/`, 2002.

[SM09] Sun Microsystems. MySQL. `http://www.mysql.com/`, 2009.

[SRLH05] Margaret A. Stone, Sarah A. Redsell, Jennifer T. Ling, and Alastair D. Hay. Sharing patient data: competing demands of privacy, trust and research in primary care. *British Journal of General Practice*, 55:783–789, 2005.

[Ste97] Thomas A. Stewart. *Intellectual capital: the new wealth of organizations*. Doubleday, New York, NY, USA, 1997.

[SVB08] Jatinder Singh, Luis Vargas, and Jean Bacon. A model for controlling data flow in distributed healthcare environments. In *PervasiveHealth '08: Proceedings of the 2nd International Conference on Pervasive Computing Technologies for Healthcare*, pages pages 188–191, 2008.

[SVBM08] Jatinder Singh, Luis Vargas, Jean Bacon, and Ken Moody. Policy-Based Information Sharing in Publish/Subscribe Middleware. In *Proceedings of the 9th International Workshop on Policies for Distributed Systems and Networks (Policy'08)*, pages 137–144. IEEE Computer Society, 2008.

[SYA09] Cheng-Chun Shu, Erica Y. Yang, and Alvaro E. Arenas. Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques. In *Proceedings of the 10th International Workshop on Policies for Distributed Systems and Networks (Policy'09)*, pages 182–185. IEEE Computer Society, 2009.

[TGNO92] Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous Queries over Append-Only Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data (SIGMOD'92)*, pages 321–330. ACM, 1992.

[TGP06a] Anthony Tomasic, Charles Garrod, and Kris Popendorf. On the evaluation of symmetric publish/subscribe. In *ExpDB: First International Workshop on Performance and Evaluation of Data Management Systems*, 2006.

[TGP06b] Anthony Tomasic, Charles Garrod, and Kris Popendorf. Symmetric publish/subscribe via constraint publication. Technical Report CMU-CS-06-129R, Carnegie Mellon University, Pittsburgh, PA, 2006.

[TIB09] TIBCO. TIBCO Rendezvous. `http://www.tibco.com/software/messaging/rendezvous/`, 2009.

[TLDS08] Kevin Twidle, Emil Lupu, Naranker Dulay, and Morris Sloman. Ponder2 - a policy environment for autonomous pervasive systems. In *POLICY '08: Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pages 245–246, Washington, DC, USA, 2008. IEEE Computer Society.

[UBJ+04] Andrzej Uszok, Jeffrey M. Bradshaw, Matthew Johnson, Renia Jeffers, Austin Tate, Jeff Dalton, and Stuart Aitken. Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.

[UK 98] UK Crown. Data Protection Act, 1998.

[UK 06] UK Crown. The Controlled Drugs (Supervision of Management and Use) Regulations, 2006.

[UKACR09] United Kingdom Association of Cancer Registries. Legal background. `http://www.ukacr.org/confidentiality/background.asp`, 2009.

[Var09] Luis Vargas. *Integrating Databases and Publish/Subscribe*. PhD thesis, University of Cambridge, 2009.

[VBM05] Luis Vargas, Jean Bacon, and Ken Moody. Integrating Databases with Publish/Subscribe. In *DEBS '05: Proceedings of the Fourth International Workshop on Distributed Event-Based Systems*, pages 392–397. IEEE Computer Society, 2005.

[VBM08] Luis Vargas, Jean Bacon, and Ken Moody. Event-Driven Database Information Sharing. In *Proceedings of the 25th British National Conference on Databases (BNCOD'08)*, volume 5071 of *Lecture Notes in Computer Science (LNCS)*, pages 113–125. Springer, 2008.

[Vit09] Vitria. Vitria Enterprise Service Bus (ESB). `http://www.vitria.com/M3O/Enterprise-Service-Bus.php`, 2009.

[W3C99] W3C. XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath`, 1999.

[W3C01] W3C. W3C Semantic Web Activity. `http://www.w3.org/2001/sw/`, 2001.

[W3C06] W3C. Web Services Eventing (WS-Eventing). `http://www.w3.org/Submission/WS-Eventing/`, 2006.

[W3C08] W3C. Extensible Markup Language (XML) Fifth Edition. `http://www.w3.org/TR/REC-xml/`, 2008.

[WBL+07] Andrew Witkowski, Srikanth Bellamkonda, Hua-Gang Li, Vince Liang, Lei Sheng, Wayne Smith, Sankar Subramanian, James Terry, and Tsae-Feng Yu. Continuous queries in oracle. In *VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 1173–1184. VLDB Endowment, 2007.

[WCEW02] C. Wang, A. Carzaniga, D. Evans, and A. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, page 303, Washington, DC, USA, 2002. IEEE Computer Society.

[WFSM02] Marc Wilikens, Simone Feriti, Alberto Sanna, and Marcelo Masera. A context-related authorization and access control method based on rbac:. In *SACMAT '02: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, pages 117–124, New York, NY, USA, 2002. ACM.

[WHEH06] Richard Whiddett, Inga Hunter, Judith Engelbrecht, and Jocelyn Handy. Patients attitudes towards sharing their health information. *International Journal of Medical Informatics*, 75(7):530–541, 2006.

[WHO00] World Health Organisation. The world health report 2000 - Health systems: improving performances. `http://www.who.int/entity/whr/2000/en/whr00_en.pdf`, 2000.

[WHO02] World Health Organisation. Innovative Care for Chronic Conditions. `http://www.who.int/entity/diabetesactiononline/about/icccglobalreport.pdf`, 2002.

[WJ07] Alex Wun and Hans-Arno Jacobsen. A policy management framework for content-based publish/subscribe. In *Middleware '07*, Lecture Notes in Computer Science 4834, pages 368–388, Newport Beach, CA, 2007. Springer.

[ZS06] Yuanyuan Zhao and Daniel C. Sturman. Dynamic access control in a content-based publish/subscribe system with delivery guarantees. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 60, Washington, DC, USA, 2006. IEEE Computer Society.

Appendices

A

Transformations and Interaction Points

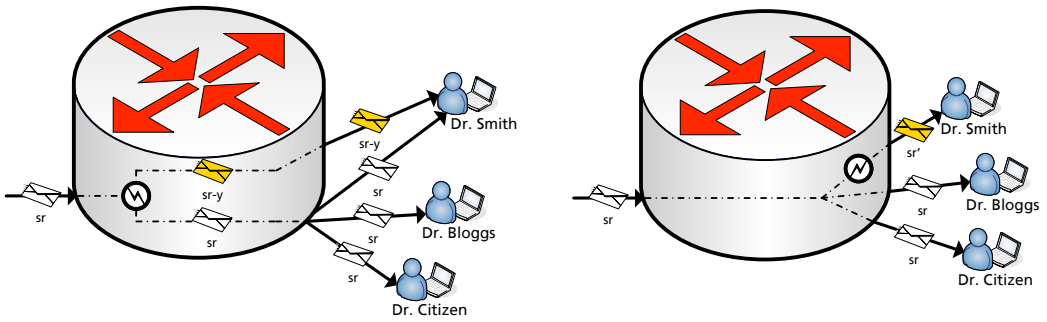
Here we describe the suitability of transformation rules to particular interaction points.

Publication transformations are applied when a broker receives an input event that satisfies all applicable imposed conditions. Such transformations are useful for purposes of interoperability, as they can convert events into a form more suitable for local processing. From a data control perspective, they are best used when the transform applies to general classes of subscribers, as a single transform is executed regardless of the number of subscribers.

An event type embodies a particular semantic. Subscriptions are made to a specific type that represents (some of) the information the subscriber is interested in receiving. As publication transformations are applied before the subscription matching phase, they are well-suited to type transformations (Fig. A.2). Notification transformations are more appropriate for transformations altering content for *specific* subscribers (Fig. A.1); especially for events with infrequent subscriptions.

The appropriate interaction point for a transformation, or for that matter any policy, is a question of design. Consider the example presented in Fig. A.1 where Patient Y requests that a certain attribute of a sensor reading event is hidden only from Dr. Smith. As shown in Fig. A.1(b), this is easily effected through a notification transformation targeted at Dr. Smith that nullifies the appropriate value on delivery of events concerning Patient Y. To achieve this restriction through a publication transformation, a specific event type (`sr-y`) is required for which only Dr. Smith can subscribe (Fig. A.1(a)). The transform would produce this event, with the nullified value, when events concern Patient Y for delivery to Dr. Smith—while the unperturbed events (of the other type) are delivered to other subscribers. Awkwardness aside, Dr. Smith is aware that some policy pertains to him, as he is required to issue a different subscription for Patient Y than for his other patients.

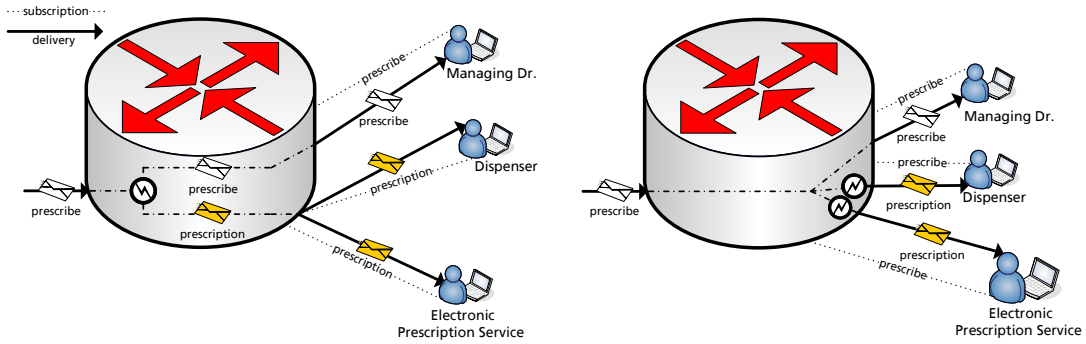
The process of prescribing a drug in a hospital involves recording symptoms and observations. Neither the hospital dispenser, nor the prescribing service funding medications, should receive such information (§9.1). As shown in Fig. A.2(a), it is sensible to convert the `prescribe` event into a `prescription` on receipt by a broker, which is forwarded to



(a) Publication transform for Dr. Smith (b) Notification transform for Dr. Smith

Figure A.1: A situation where a notification transformation is appropriate.

those directly subscribed to the respective types. Fig. A.2(b) shows that modelling this scenario through notification transformations is cumbersome, resembling more topic-based pub/sub, where all those interested subscribe to the general `prescribe` event but may receive events of other types. Further, here notification transformations result in multiple executions of the same transformation function, while the subscription channel is type-overloaded.[1] We advocate type transformations only on publication, as overloading a type channel is unintuitive for clients, and can confuse the application of policy (§7.8.3).



(a) Publication transformations (b) Notification transformations

Figure A.2: A situation favouring publication transformations for `prescribe` events.

[1]Types appear overloaded from the perspective of the subscriber, in that they receive an event of a type different from that of their subscription. From a middleware perspective, if an event channel does not exist for the output type, one could be temporarily created for that subscriber for the delivery of the event. See §7.8.3 for discussion.

B

Fluent Representation

Fig. B.1 illustrates how the emergency fluent, as described in §6.1.1, is updated for a particular patient. The fluent is initiated by three `sensor_reading` events with a heart rate of concern, or through the patient raising a `panic` event by pressing an emergency button. The fluent is terminated when an `emergency_clear` event is received.

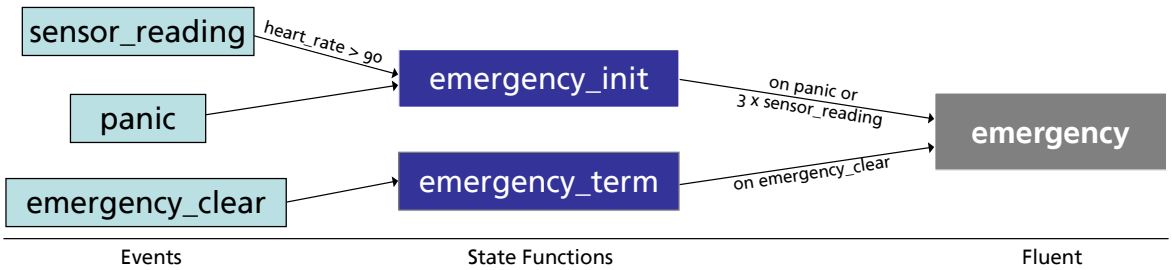


Figure B.1: The events and functions maintaining the the state of the `emergency` fluent for a particular patient.

In this example, the emergency status of a patient is maintained in the table (materialised view) `fluent_emergency [int patient_id, bool holds]`, where the current state of the fluent is accessed by the function `emergency(patient_id)`. The function queries the table, returning the state as represented by the value of `holds`.

The initiation and termination of a fluent are performed through its `_init(a)` and `_term(a)` functions respectively. These functions *may* initiate or terminate a fluent—an execution does not necessarily entail a change in fluent state because the function may have code defining the situations in which fluent state should be updated. That is, the function might involve queries or calculations that, for example, are useful for integrating composite event functionality. In this example, the `countWarning(pat_id)` and `clearWarnings(pat_id)` functions record the number of `sensor_reading` events of concern received for the particular patient.[1]

[1]The `countWarning(pat_id)` function counts the number of `sensor_reading` warnings received for

Here, the `_init(a)` and `_term(a)` functions are tied to events through active rules (internal subscriptions). The event triggering the rule is injected into the function, to help decide whether to initiate/terminate the fluent. The code depicted in Fig. B.2 uses the event instance to determine the relevant patient ID for the fluent.

```
--Create the table to cache the fluent result
create table fluent_emergency(int patient_id, bool holds);

--The function representing the HoldsAt predicate
create function emergency(int pat_id) returns bool as $$
    select count(patient_id) > 0 from fluent_emergency where patient_id = pat_id and holds;
$$ language sql;

--The function representing the Initiates predicate
create function emergency_init() returns void as $$
begin
    if ctx_event_type = 'panic' OR
        (ctx_event_type = 'sensor_reading'
            AND countWarning(ctx_event.patient_id)) then
    begin
        execute 'update fluent_emergency set holds=true
            where patient_id = ' || ctx_event.patient_id;
    end;
end;
$$ language plpgsql;

--The active rules (subscriptions) tying the Initiates operation to the events
create internal subscription em_set_p on panic execute procedure emergency_init();
create internal subscription em_set_sr on sensor_reading where sensor_reading.heart_rate > 90
    execute emergency_init();

--The function to Terminate the fluent and reset the warning counter
create function emergency_term() returns void as $$
    begin
        execute 'update fluent_emergency set holds=false
            where patient_id = ' || ctx_event.patient_id;
        clearWarnings(ctx_event.patient_id);
    end;
$$ language plpgsql;

--Subscription to Terminate the fluent
create internal subscription em_cease on emergency_clear execute procedure emergency_term();
```

Figure B.2: Code to maintain state of an emergency fluent.

a particular patient. The function increments the counter and returns true if the warning limit (3) has been reached. The function `clearWarnings(pat_id)` resets this counter.

C

Example XML Rule Definitions

Here we present an example XML rule definition for each type of IC rule.[1]

C.1 Link Authorisation Rules

Link authorisation rules permit the establishment of a link for a remote broker. As shown in Fig. C.1, such rules reference the credentials of a broker.

```
<link_authorisation>
    <rule_name>allowsurgery</rule_name>
    <credentials>NHSCred(usernm,'surgery')</credentials>
    <notes>Permit connections from other surgeries</notes>
</link_authorisation>
```

Figure C.1: A rule authorising links with brokers of registered surgeries.

C.2 Request Authorisation Rules

Fig. C.2 presents a request authorisation rule that authorises a doctor to subscribe to event types for patients that he treats. This includes monitored conditions and a permission attribute to ensure an entity has a working relationship with the patient. The `atPatientHome(...)` fluent is monitored, triggering re-evaluation on a value change.

Note that monitored conditions may also be included in the `<conditions>` element to describe how they are combined with other predicates for initial evaluation. If the monitored conditions are not specified in this element, they are automatically applied in conjunction.

[1] Note that we avoid XML escaping to ease illustration.

```
<request_authorisation>
    <rule_name>doctorathome</rule_name>
    <event_type>sensor_reading</event_type>
    <request_type>a</request_type>
    <credentials>NHSCred(usernm,'doctor')</credentials>
    <permission_attributes>patient_id:int8</permission_attributes>
    <conditions>onCall(usernm) AND atPatientHome(usernm, att.patient_id)</conditions>
    <mon_conditions>atPatientHome(usernm, att.patient_id)</mon_conditions>
    <notes>Allows a locum doctor to subscribe to a patients sensor stream when visiting
        the patient's home. The subscription is cancelled once the doctor leaves.</notes>
</request_authorisation>
```

Figure C.2: A request authorisation rule concerning patient sensor streams.

C.3 Imposed Conditions

Imposed conditions apply a restriction filter to an event channel. An imposed condition rule definition for controlling information flows for research is depicted in Fig. C.3.

```
<imposed_condition>
    <rule_name>researchuse</rule_name>
    <event_type>treatment</event_type>
    <interaction_point>n</interaction_point>
    <credentials>NHSCred(usernm,'researchDomainX')</credentials>
    <permission_attributes />
    <restrictions>givenConsent(treatment.patient_id,'projX')</restrictions>
    <notes>A research service must only receive events concerning patients who have given consent</notes>
    <hidden>f</hidden>
</imposed_condition>
```

Figure C.3: An imposed condition rule filtering data for research.

As shown in Fig. C.4, request forwarding filters are similar, except that they are evaluated in the context of a request.

```
<request_imposed_condition>
    <rule_name>surgerieswithtreatingrelations</rule_name>
    <event_type>treatment</event_type>
    <request_type>s</request_type>
    <credentials>NHSCred(usernm,'surgery')</credentials>
    <restrictions>domainTreatsPatient(usernm,req.patient_id)</restrictions>
    <notes>Subscriptions for treatment events are only forwarded to surgeries
        treating the particular patient</notes>
</request_imposed_condition>
```

Figure C.4: Request forwarding rule controlling subscription propagation based on treating relationships.

C.4 Transformations

Transformations consist of a *transformation function* defining the database function to convert an event instance into another (see Fig. E.3),[2] and a *transformation rule* defining the situation in which the function is executed.

[2]Our implementation includes *mapping functions*, allowing XML specification of simple transformation functions. These are converted into stored procedures—see [SVBM08] for an overview.

Fig. C.5 presents an example transformation rule. The policy conversion function ensures that the defined function exists in the database, referenced by its unique database object ID (`Oid`), and that the function returns the appropriate type.

```
<transformation>
    <rule_name>familyplanningpseudonym</rule_name>
    <event_type>treatment</event_type>
    <output_event>treatment</output_event>
    <interaction_point>p</interaction_point>
    <consumable>t</consumable>
    <function>perturbPatID</function>
    <!--- applicability conditions --->
    <permission_attributes />
    <credentials />
    <conditions>treatment.specialisation=familyplanning</conditions>
    <notes>Use a pseudonym for family planning data</notes>
</transformation>
```

Figure C.5: A transformation rule definition to hide patient information

Fig. C.6 shows a sample request transformation rule. These rules differ to event transformation rules in that a) the output type is fixed, returning a request (or `null`); and b) such rules are connected to a link authorisation policy, and thus do not reference credentials.

```
<request_transformation>
    <rule_name>surgerytransform-X123F</rule_name>
    <event_type>prescribe</event_type>
    <request_type>a</request_type>
    <authorising_link>X123FauthSurgery</authorising_link>
    <function>convertPrescribeForSurgeryX123F()</function>
    <notes>Transformation an advertisement request for a particular surgery</notes>
</request_transformation>
```

Figure C.6: A request transformation rule definition for a particular surgery.

C.5 Conflict Resolution

Example conflict resolution rule definitions are presented in Figs. C.7 and C.8.

```
<override>
    <rule_name>overriderule</rule_name>
    <overrider>rule2</overidder>
    <overridden>rule1</overridden>
    <notes>Rule2 overrides rule1 in particular circumstances as it is a specialisation.</notes>
</override>
```

a) Overrides constraint definition

```
<ordering>
    <rule_name>orderrule</rule_name>
    <applied_before>rule1</applied_before>
    <applied_after>rule2</applied_after>
    <notes>Transformation rule1 must occur before rule2 as it alters a table from which rule2 reads.</notes>
</ordering>
```

b) Ordering constraint definition

Figure C.7: Conflict resolution definitions.

```
<incompatible>
    <rule_name>inc_id</rule_name>
    <policy>rule1</policy>
    <policy>rule2</policy>
    <policy>rule3</policy>
    <notes>Rules 1,2,3 are incompatible because...
        If all are active, something erroneous has occured.</note>
</incompatible>
```

Figure C.8: Definition of incompatible rules.

.

D

Hook Rule Specifics

This Appendix describes the operational specifics of PostgreSQL-PS hook rules.

D.1 Event Transformations

An *event transformation* hook executes a transformation function, which returns an (altered) event instance, or `null`. The output event of the transformation function moves to the next stage of the messaging process. The syntax for a transformation hook rule is shown in Fig. D.1.

```
CREATE TRANSFORMATION rulename
ON
     { PUBLICATION OF eventtypename FROM username
     | NOTIFICATION OF eventtypename TO username }
[WHERE filter]
EXECUTE FUNCTION funcname
```

Figure D.1: The syntax for *event transformation* hook rules.

A transformation hook can be defined to apply on publication or on notification (delivery) of a particular event type. The *username* specifies the principal whose connection is subject to the transformation rule. This means that the rule applies to all relevant event channels (of the given type) for the connection. The transformation function executes subject to any conditions set by the optional *filter;* absence of a filter means the function is always executed. The transformation function, *funcname*, is a locally defined database function that performs the relevant transformation operations, returning an event which moves to the next stage of the messaging process.

The callback mechanisms inject the event instance (`CTX_EVENT`) into the execution space of an event transformation function, and other metadata such as user details.

D.1.1 Resolving Event Transformations

Resolver hook rules deal with conflicts between transformation hook rules. A resolution function (*funcname*) takes as input an array of the names of the *applicable rules*, transformation rules whose conditions (guard) for execution hold in the context of a particular event instance. The purpose of the function is to resolve any conflicts in this set by returning an ordered list of policies—the functions of which are executed in order. The syntax for defining a resolution hook rule is presented in Fig. D.2.

```
CREATE RESOLVER resolvername
ON { PUBLICATION | NOTIFICATION }
EXECUTE FUNCTION funcname
```

Figure D.2: The syntax for *resolver* hook rules.

D.2 Request Validators

The *request validator* hook (Fig. D.3) enables the data control layer to govern the establishment of an event channel. These hooks call functions that query the policy store to either deny a request, or authorise a request and impose the relevant restrictions. The pub/sub layer uses the resulting request instance to establish the event channel.

```
CREATE REQUEST VALIDATOR rulename
ON { SUBSCRIPTION | ADVERTISEMENT }
EXECUTE FUNCTION funcname
```

Figure D.3: The syntax for a *request validator* rule.

Request processing, unlike event transformations, involves more than a one-off manipulation of attribute values. Requests concern the construction of event channels, which involves authorising the event channel, imposing restrictions (transformations, imposed conditions) and creating active rules to handle monitored conditions. As such, request validators are not subject to a (hook-level) resolution strategy, and are defined without conditional filters to ensure a standardised validation process; i.e. only one validator hook can be defined for each request type.

D.3 Connection of Links

Link authorisation rules control the permissibility of connections between brokers. These are evaluated through the *link validator* hook (Fig. D.4), which fires on broker interconnection. This hook also enables the establishment of advertisement forwarding restrictions. As links are reciprocal, in that requests and events flow in both directions, *both* the receiving and initiating broker[1] execute (their respective version of) the hook function. The function is passed a structure with the identifier of the remote broker, which also contains an empty filter for recording the advertisement filters relevant for the link, an attribute

[1]The *initiating* broker initiates the connection to the *receiving* broker.

for a (feedback) message, and an attribute recording the policy ID of the link authorisation rule allowing the connection, which is set to `null` if unauthorised. The value of any advertisement filter returned by the function is persisted in the *Links* catalogue, along with authorising policy ID.

```
CREATE LINK VALIDATOR rulename
EXECUTE FUNCTION funcname
```

Figure D.4: The syntax for the *link validator* hook rule.

D.4 Request Transformations

Request transformation hook rules, as shown in Fig. D.5, are executed on the forwarding of a request through a link to another broker. The function takes a request, consumes it, and returns a request, which is then evaluated against applicable forwarding filters. For consistency, only a single transformation rule can be defined per link for a particular event and request type (§5.6.3).

```
CREATE { ADVERTISEMENT | SUBSCRIPTION } TRANSFORMATION rulename
TO brokername  ON eventtypename
[WHERE filter]
EXECUTE FUNCTION funcname
```

Figure D.5: The syntax for a request forwarding validation rule.

D.5 Advertisement Processor

The *link advertisement processor* hook rule fires on the receipt of an advertisement request through a link. This hook is defined to enable the establishment of subscription filters on receipt of an advertisement. The syntax of the rule is presented in Fig. D.6

```
CREATE LINK ADVERTISEMENT PROCESSOR rulename
EXECUTE FUNCTION funcname
```

Figure D.6: The syntax for the *link advertisement processor* hook rule.

D.6 Bootstrapping

PostgreSQL-PS uses PostgreSQL's initialisation process to configure the broker, creating the default structures (catalogues) to store information of advertisements, subscriptions, connected clients, links, routing tables and event type definitions. We extend this process to introduce the extra catalogues required to implement hook rules, and, as shown in

Hook Rule Type	Function	Purpose
REQUEST VALIDATOR ON ADV	ValidateAdvRequest()	Validates and processes the incoming request, establishing the relevant restrictions.
REQUEST VALIDATOR ON SUB	ValidateSubRequest()	Validates and processes the incoming request, establishing the relevant restrictions.
RESOLVER	ResolveConflicts()	Resolves conflicts within the active set of policies.
LINK VALIDATOR	ValidateLink()	Authorises the connection to the remote broker and loads advertisement forwarding restrictions.
LINK ADVERTISEMENT PROCESSOR	LoadSubReqRestrictions()	Loads the subscription forwarding restrictions relevant to the advertisement.

Figure D.7: Hook rules and associated functions loaded on initialisation.

Fig. D.7, to define the general functions and associated hooks to provide essential IC functionality. The audit mechanisms detailed in Ch. 10 are also created at this stage.

Also defined is a generic function for conflict resolution, which takes a set of active policies and uses the resolution strategies described in §6.4.3 to return an ordered set of policies conforming to any defined constraints. If the resolver returns `null` it means the policy set is incompatible, either by an incompatibility definition, or due to cyclical overriding/ordering constraints. The resolver function audits and publishes an event to inform of the incompatibility, which includes the values of the `notes` defined in the resolution strategy. This same function is used throughout the enforcement process: a common resolution strategy gives clarity and predictability of rule enforcement.

To implement the consumable property of transformation rules, we define a placeholder transformation hook rule representing the original event instance. Consumable transformation rules override the placeholder, which prevents the original event from propagating. This is illustrated by the `original` policy in Fig. 10.3.

In our implementation, we only forward advertisements to brokers that have the *possibility* to subscribe to the particular type. This privilege is encapsulated in the fluent `CanSubscribe(brokername, eventtype)`, which is created on initialisation. Its values are derived from the local subscription authorisation policy store: if an authorisation rule exists for the broker and the event type, the fluent holds. The non-credential predicates of the rules are not considered, as the existence of an authorisation policy means there is *some* set of circumstances in which a subscription is authorised. Fluent values are updated through rules that monitor the policy store for changes in subscription authorisation policies. This is monitored, as a change in state can affect routing tables.

E

Prescribing Policy Rules

This Appendix presents the policy rules governing the prescribing scenario of §9.1

E.1 Event Authorisation Rules

```
<request_authorisation>
    <rule_name>drprescribe</rule_name>
    <event_type>prescribe</event_type>
    <request_type>s</request_type>
    <credentials>NHSCred(usernm, 'doctor')</credentials>
    <permission_attributes>patient_id:int8</permission_attributes>
    <mon_conditions>treatsPatient(usernm, att.patient_id)</mon_conditions>
    <notes>Allows a doctor to subscribe to prescribe events for patients that they treat.</notes>
</request_authorisation>
```

```
<request_authorisation>
    <rule_name>epsprescription</rule_name>
    <event_type>prescription</event_type>
    <request_type>s</request_type>
    <credentials>NHSCred(usernm,'eps')</credentials>
    <notes>The Electronic Prescribing Service is authorised to subscribe to all prescription events</notes>
</request_authorisation>
```

```
<request_authorisation>
    <rule_name>drugauditauditor</rule_name>
    <event_type>drug_audit</event_type>
    <request_type>s</request_type>
    <credentials>NHSCred(usernm,'drug_auditor')</credentials>
    <notes>The auditor can subscribe to all drug_audit events</notes>
</request_authorisation>
```

```
<request_authorisation>
    <rule_name>drugauditprescribe</rule_name>
    <event_type>prescribe</event_type>
    <request_type>s</request_type>
    <credentials>NHSCred(usernm,'drug_auditor')</credentials>
    <notes>The auditor can subscribe to all prescribe events</notes>
</request_authorisation>
```

E.2 Imposed Condition for the Auditor

```
<imposed_condition>
    <rule_name>auditorprescribeinvestigation</rule_name>
    <event_type>prescribe</event_type>
    <interaction_point>n</interaction_point>
    <credentials>NHSCred(usernm,'drug_auditor')</credentials>
    <restrictions>underInvestigation(prescribe.prescriber_id)
        AND givenAuditorConsent(prescribe.patient_id)</restrictions>
    <notes>Prescribe events can be delivered to the auditor when the prescriber
        is under investigation and consent has been given.</notes>
    <hidden>f</hidden>
</imposed_condition>
```

E.3 The `prescribe–prescription` Transformation

```
<transformation>
    <rule_name>createprescription</rule_name>
    <event_type>prescribe</event_type>
    <output_event>prescription</output_event>
    <interaction_point>p</interaction_point>
    <consumable>f</consumable>
    <function>prescribe_to_prescription</function>
    <notes>Create the prescription from the prescribe event.</notes>
</transformation>
```

```
create or replace function prescribe_to_prescription() returns callback as $BODY$
declare
     inp prescribe;
     out prescription;
     pat patient%ROWTYPE;
begin
     --copy across the fields
     inp := CTX_EVENT::prescribe;
     out.prescription_id := inp.prescription_id;
     out.drug_id = inp.drug_id;
     out.dosage := inp.dosage;
     out.prescriber_id := inp.prescriber_id;
     out.issuedate := inp.issuedate;
     --surgery details
     out.domain_stamp := SELECT domainstamp FROM domain_defaults;
     --lookup patient details...
     select into pat * from patient where patient_id = inp.patient_id LIMIT 1;
     out.patient_name := pat.full_name;
     out.patient_address := pat.address;
     out.patient_dob := pat.dob;
     return out;
     end;
$BODY$ language plpgsql;
```

E.4 The prescribe–audit Transformation

```
<transformation>
    <rule_name>createdrugaudit</rule_name>
    <event_type>prescribe</event_type>
    <output_event>drug_audit</output_event>
    <interaction_point>p</interaction_point>
    <consumable>f</consumable>
    <function>prescribe_to_audit</function>
    <conditions>controlledDrug(prescribe.drug_id)</conditions>
    <notes>Converts prescribe to drug_audit events</notes>
</transformation>
```

```
CREATE FUNCTION prescribe_to_audit RETURNS drug_audit AS $$
DECLARE
    inp prescribe;
    out drug_audit;
BEGIN
    -- Generated assignments
    inp := CTX_EVENT::prescribe;
    out.prescriber_id := inp.prescriber_id;
    out.drug_id := inp.drug_id;
    out.dosage := inp.dosage;
    out.repeat := inp.repeat;
    out.timestamp := inp.timestamp;
    -- Policy specified operations
    return out;
END;
$$ LANGUAGE plpgsql;
```