Policy-based Information Sharing in Publish/Subscribe Middleware

Jatinder Singh, Luis Vargas, Jean Bacon and Ken Moody
Computer Laboratory, University of Cambridge
Firstname.Lastname@cl.cam.ac.uk

***Abstract*—Healthcare is a highly collaborative environment, where the active sharing of information is central to the care process. Due to the sensitive nature of medical information, care providers are responsible for protecting data, controlling the circumstances in which it is released to others. The publish/subscribe (pub/sub) communication paradigm is useful for data dissemination, as it allows parties to specify their interest in receiving particular information. However, general pub/sub implementation frameworks lack mechanisms to control the flow of data. This paper describes the details of a model to define and enforce fine-grained information sharing policies in an active notification environment. The model, built above a pub/sub middleware, allows policy definitions to control information flow by 1) specifying the conditions for data access, and 2) tailoring information to suit particular circumstances.**

I. Introduction

Information sharing is a core requirement of modern application environments. Networking systems supporting such environments must manage the transmission of data across boundaries, which may be logical, physical or organisational. One function of policy is to define how information may be shared. Examples are the use of schemas to ensure interoperability, and the specification of authorisation rules to determine conditions in which particular actions are permitted.

Certain scenarios require that data be stringently controlled. Healthcare is one such environment; its collaborative nature requires that data must be available to those providing care, but the sensitivity of medical information means that it must also be protected. Work on data availability tends to focus on networking aspects, such as routing and interoperability, whilst access control schemes are usually request-based, granting or denying access to a particular action or data record. There is, however, an increasing need to share information actively, that is, to propagate and react to relevant information automatically, as soon as it becomes available. In such a setting, data must be *actively* controlled as part of the dissemination process.

Healthcare is a data-driven scenario, where one incident is often relevant to many care providers [1]. Each provider is responsible for different aspects of patient care, and thus each requires specific information. The information appropriate to a provider is sensitive to context; it depends on factors such as environmental state, patient preference, business procedures/-workflow, and the provider's identity, role and credentials. The aim of this work is to allow policy to regulate information flow on a *need-to-know* basis, so that collaborators receive that and only that information necessary to perform their duties.

This paper describes the use of policy to control and adapt data according to circumstances. We describe the specifics of a data control model built on publish/subscribe (pub/sub) middleware, explaining how policy functions to control the transmission of data as it flows through a system.

We begin by describing our application domain, the healthcare environment, noting the constraints and special considerations it imposes. In Section III, we outline our previous work on extending a database system with pub/sub functionality, describing the mechanisms provided to allow integration of data control policy into the active notification environment. In Section IV we give an overview of the functionality of our model, and describe how conditional clauses and transformations are used to control information. The definition, storage and representation of policy in our system is described in Section V: Section VI covers policy conflicts, explaining how they may be detected and resolved. In Section VII we describe our implementation, detailing the process of policy enforcement. We conclude with a discussion of related work.

II. Motivation - Healthcare

England is currently undertaking a large project to computerise and integrate National Health Service (NHS) data. Our work takes place under the CareGrid project, which focuses on middleware to support homecare environments. Despite differences in scope, the projects address similar issues: 1) how systems and parties can interoperate, potentially across management domains; 2) how to balance confidentiality and privacy against the data availability requirements for proper care; 3) how to support collaboration in an environment where entities deliver different services as part of the care process. Failures in health information systems can have serious personal, legal and privacy repercussions; thus policy must offer appropriate controls, in order to support collaboration whilst preserving the confidential nature of medical information [1].

A. Information Relevance

Healthcare is a collaborative environment, where care providers interact to provide various services as part of the care process. Each provider requires access to *relevant* information to perform their duties, but this relevance depends on circumstance, such as the care provider's credentials and role in the care process, patient particulars and preference, environmental state (e.g. emergency), and so forth. For example, a doctor is directly responsible for the care of a patient, and thus is

interested in all aspects of treatment, and more generally, in the patient's wellbeing. Other providers take specialist roles, dealing only with an aspect of care – such as a pharmacist who is concerned solely with dispensing drugs, or a medical accountant who deals only with funding. Pervasive/homecare environments lend themselves to cross-organisational collaborations, while introducing a range of technologies (e.g. sensors and monitoring devices) into the interaction mix.

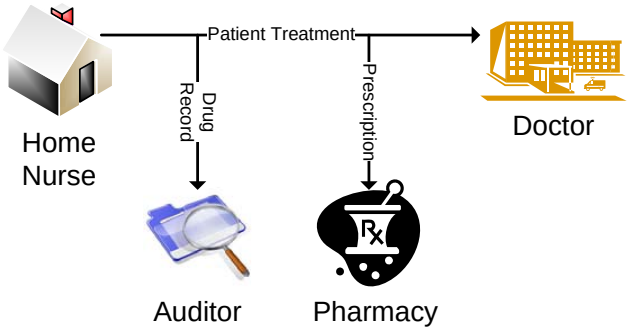

Fig. 1. Care providers requiring information on the prescribing of drugs.

As healthcare is a collaborative, data-driven environment, it follows that a single incident (event) regarding a patient is relevant to several parties. Given that carers deliver different services, often only aspects of an incident will be relevant to a particular provider. For example, a nurse caring for a patient at home might prescribe a drug [2], see Figure 1. As part of this process, he might record 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 care provider. Firstly, the doctor responsible for a patient needs all information regarding the treatment process, including notes, observations and reasons for the allocation of drugs. The pharmacist requires a valid prescription in order to dispense drugs legally, including the patient's personal details, and prescriber and drug information, but does *not* need any information about symptoms or observations [3]. The supply of controlled drugs is monitored by a designated auditor, who must be notified when such drugs are prescribed; but this monitoring focuses on the prescriber, and the auditor should usually not receive information of patient particulars [4].

This example demonstrates that while multiple providers interact as part of the care process, their roles within the process differ. Similarly, the data requirements for performing a particular task depend on the circumstances. In an environment where information is highly sensitive, it is desirable to control data so as to release *only* the information relevant for a particular party to perform their duty – i.e. provide data on a *need-to-know* basis.

B. Local responsibility

There is an explicit push at the executive level of the NHS to give more control to those providing (front-line) care services. This is reflected in the national technical programme [5], where systems, which must conform to central standards, are to be customised to support the requirements of local service providers. For instance, detailed care record services are being proposed, so that care providers can share information (observations and treatments) at a local level. Home environments are similar, in that a specific body (typically the Primary Care Trust, i.e. Surgery) will be responsible for managing the care of the patient. With control comes responsibility. Care providers share information according to circumstance, considering organisational and clinical requirements, as well as patient preference [5]. Thus, in addition to the immediate care of a patient, the responsibility of care providers extends to issues of data management – to ensure that relevant data is shared, while sensitive information is protected.

The healthcare domain differs from general networking environments: 1) the environment is necessarily controlled, so that interacting entities are not *anonymous* per se, but require credentials to act within the health-space; 2) because healthcare concerns wellbeing, care providers are held responsible for their contributions to the care process – both in delivering care and protecting data. Meeting these responsibilities requires policy enforcement mechanisms that can respond to circumstance (credentials and context).

III. Middleware

Middleware is a software layer that lies between a physical (network) infrastructure and a number of applications, allowing applications to communicate irrespective of implementation specifics. As middleware provides the sole means of communication for applications, it is an appropriate place to define and enforce information sharing policy. We use middleware to bring real world considerations and context into the messaging environment.

A. Publish/subscribe Middleware

Publish/subscribe [6] (pub/sub) is a widely used communication paradigm for large-scale distributed systems. Pub/sub is built on the notion of events, i.e. occurrences in the system that are to be shared. In this paradigm, an application takes the role of a publisher and/or a subscriber who communicates through the middleware. Subscribers register their interest in receiving an event of a particular type through a subscription, optionally specifying a condition (filter) on the content of events. A publisher produces events independently of subscribers; through a process termed *notification*, each event is delivered to a subscriber if it matches a relevant subscription. The interaction between publishers and subscribers occurs through a pub/sub middleware, which might be centralised as a single event broker, or decentralised as a network of event brokers that cooperate to route events from publishers to subscribers [7].

B. Publish/Subscribe Database Middleware

Databases provide an obvious point for information control. We have extended the PostgreSQL [8] open-source database system to include publish/subscribe middleware functionality [9]. This allows a database system in the local domain to function as an event broker (*broker*), reliably routing

```
CREATE HOOK RULE rulename
  ON SUBSCRIBE EXECUTE funcname(args)

CREATE HOOK RULE rulename
  ON PUBLISH ev_type condition EXECUTE funcname(args)

CREATE HOOK RULE rulename
  ON NOTIFY ev_type TO username condition
    EXECUTE funcname(args)
```

Fig. 2. Syntax for defining hook rules.

events between publishers, subscribers, and other brokers. This integration simplifies information management by grouping security, configuration (e.g. type schema) and recovery tasks for database and pub/sub operations under the same interface.

In our model, events consist of a set of attribute-name/typed-data-value pairs. Event types have a system-wide name and a schema, to allow strong type-checking of event instances and subscriptions at runtime. Events are transmitted in XML, providing a common, human-readable standard for communication. Subscription filters take the form of arbitrary SQL conditional statements allowing powerful and fine-grained conditional clauses that may reference both context and stored data through built-in and user-defined functions.

C. Hook Rules

Hook rules allow integration of information sharing policy into the pub/sub middleware. A *hook rule* is an active rule evaluated by the database system at a specific point of the messaging process: *ON SUBSCRIBE*, *ON PUBLISH*, or *ON NOTIFY*. An ON SUBSCRIBE rule is evaluated when a broker receives a subscription request. An ON PUBLISH rule is evaluated when a broker receives an event from a publisher. An ON NOTIFY rule is evaluated when an event type matches a subscription. This occurs before application of the subscription filter determining message delivery.

A hook rule definition includes a name, an interaction point, a condition, and a function. The name uniquely identifies the rule at the local broker. The interaction point (ON *interaction point*) defines when the rule is to be evaluated. The ON PUBLISH hook references an event type, while the ON NOTIFY hook references an event type and a user whose subscription is subject to the policy. The condition, like a subscription filter, is defined as a SQL conditional statement. When a hook rule is executed the function is called, with aspects of messaging system context supplied as arguments.

As shown in Table I, the contextual inputs and return values of the function vary according to the interaction point. The results of the executed functions replace the original occurrence, allowing functions to modify subscriptions or events. Thus, the subscription returned from the ON SUBSCRIBE function is the one registered in the system, the event returned by an ON PUBLISH function is matched against active subscriptions and the event returned by an ON NOTIFY hook is delivered, subject to any filters, to the particular subscriber.

At an interaction point, a single occurrence may activate several hook rules. The middleware allows an administrator to detect and resolve conflicts through a specified function. This is passed the set of active hook rules, and returns an ordered set of those applicable in the current context. The functions for these rules are executed in order on the original event; the results introduced to the next phase of the pub/sub process.

Interaction point	Context	Return value
ON SUBSCRIBE	subscription, user	subscription
ON PUBLISH	event, user (publisher)	event
ON NOTIFY	event, user (subscriber)	event

TABLE I
CONTEXTUAL INPUTS AND RETURN VALUES OF HOOK RULE FUNCTIONS.

D. Middleware Summary

This section describes the mechanisms that bring application level considerations into the messaging system. Event types define particular semantics for event instances (messages), which together with subscriber information (credentials) establish the application context. SQL-based conditionals support powerful definitions of state, while hook rules provide an interface to interact with pub/sub messaging. This middleware forms the basis of our information control model.

IV. INFORMATION CONTROL MODEL - FUNCTIONALITY.

The basic pub/sub paradigm supports information exchange between anonymous parties. Information flow in environments such as healthcare must be controlled. Our model allows control of data from the source, through a broker that releases information according to the policy of its local domain [10]. A broker may, in the same manner as other entities, subscribe to events occurring at another broker, in order to replicate data locally, or to forward information actively to other subscribed parties. In this model, communication occurs directly between the broker and the subscriber, so that information dissemination is subject to the broker's data disclosure policy.

A. Information Control

Our model allows active control over information dissemination through policy definitions that: 1) transform events as appropriate to circumstance; 2) control (subscriber) access to events. Both mechanisms use conditional clauses to define the criteria for policy activation. As such, policies reference *context*, considering system and environmental (real-world) state through event content, subscriber specifics, credentials, stored data and the results of functions.

1) Data transformation: Transformation allows information to be tailored to circumstance, altering an event as it moves through the system. A conditional clause states the circumstances in which a transformation occurs. Events are transformed through invocation of a function, which may alter attribute values in an event instance, by nullifying fields, performing calculations/conversions, or bucketing values; or it may convert an event into another type. Type conversion may include value transformations, and may enrich an event (e.g. by adding attributes, or other related information), restrict information by degrading an event (e.g. hiding sensitive attributes), or create a new event instance that is only loosely related. A schema service provides users with event type definitions.

As illustrated in Figure 3, policies may define transformations either at publication time or on notification.

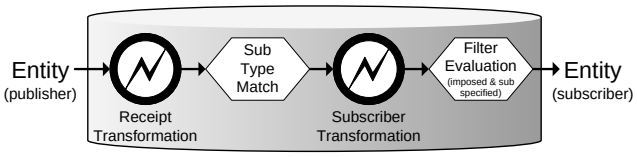

Fig. 3. Points of definition for transformation policies.

Transformation on receipt. Policy may specify a transformation, in certain conditions, on publication of an event; the result is introduced into the system as a new event instance[1]. If there are multiple transformation policies for a given type, then a single published event is converted into many.

Transformation on notification. This transforms an event instance before delivery to the subscriber. This allows customisation of an event to a particular subscriber. Here, policy refers to an event type, a set of credentials and a condition.

Applying transformations. Transformation on receipt is useful for interoperability, to convert events into a form more suitable for local processing. Where the result of a transform is likely to interest several parties, application at this stage improves efficiency – performing a single transform regardless of the number of subscribers. Type transformations are more naturally performed on event receipt[2] This is because an event type embodies a particular semantic, so that subscriptions more accurately represent an interest if they reference the output type directly. Transformation on notification is appropriate for policy applicable to a *specific* group of subscribers, especially for events occurring infrequently or with few subscribers.

The appropriate interaction point for a transformation is a question of design, depending upon the scenario. Suppose that a notification policy entails nullifying a value for Dr. Nick when an event concerns Patient Y. Effecting this through a receipt transformation would require an event type specific to the doctor, thus revealing the existence of such a policy. Prescribing a drug involves recording symptoms, yet the pharmacy and billing agency require only a legal prescription, with observation details removed. Here it is sensible to convert the prescribing event on receipt into a prescription, which is forwarded to those who subscribe specifically to prescription events. Modelling this scenario through notification policies is cumbersome, as those interested in prescriptions must subscribe to the general prescribing event, and the prescripiton transformation function is executed for each subscriber.

2) Access control - Subscriber Specific Policies: Each broker requires users to be authorised to publish or to subscribe to instances of an event type. For reasons of management and scalability, our data control layer allows users to hold, and policy to reference, sets of credentials. From now on, we shall use the term *subscriber* to refer to a subscribed user holding a particular credential set. Our model allows policy to impose credential-based conditions to control event dissemination, or to tailor an event to suit the subscriber and circumstances.

Restrictions - Condition Imposition. Conditions define circumstances in which a subscriber may receive an event instance. Imposed conditions are similar to subscription filters, setting prerequisites for message delivery; but take precedence over subscriber preference. Such conditions may reference event content, context, environmental state, etc., and might enforce a relationship – for example, where a Doctor receives notifications only for patients that they manage.

Imposed conditions are absolute, in the sense that an event instance is delivered *only* if all the conditions are satisfied. As the policy can itself contain sensitive information, impositions are neither revealed to a subscriber, nor do they prevent a subscription if they are in conflict with a subscriber's filter. For example, policy stating that Dr. Nick cannot receive treatment information about Jill Smith where `disease='HIV'`, suggests that Jill is HIV positive. A subscription is denied only when no policy authorises access to the event type; otherwise it is accepted, with conditions imposed on event delivery.

Subscriber specific transformations. Notification transformations allow fine-grained control over the information released to particular subscribers. These occur when an event instance matches the defined transformation conditions and the event type matches a subscription. Evaluation of filters (imposed and subscriber-specified) is performed *after* a transformation, to ensure that the output adheres to any restrictions.

While imposed conditions are useful for absolute restrictions, a subscriber transform can also represent access control policies (grant/deny), to allow resolution with other transformations. For example, policy stating that doctors should not receive information for Patient Y, unless a specific privacy transform has been executed, can be modelled as two transformation policies: P1 to enforce the denial, and P2 to perform the transform. Since both policies are transformations, they can be resolved to give the desired effect – i.e. P1 denies the event, unless both policies are active, in which case P2 overrides P1, and the transform is performed. Conflict and resolution is discussed in detail in Section VI.

B. Broker functionality - Scenario

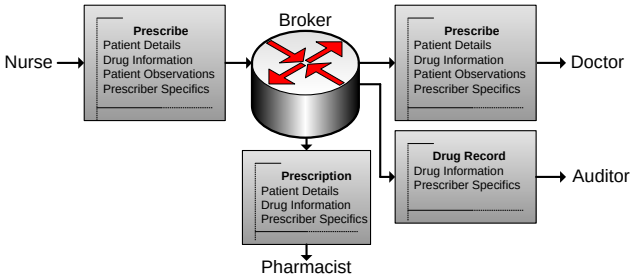

Fig. 4. Transformations for the publication of a Prescribe event.

Figure 4 shows a broker tailoring information to circumstance. Data is released on a *need-to-know* basis, where for the publication of a Prescribe event, the Doctor receives the complete event, the Pharmacist receives a valid Prescription

[1]Whether the original event is propagated depends upon the policy set.

[2]However, there are instances suited to notification type transforms. Thus, a schema service provides the types and structures possible for a subscription.

with the medical notes removed, and the Auditor receives the details of the drug and the prescriber without patient specifics.

V. Policy Definition

Policy is central to our information control model, defining the circumstances in which the data control mechanisms apply. As this model is built in a database environment, policy itself is stored as data; where event types, credentials, functions (transformations) and conditions are represented in relational tables. This allows the use of query languages to simplify the process of policy activation, conflict detection and resolution. Further, policies are subject to relational constraints, and may be defined, modified and revoked in a transactional manner to ensure a consistent policy set.

Although information control policy is represented as data rows, it is specified in XML. This is because XML is easily understood, sufficiently expressive for defining data control specifics and is a standard endorsed by the NHS. Further, it is the messaging format of the pub/sub service, meaning that policy can be dynamically updated through messages. XML policy specifications are parsed, and if valid, converted into the relevant data representation. This eases the authoring process, allowing policy to be analysed before it is committed.

Policy definitions may reference database loaded functions. Functions used within a conditional clause may help to determine state, e.g. whether a relationship exists between a subscriber and event content. Transformation functions are defined in XML, and converted into database functions after parsing and validation. They alter the content of an event, and may invoke other functions during the transformation process.

An organisation might have a rule that a doctor can only receive information about patients that they treat. Policy imposing this relationship is represented in Figure 5 a), through a function taking as arguments the subscriber, i.e. the `user` as derived from the messaging context, and the value of `patient` from the event instance. Figure 5 b) shows (partial) policy for creating a `prescription` from a `prescribe` event. The policy specifies the conditions for execution (here in all circumstances), and includes a `publish` block defining the transformation. The policy fragment shown in Figure 5 c) defines a subscriber specific transformation for an auditor, applying a mapping function to nullify patient specific attributes for `prescribe` events that reference controlled drugs Mapping functions are named, and thus may be referenced by a number of policies, at various interaction points. For convenience, unless otherwise specified, where attributes have an identical name and type, values are automatically copied from the input to the output event type.

VI. Policy conflicts

Data control policies apply to an event type, credentials (for subscriber-specific policies) and conditions that reference aspects of state. Policies *conflict* when multiple policies become active (apply) at an interaction point. Conflicts may be *static*, where the policy definitions overlap, or *dynamic*, where a conflict arises as a result of circumstances at run-time.

```
<subscriber_restriction>
  <event_type>prescribe</event_type>
  <credentials>doctor</credentials>
  <restriction>treatsPatient(user, patient)</restriction>
</subscriber_restriction>
```

a) Policy for imposing a condition on a subscriber.

```
<receipt_transform>
  <event_type>prescribe</event_type>
  <condition></condition>
  <publish output_type="prescription">
    <field id="patient_info">getDemographics(patient)</field>
    <field id="drug_name">drug</field>
    <field id="prescriber">getStaffDetails(staffid)</field>
    ...
  </publish>
<receipt_transform>
```

b) Policy transforming an event upon receipt.

```
<mapping_function>
  <name>remove_patient_details</name>
  <input_type>prescribe</input_type>
  <publish output_type="prescribe">
    <field id="patient"></field>
    <field id="observations"></field>
  ...
</mapping_function>
<subscriber_transform>
  <event_type>prescribe</event_type>
  <credential>drugauditor</credential>
  <condition>isControlledDrug(drug)</condition>
  <mapping>remove_patient_details</mapping>
</subscriber_transform>
```

c) A subscriber specific transformation utilising a mapping function.

Fig. 5. Sample policy fragments.

A. Detecting potential conflicts

To assist policy administrators in maintaining a consistent policy set, conflicts are detected through a combination of queries and code that analyse and compare policy definitions. **Receipt transformation conflicts.** Conflicts occur when several transformation policies are applicable on event receipt. At this stage, applying multiple policies is often appropriate, particularly where the output types differ. Each transformation produces an event with its own semantics, which is matched against a subscription specific to the (output) type – an example being the prescribing scenario. Care must be taken if policies that produce events of the *same* type are applied simultaneously, because the outputs may match (subject to filters) a single subscription. Whether this is appropriate depends on the scenario, but it is important not to mislead subscribers.

Receipt transformation policies are defined with conditions that reference event attributes, functions and/or context. On receipt, conflicts occur where multiple policies are defined for the same input (received) and output (transformed) types. Policies conflict statically if the activation conditions overlap in definition, otherwise such policies conflict dynamically.

Subscriber policy conflicts. Policy rules relating to subscribers are defined for an event type, a credential set and an imposed or transformation condition. This makes policy more specific, but it also provides more opportunity for conflict, given that subscribers will often hold multiple credentials.

In this scenario, the first step in detecting potential conflicts is to determine whether a particular credential set activates

multiple policies for an event type. This involves parsing out the credentials and associated operators (`AND, OR, NOT`) from the policy definitions, in order to determine the combinations of credentials that may cause policies to conflict. Credential definitions conflict statically when there is direct overlap. For example, a policy defined for a `Doctor` and another for `Doctor or Nurse` conflict statically, as a subscriber holding the `Doctor` credential activates both. Dynamic conflicts arise because of the credentials held by a particular subscriber. For instance, if policy is defined for a `Nurse` and another for a `Head Nurse`, these policies are both activated when a subscriber holds both credentials.

Subscriber specific policies define a condition, which serves either to impose a restriction (filter) on a subscription or to state the conditions in which to perform a transformation. In the same way as for receipt transformations, determining static or dynamic conflicts for policies with conflicting credential definitions involves considering the output types (for transformation policies) and the overlap of conditional clauses.

The purpose of detecting potential conflicts is to provide the policy administrator with sufficient information to maintain a consistent policy set. The query capabilities of database systems assist in conflict detection and policy definition processes; for example, maintaining a log of credential activations can provide heuristics for estimating the probability of conflict.

B. Conflict resolution

We stated that policy rules conflict if they are simultaneously activated by a particular set of conditions. However, this definition considers only *possible* conflicts at the system level – *not* the real world aspect. Our model does not itself resolve conflicts, as it is extremely difficult, if not dangerous, to do so in a complex environment such as healthcare. It is argued that application-level conflict resolution is often better addressed by careful policy (re)authoring, over formal and complex resolution strategies [11]. As such, we provide the means for policy administrators to detect and define conflicts and appropriate resolution strategies.

As policy is local to a broker, the administrator can view policy definitions – introducing, overriding or revoking policy rules as appropriate. A function specified by the administrator is passed the set of active transformation rules, and returns an ordered set of those applicable in the current context. Our model provides the following tools to help resolve conflicts at an interaction point:

Ordered Invocation. This allows specification of the order in which transformation functions are executed on the original event. This is illustrated in Figure 6 a), in which R2 is applied before R1.

Overrides. When it is inappropriate for multiple policy rules to apply, for example if both generate the same output type, one rule may be defined as overriding another. When both rules are applicable, the overridden rule will not apply.

Aggregation. Aggregation is a resolution strategy, applied to sets of rules that impose conditions in the context of a single subscription. Aggregation concatenates conditional clauses through boolean operators (`AND,OR,NOT`). First, if several rules apply the same transformation function, it is applied if the condition of any one of the rules is satisfied. This prevents the transformation from being executed more than once, so avoiding duplicate messages. Filters are applied to the results of a transformation. If more than one restriction (imposed condition or subscription filter) applies to a given event type, any boolean combination of the conditions can be specified; the system default action is that the event will be delivered only if all the conditions are satisfied (`AND`).

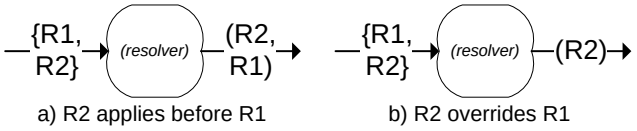

Fig. 6. Conflict resolution strategies.

The conflict detection procedures are implemented as database functions, executable by authorised applications, allowing policy to be tested for conflicts before committal. Policy resolution definitions are represented as data, and thus can be queried as part of the policy design process. Our implementation provides the means for policy authors to ascertain whether the policy set is consistent, allowing the definition and resolution of policy conflicts.

VII. Implementation and Enforcement

Information sharing policy is implemented in the data control layer. This layer interacts with the pub/sub service through hook rules, and an API providing access to messaging facilities, such as event creation and publication. This section describes the process of policy enforcement within a broker.

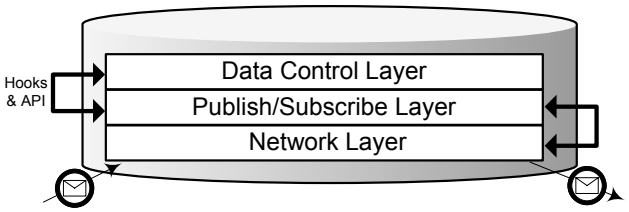

Fig. 7. The middleware layers of a broker.

A. Receipt transformations

Receipt transformation policies are enforced through the use of hook rules. On broker instantiation or when policy changes, ON PUBLISH hook rules are created to inform the pub/sub layer of the policy applicable on receipt of a particular event type. The conditions for policy activation are defined in the conditional segment of the rule. When an event is received, the pub/sub layer determines the set of hook rules that are applicable. This set is passed to a function in the data control layer, which analyses conflict resolution policy to order and override policy rules as appropriate. The resulting ordered set is returned to the pub/sub layer, where the transformation functions for policies remaining in the set are executed in order – the resulting event instances moving through the pub/sub layer for transmission to subscribers.

B. Subscriber policies

The application of subscriber specific policies is a two-phase process. Firstly, upon a request to subscribe to an event type, the data control layer searches for policy applicable to the subscriber's credentials. If no policies are found, the subscription is denied. For each applicable transformation policy, an ON NOTIFY hook rule is created, corresponding to the event type, subscriber, activation conditions and the transformation function defined by the policy. Policies imposing conditions are resolved (overridden and aggregated), producing a filter encapsulating all restrictions relevant to this subscription. Filters are also created, as defined by policy, for each output type of the applicable transformation rules, persisting for the lifetime of subscriptions. This ensures that restrictions are enforced when notification transformations involve a type conversion. As illustrated in Figure 8 a), the subscription process concludes with an acknowledgement.

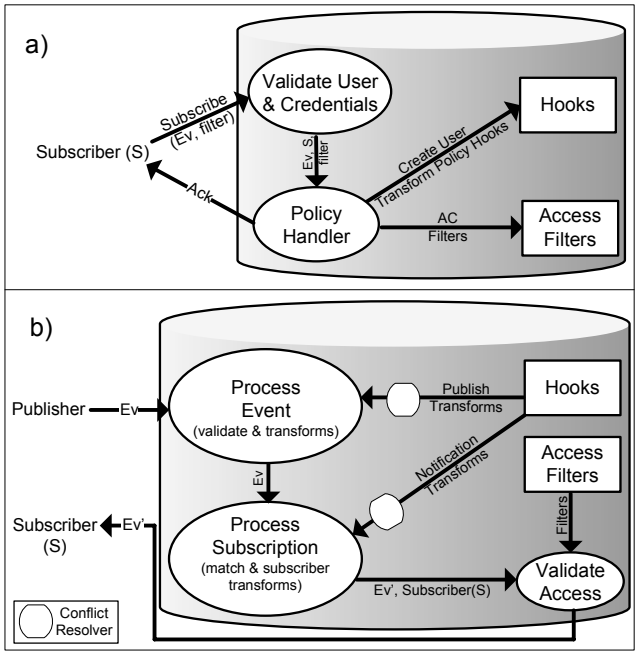

Fig. 8. Process for the enforcement of subscriber policy.

When the type of an event instance matches a subscription, the set of applicable ON NOTIFY rules is determined and passed to the data control layer. In the same way as for a receipt transformation, conflicts are resolved (ordered and overridden), before ordered execution of the transformation functions. The events are delivered to the subscriber if they satisfy the imposed conditions (restrictions) and the subscriber specified filter. Conflict resolution for notification policies does *not* occur upon subscription, because such policies are activated in response to runtime circumstances (event instance and state). Resolving transformation policies at the subscription stage cannot take account of context.

Figure 8 b) illustrates the process of enforcing policy for the publication of an event `Ev`. When an event instance is received at a broker, the applicable ON PUBLISH rules are resolved, and the associated transformations are executed in order. The result event instances are passed to the pub/sub layer to be matched against active subscriptions. In this example, as no receipt transformation policies apply, the original event (`Ev`) moves to the subscription matching phase. After matching a subscription, the applicable (resolved) subscriber specific transformations are applied – in this case, `Ev` is transformed into `Ev'` due to a subscriber specific policy for `S`. The resulting events are then tested against the filters to determine whether to allow delivery. Here the subscriber is authorised to receive `Ev'`, a transformed version of the original publication.

VIII. Discussion and Related Work

Pub/sub systems provide a level of indirection between producers and consumers, where there is generally little information regarding the source or the recipients of events. Healthcare, however, is a domain that requires control. In healthcare, much is known about system users through credentials (e.g. registered with the NHS, licensed with an authority, active doctor on duty). There is also a degree of local autonomy; care providers dealing with medical information are responsible for its protection, and the circumstances in which it is shared. Our model applies local control above the pub/sub layer, with each broker maintaining its own policy set governing the sharing of data. The database environment brings powerful data-handling and transaction facilities, providing a common interface for managing data, policies, types, messaging-specifics and conflict resolution strategies. Policy rules can take account of context, referencing the credentials of recipients, system and environmental state. The model enables information flow to be adapted to circumstances — allowing data dissemination on a *need to know* basis.

Policy models considering the actions of users have been widely researched. Here, we are concerned with building information sharing policy into the messaging architecture. As mentioned, work dealing specifically with data dissemination or control tends to focus either on networking aspects or on access control to data records [12]–[14]. These are particularly useful in request-response (pull) scenarios. Our work complements these by *actively* controlling information flow in a *notification* environment.

Scoping [15] is a pub/sub concept that involves bundling a set of components into a *scope* (group), allowing events to flow freely between its members. Transformations can be defined to translate events automatically as they move between scopes. Such work considers security issues including routing specifics (e.g. tunnelling), and it is relevant to interoperability, in terms of managing data models in heterogeneous environments. However, in many situations members of a common scope will have different data access privileges/requirements. In our approach, information control policy may reference credentials at the granularity of individual users, and event transformations are applied according to policy, outside the pub/sub layer. Our model is compatible with scoping, as scope membership provides a separate credential that can support data interoperability within the pub/sub framework.

There has been some work concerning access control in pub/sub environments. Belokosztolszki et al. [16] integrated a role-based access control model into a pub/sub service to provide type-specific access control. In that work, event types were hierarchical, where if a subscriber lacked the privilege to subscribe to a particular type, rather than deny the request, the subscription could be downgraded to a subtype for which they have access. The work was extended by Bacon et al. [7], [17] to provide attribute-level control. This considered multi-domain environments, encrypting attributes of an event to allow liberal distribution, where sensitive information is only accessible by those holding the appropriate key. The focus was on open networking environments, concerning issues of routing, key management and untrusted brokers. Our work is complementary, providing end-to-end information flow control that can be tailored to a particular application environment. Policy is enforced locally at brokers (local database systems), allowing a degree of autonomy to individual administrative domains. This model is appropriate in a healthcare environment. In addition to access control mechanisms that take account of context, our model provides transformations, allowing policy to customise events (and subscriptions). This customisation might restrict, to protect/hide sensitive information, enrich, providing more detail for a particular subscriber or create some loosely-related event instance. While our model does not preclude the use of an event type hierarchy, we do not require one. Our approach is compatible with encryption models, where a transformation function can encrypt/decrypt (aspects of) a message. Our focus is on local responsibility and control, providing the means to define the content and circumstances (context, recipient credentials) for information release, irrespective of key management concerns.

The recent work of Wun and Jacobsen [18] describes the integration of policy into a pub/sub middleware; it suggests possible applications, and investigates the distribution and performance implications of policy integration. Our work bears similarities, in that policies are executed at specific points of the notification process; however, our aim is different, given our specific focus on information control. Wun describes a *post-matching* policy model, which couples a policy to a specific pub/sub matching operation. Policies are evaluated as part of the existing pub/sub process; i.e. if a subscription is matched, the associated policies are triggered. As policy rules do not have *pre-matching* activation conditions, some of the overhead of policy evaluation is avoided[3]. In our model, policies are conditional, allowing them to be executed either before or after matching operations, with full reference to event content and context. While this incurs an evaluation overhead, it is essential for control – for example, to ensure that notification transforms do not circumvent any imposed or subscriber specified filters. In our application we also need to consider policy conflict, and therefore introduce explicit mechanisms for conflict detection and resolution.

[3]Their model also allows policies to have activation conditions that are evaluated after matching is complete. This imposes an additional overhead.

IX. Conclusion

This paper describes a middleware model to control the sharing of information in a publish/subscribe environment. Our model builds policy into the messaging system, so that information can be tailored to real world circumstances – through reference to recipient credentials, message content and environmental context. We have described how our model allows policy rules to restrict access to information subject to a variety of conditions, and to transform data in various ways. Policy rules are local to each administrative domain; mechanisms are provided to assist administrators to maintain a consistent policy set. This work supports collaboration, a fundamental requirement of the healthcare process, while providing the means for health providers to protect sensitive information for which they are responsible. We are currently working towards updating subscriber-related policy automatically on credential revocation. We also intend to provide rigorous auditing mechanisms in order to track data flow, and so assist in policy authoring and evaluation.

Acknowledgments

EPSRC GR/C53719 CareGrid and Microsoft Research support Jatinder Singh. CONACYT Mexico supports Luis Vargas.

References

[1] J. Singh, J. Bacon, and K. Moody, "Dynamic trust domains for secure, private, technology-assisted living," in *ARES*, 2007, pp. 27–34.
[2] Department of Health (UK), "Safer management of Controlled Drugs," 2007.
[3] British Medical Association and the Royal Pharmaceutical Society of Great Britain, "British National Formulary," September 2007.
[4] UK Crown, "The Controlled Drugs (Supervision of Management and Use) Regulations 2006."
[5] The House of Commons U.K., "The Electronic Patient Record: HC 422-I, Sixth Report of Session 2006-07," September 2007.
[6] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.
[7] J. Bacon, D. M. Eyers, K. Moody, and L. I. W. Pesonen, "Securing publish/subscribe for multi-domain systems," in *Middleware*, 2005, pp. 1–20.
[8] The PostgreSQL Consortium, "PostgreSQL," http://www.postgresql.org.
[9] L. Vargas, J. Bacon, and K. Moody, "Integrating Databases with Publish/Subscribe," in *DEBS*, 2005, pp. 392–397.
[10] J. Singh, L. Vargas, and J. Bacon, "A model for controlling data flow in distributed healthcare environments," in *Pervasive Health*, 2008.
[11] R. Chadha, "A cautionary note about policy conflict resolution," in *MILCOM*, October 2006, pp. 1–8.
[12] Connecting For Health (UK Crown), "Sealed Envelopes Briefing Paper: 'Selective Alerting' Approach," 2006.
[13] R. J. Anderson, "A security policy model for clinical information systems." in *IEEE Symposium on Security and Privacy*, 1996, pp. 30–43.
[14] J. Longstaff, M. Lockyer, and J. Nicholas, "The TEES confidentiality model: an authorisation model for identities and roles," in *SACMAT*, 2003, pp. 125–133.
[15] L. Fiege, A. Zeidler, A. Buchmann, R. Kilian-Kehr, and G. Mühl, "Security aspects in publish/subscribe systems," in *DEBS*, 2004, pp. 44–49.
[16] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody, "Role-based access control for publish/subscribe middleware architectures," in *DEBS*, 2003, pp. 1–8.
[17] L. I. W. Pesonen, D. M. Eyers, and J. Bacon, "A capabilities-based access control architecture for multi-domain publish/subscribe systems," in *SAINT*, 2006, pp. 222–228.
[18] A. Wun and H.-A. Jacobsen, "A policy management framework for content-based publish/subscribe middleware," in *Middleware*, 2007, pp. 368–388.