Data Integration Techniques in Grid Monitoring

Andy Cooke, Alasdair J G Gray, and Werner Nutt

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh, UK

Abstract. Grids are distributed systems that provide access to computational resources in a transparent fashion. Collecting and providing information about the status of the Grid itself is called Grid monitoring. As an approach to this problem, we present the Relational Grid Monitoring Architecture (R-GMA), which tackles Grid monitoring as an information integration problem.
A novel feature of R-GMA is its support for integrating stream data via a simple "local as view" approach. We describe the infrastructure that R-GMA provides for publishing and querying monitoring data. In this context, we discuss the semantics of continuous queries, provide characterisations of query plans, and present an algorithm for computing such plans.
The concepts and mechanisms offered by R-GMA are general and can be applied in other areas where there is a need for publishing and querying information in a distributed fashion.

1 Introduction

In this paper, we discuss data integration techniques for solving the problem of how to monitor the state of a dynamically changing Grid. Grid monitoring requires facilities for publishing data about the computing resources scattered across the Grid. The data comes in two kinds, as static data held in databases or as dynamic data produced as streams. A system that supports Grid monitoring should provide a global view of the data and be able to answer queries about the current state and the history of streams as well as continuous queries, which ask for an answer stream. It needs to be scalable to allow hundreds of nodes to publish, and it must be resilient if any node fails. There are also issues of privacy of data that need to be addressed.

Together with Work Package 3 [5] of the European Union's DataGrid project (2001–2004) [4], we have developed the Relational Grid Monitoring Architecture (R-GMA) as a framework for realising a Grid monitoring system. A distinguishing feature of R-GMA is that it approaches Grid monitoring as a data integration task.

DataGrid's aim was to build and deploy middleware for a Grid that will allow three major user groups to process and analyse the results of their scientific experiments: (i) high energy physicists who want to distribute and analyse the vast amounts of data that will be produced by the Large Hadron Collider at CERN,

(ii) biologists who need to process medical images as part of the exploitation of genomes, and (iii) scientists of the European Space Agency's Earth Observation project who are analysing images of atmospheric ozone.

During the past two years, we have implemented a working R-GMA system within DataGrid. Our aim was to develop functionality that had a firm theoretical basis and that was flexible enough to quickly respond to user requirements as they became clearer. We will describe the status of our implementation at the end of the DataGrid project. R-GMA has an open-source licence and can be downloaded from [5].

In Section 2 we give an overview of Grid computing, outline the requirements of a Grid monitoring system and discuss how far existing Grid monitoring systems meet these requirements. Then, in Section 3, we present an idealised architecture of a data integration system for Grid monitoring, which abstracts from the idiosyncrasies of the actual R-GMA implementation. A distinguishing feature of R-GMA are so-called republisher components, which are defined by a query and that make the answers available for further querying. Republishers allow queries to be answered more efficiently, but make query planning more involved. We define the semantics of stream queries in R-GMA in Section 4, develop characterisations for query plans in the presence of republishers in Section 5, and present an algorithm for computing query plans in Section 6. Finally, we discuss the state of the current implementation and the experiences with it in Section 7. Section 8 concludes.

2 Grid Monitoring: Overview and Requirements

We shall now present the idea of Grid computing and describe the components of a Grid. We explain what is meant by Grid monitoring and identify requirements for a Grid monitoring system.

2.1 Grids

A *Grid* is a collection of connected, geographically distributed computing resources belonging to several different organisations. Typically the resources are a mix of computers, storage devices, network bandwidth and specialised equipment, such as supercomputers or databases. A Grid provides instantaneous access to files, remote computers, software and specialist equipment [7]. From a user's point of view, a Grid is a single virtual supercomputer.

In the late 90s, the concept of a Grid emerged as a new model for pooling computational resources across organisations and making them available in a transparent fashion, using communication networks [7]. Since then, there has been a growing number of projects to construct Grids for different tasks. They include the NASA Information Power Grid [12] and TeraGrid [1] in the US, and CrossGrid [2] and DataGrid [4] in Europe. The Globus group [9] is developing the Globus Toolkit, a suite of middleware components, which are being widely used as a platform for building Grids.

To make a Grid behave as a virtual computer, various components are required that mimic the behaviour of a computer's operating system. The components of DataGrid, and their interactions, can be seen in Fig. 1 and are similar to those presented in [**?**]:

User Interface: allows a human user to submit and track jobs, e.g. "analyse the data from a physics experiment, and store the result".

Resource Broker: controls the submission of jobs, finds suitable available resources and allocates them to the job.

Logging and Bookkeeping: tracks the progress of jobs, informs users when jobs are completed, which resources were used, and how much they will be charged for the job.

Storage Element (SE): provides access to physical storage devices for storing data files.

Replica Catalogue: tracks where data is stored and replicates data files as required.

Computing Element (CE): provides access to a cluster of CPUs, and manages the jobs that run on these.

Monitoring System: monitors the state of the components of the Grid and makes this data available to other components.

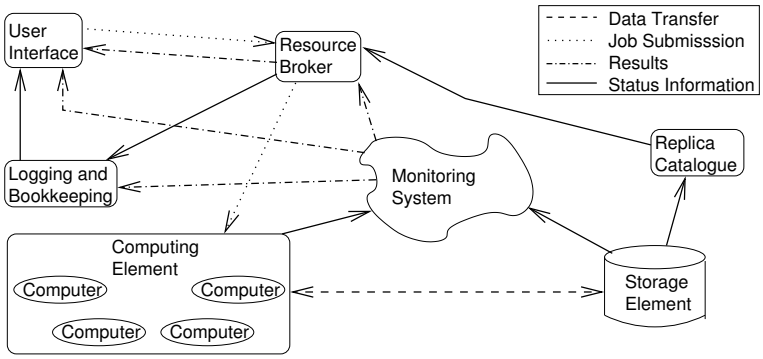

Fig. 1. The major components of DataGrid.

2.2 Grid Monitoring Requirements

The purpose of a Grid monitoring system is to make information about the status of a Grid available to users and to other components of the Grid. As a basis for discussing the requirements that such a system should meet, we consider the following use cases:

1. A resource broker needs to quickly (within 10 seconds) locate a computing element (CE) that has 5 CPUs available, each with at least 200 MB of memory. The CE should have the right software installed, and the user must be authorised to use it. The throughput to an SE needs to be greater than 500 Mbps.
2. A visualisation tool that is used by users to monitor the progress of their jobs needs to be updated whenever the status of a job changes.
3. Network administrators need to interrogate the past state of the network so that typical behaviour can be ascertained and anomalies identified.

Publishing Data. There are many different kinds of information about a Grid, which come from numerous sources. The following are examples:

- Measurements of network throughput, e.g. made by sending a `ping` message across the network and publishing the runtime (use cases 1 and 3 above);
- Job progress statistics, either generated by annotated programs or by a resource broker (use case 2);
- Details about the topologies of the different networks connected (use cases 1 and 3);
- Details about the applications, licences, etc., available at each resource (use case 1).

This monitoring data can be classified into two types based on the frequency with which it changes and depending on the way in which it is queried:

Static data (pools): This is data that does not change regularly or data that does not change for the duration of a query, e.g. data that is being held in a database management system with concurrency control. Typical examples are data about the operating system on a CE, or the total space on an SE (use case 1).

Dynamic data (streams): This is data that can be thought of as continually changing, e.g. the memory usage of a CE (use case 1), or data that leads to new query results as soon as it is available, for example the status of a job (use case 2).

A core requirement, then, of a Grid monitoring system is that it should allow both static and streaming data to be published. The act of publishing involves two tasks: (i) advertising the data that is available, and (ii) answering requests for that data.

Locating Data. Data about Grid components will be scattered across the Grid, and the monitoring system must provide mechanisms for users of the Grid to locate data sources.

In addition, users need a *global view* over these data sources, in order to understand relationships between the data and to query it.

Queries with Different Temporal Characteristics. A monitoring system should support queries posed over data streams, over data pools, or over a mix of these (use case 1).

It should be possible to ask about the state of a stream right now (a *latest-state* query—use case 1), continuously from now on (a *continuous* query—use case 2), or in the past (a *history* query—use case 3).

Up-to-date answers should be returned quickly, e.g. in use case 1 the resource broker requires that the data is no more than a few seconds old. To be accepted by users, the query language should capture most of the common use cases, but should not force a user to learn too many new concepts.

Scalability, Robustness and Performance. A Grid is potentially very large: DataGrid's testbed contains hundreds of resources each producing monitoring information. In the normal use of a Grid, the fabric will be unreliable: network connections will fail and resources will become inaccessible.

It is important that the monitoring system can *scale*. It needs to be able to handle a large number of sources, publishing potentially large amounts of data. Likewise there will be a large number of users of monitoring information, both humans and grid components, who require correct answers in a timely manner. The monitoring system should not become a performance bottleneck for the entire Grid. It should be able to cope with large numbers of queries received at the same time.

The monitoring system itself should be resilient to failure of any of its components, otherwise the whole Grid could fail along with it. The monitoring system cannot have any sort of central control as resources will be contributed by organisations that are independent of each other.

Security. An information source must be able to control who can "see" its data and this must also be respected by the monitoring system. Users should be able to identify themselves so that they can make use of the resources that they are entitled to. Resources should be able to prevent access by users who are not authorised.

2.3 Existing Systems

Several Grid monitoring systems have been developed to this date: AutoPilot [16], CODE [17], and the Monitoring and Discovery Service (MDS) [3], to name some of them. MDS, being part of the Globus Toolkit [9], is the most widely known among these systems.

Monitoring and Discovery Service (MDS). The main components of MDS are *information providers,* which publish monitoring data at Grid locations, and *aggregate directories,* which collect them and make them available for querying. Aggregate directories can be organised in hierarchies, with intermediaries that forward their data to other directories at higher levels.

Also, data is organised hierarchically in a structure that provides a name space, a data model, wire protocols and querying capabilities. MDS exists currently in its third incarnation. Previous versions were based on the LDAP data model and query language. The latest version is based on XML and supports the XPath query language.

Although the hierarchical architecture makes it *scalable*, MDS does not meet other requirements outlined in Section 2.2. Firstly, hierarchical query languages have limitations. For one, the hierarchy must be designed with popular queries in mind. Moreover, there is no support for users who want to relate data from different sections of the hierarchy—they must process these queries themselves.

Secondly, to be able to offer a global view of the Grid to users, a hierarchy of aggregate directories must be set up manually—information providers and intermediary directories need to know which directory further up the hierarchy to register with. The system does not automate this, nor does it recover if any component in the hierarchy fails.

Lastly, MDS only supports latest-state queries with no assurance that the answers are up-to-date. It is claimed that users can create archives of historical information by (i) storing the various latest-state values that have been published via MDS in a database and by (ii) providing an interface to allow the system to access the database. However, this approach would require considerable effort on the side of the user.

3 The R-GMA Approach

The R-GMA approach differs from those discussed before by the fact that it perceives Grid monitoring as a data integration problem. The Grid community has proposed the Grid Monitoring Architecture as a general architecture for a Grid monitoring system. However, this architecture does not specify a data model, nor does it say how queries are to be answered. We have extended this architecture by choosing the relational data model, and by applying ideas that originated in the area of data integration. In this section we present an idealised architecture. While this is guiding the implementation work, the actual R-GMA system as deployed in DataGrid differs from it in several ways. Details of the current implementation of R-GMA can be found in Section 7.

3.1 The Grid Monitoring Architecture

The Grid Monitoring Architecture (GMA) was proposed by Tierney *et al.* [18] and has been recommended by the Global Grid Forum [8] for its scalability. It is a simple architecture comprising three main types of actors:

Producers: Sources of data on the Grid, e.g. a sensor, or a description of a network topology.

Consumers: Users of data available on the Grid, e.g. a resource broker, or a system administrator wanting to find out about the utilisation of a Grid resource.

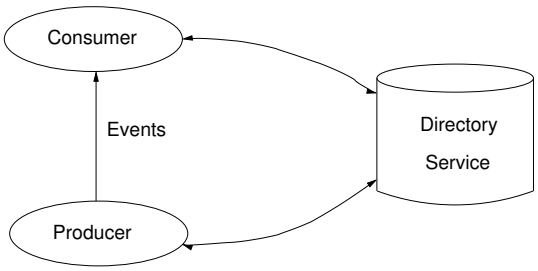

Fig. 2. The components of the GMA and their interactions

Directory Service: A special purpose component that stores details of producers and consumers to allow consumers to locate relevant producers of data.

The interaction of these actors is schematically depicted in Fig. 2. A producer informs the directory service of the kind of data it has to offer. A consumer contacts the directory service to discover which producers have data relevant to its query. A communication link is then set up directly with each producer to acquire data. Consumers may also register with the directory service. This allows new producers to notify any consumers that have relevant queries.

Intermediary components may be set up that consist of both a consumer and a producer. Intermediaries may be used to forward, broadcast, filter, aggregate or archive data from other producers. The intermediary then makes this data available to other consumers from a single point in the Grid.

By separating the tasks of information discovery, enquiry, and publication, the GMA is *scalable*. However, it does not define a data model, query language, or a protocol for data transmission. Nor does it say what information should be stored in the directory service. There are no details of how the directory service should perform the task of matching producers with consumers.

3.2 R-GMA as a Virtual Database

R-GMA builds upon the GMA proposal by choosing the relational data model. Components playing the part of a consumer need to be able to locate and retrieve data of interest (Section 2.2). R-GMA achieves this by presenting a "virtual database" into which all monitoring data appears to flow. As in a real database, the data in the virtual database conforms to a relational schema. It is this *global schema* that allows consumers to locate data of interest.

Consumers describe the monitoring data that they are interested in by posing queries against the global schema. The data is provided by a number of producers who each have a *local schema*. R-GMA uses the idea of a mediator [19] to match a consumer's request for data with the advertisements of data registered by producers.

In order for R-GMA to be able to present the illusion of a virtual database, extra components are needed. These components, along with their interactions, are shown in Fig. 3. In the rest of this section, we shall introduce each component, and explain the rationale behind its design: consumers, producers, consumer and producer agents, schema, republishers and registry.

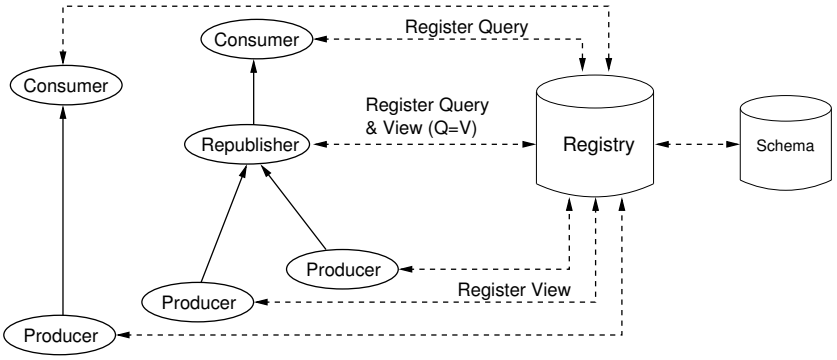

Fig. 3. Components of R-GMA

3.3 Roles and Agents

R-GMA takes up the consumer and producer metaphors of the GMA and refines them. An R-GMA installation allows clients, which may be Grid components or applications running on the Grid, to *play the role* of an information *producer* or a *consumer*.

Producers. In order that both data pools and streams can be published, two producer roles should be supported: a *database producer* and a *stream producer*. A database producer publishes a collection of relations maintained in a relational database. A stream producer publishes a collection of streams, each of which complies with the schema of a specific relation. We refer to these static or streamed relations as the *local relations* of a producer.

A producer advertises its local relations by describing them as simple views on the global schema. In the current implementation of R-GMA, the views can only be selections.

Consumers. A consumer is defined by a relational query. If the query is posed over stream relations, then the consumer has to declare whether it is to be interpreted as a continuous, a history or a latest-state query (see Section 2.2). Once the execution of a continuous query has started, the consumer receives the answer as a stream of tuples.

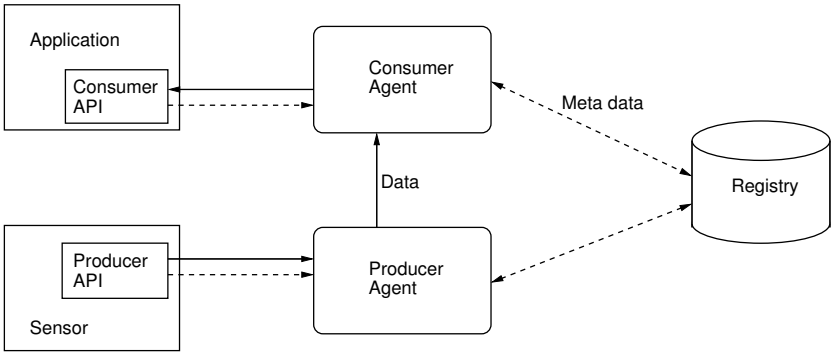

Fig. 4. Roles and agents of R-GMA.

Agents. R-GMA provides *agents* that help clients to play their roles. The interactions of producers and consumers with their agents is illustrated in Fig 4. To play a role, an application uses an API, which in turn communicates with a remote agent. All of the functionality required to play the role is provided by the agent. Agents are realised using Java servlet technology, and are hosted in web servers. Details are discussed in Section 7.

3.4 The Schema

To interact with each other, producers and consumers need a common *language* and *vocabulary*, in which producers can describe the information they supply and consumers the information for which they have a demand. In R-GMA, both the language for announcing supply and the one for specifying demand—that is, the query language—are essentially fragments of SQL. The vocabulary consists of relations and attributes that make up a global schema, which is stored in R-GMA's *schema* component.

Ideally the global schema distinguishes between two kinds of relations, *static* and *stream* relations. The two sets are disjoint. The global schema contains a core of relations that exist during the entire lifetime of an installation. In addition, producers can introduce new relations to describe their data, and withdraw them again if they stop publishing.

The attributes of a relation have types as in SQL. In addition to its declared attributes, every stream relation has an additional attribute timestamp, which is of a type DateTime and records the the time a tuple was published.

For both kinds of relations, a subset of the attributes can be singled out as the *primary key*. Primary keys are interpreted as usual: if two tuples agree on the key attributes and the timestamp, they must also agree on the remaining attributes. However, since data are published by independent producers, the constraint cannot be enforced.

For stream relations, the keys play an additional semantic role. The key attributes specify the parameters of a reading, i.e. they identify "where" and

"how" a reading was taken. The rest of the attributes, except the timestamp, are the *measurement attributes*, i.e. the attributes that state "what" the current reading is.

For instance, R-GMA's schema contains the core relation tp for publishing readings of the throughput of network links. The relation has the schema

$$\mathsf{tp}(\mathsf{from}, \mathsf{to}, \mathsf{psize}, \mathsf{tool}, \mathsf{latency}, \mathsf{timestamp}),$$

which records the time it took (according to some particular tool) to transport packets of a specific size from one node to another. All attributes except latency make up the primary key of tp.

Intuitively, a specific set of values for the key attributes of a stream relation identify a *channel* along which measurements are communicated. For example, for the tp relation with the tuple

$$(\texttt{'hw'}, \texttt{'ral'}, \texttt{'ping'}, 256, 93, \texttt{2004-03-17 14:12:35}),$$

measuring a latency of 93 ms for a 256 byte ping message between Heriot-Watt University and Rutherford Appleton Laboratories on Wednesday 17 March 2004 at 2:12 pm, the channel is identified by the values

$$(\texttt{'hw'}, \texttt{'ral'}, \texttt{'ping'}, 256).$$

Consumers pose queries over the global schema. Similarly, producers describe their local relations as views on the global schema. In Halevy's terminology [11], this means that R-GMA takes a "local as view" approach to data integration.

3.5 Producers and Consumers: Semantics

At present, R-GMA requires that producers declare their content using views *without projections*. Thus, each producer contributes a set of tuples to each global relation. This allows us to give an intuitive semantics to an R-GMA installation: a static relation is interpreted as the union of the contributions published by the database producers; a stream relation is interpreted as a global stream obtained by merging the streams of all the stream producers.

A *static query* is interpreted over the collection of all static relations, while a *continuous query* is conceptually posed over the virtual global stream. An *history query* refers to all tuples that have ever been published in the stream. Finally, a *latest-state query* posed at time t_0 refers to the set of tuples obtained by choosing from each active channel the last tuple published before or at time t_0.

Actually, the semantics of stream relations is not as well-defined as it may seem because it does not specify an order for the tuples in the global stream. We do not guarantee a specific order on the entire global stream. However, we require that global streams are *weakly ordered*, that is, for a given channel the order of tuples in the global stream is consistent with the timestamps. This property ensures that aggregation queries on streams that group tuples according to channels have a well-defined semantics. As we shall see later on, it also

facilitates switching between query plans. We explain in Sections 5 and 6 how one can enforce this constraint.

We are aware that our semantics of stream relations causes difficulties for some kinds of queries, for instance, aggregate queries over sliding windows where the set of grouping attributes is a strict subset of the keys. In such a case, different orderings of a stream can give rise to different query answers. We have not yet dealt with this issue.

Among the three temporal interpretations of stream queries, only continuous queries are supported by default by a stream producer agent. However, when a stream producer is created, the agent can be instructed to maintain a pool with the history and/or the latest-state of the stream. This would enable it to answer queries of the respective type. The creation of these pools is optional because their maintenance will impact on the performance of the stream producer agent.

3.6 Republishers

Republishers in R-GMA resemble materialised views in a database system. A republisher is defined by one or more queries over the global schema and publishes the answers to those queries. The queries either have to be all continuous or all one-time queries. Republishers correspond also to the intermediaries in the GMA. Their main usage is to reduce the cost of certain query types, like continuous queries over streams, or to set up an infrastructure that enables queries of that type in the first place, like latest-state or history queries.

A republisher combines the characteristics of a consumer and a producer. Due to the redundancy of information created by republishers, there are often several possibilities to answer a query. Section 6 describes how this is taken into account in the construction of query execution plans for simple stream queries.

In principle, two types of republisher are conceivable, corresponding to the distinction between static and stream relations. However, the current implementation of R-GMA supports only stream republishers.

Stream Republishers. Stream republishers pose a continuous query and output the answer stream. All stream republishers can answer continuous queries. In addition, similar to a stream producer agent, a stream republisher agent can be configured to maintain also a pool of latest-state values or a history so that it can answer also latest-state and history queries.

Since both input and output are streams, one can build *hierarchies* of stream republishers over several levels. An important usage for such hierarchies is to bundle small flows of data into larger ones and thus reduce communication cost.

Stream producers often publish data obtained from sensors, such as the throughput of a network link measured with a specific tool. While such primary flows of data, to elaborate on the metaphor, tend to be trickles, with stream republishers they can be combined into streams proper. For instance, stream republishers may be used to first collect data about the network traffic from one site and then, at the next level up, between the sites belonging to an

entire organisation participating in a Grid. Thus a consumer asking for network throughput on all links from a particular site need only contact the republisher for that site or for the organisation instead of all the individual stream producers.

Database Republishers. A database republisher will pose a one-time query at a set of published databases and make the answer available as a *materialised view*. This is useful for pre-computing union and join queries where the source relations are distributed across the Grid.

Applications of R-GMA in DataGrid have shown a clear need for this functionality. It has been met so far by periodically publishing static relations as streams and by using a stream republisher with latest-state pool to collect the union of those relations. This approach is bound to become unfeasible as the size of applications increases. Instead, view maintenance techniques will be needed to propagate only changes of the underlying data instead of the full data sets.

3.7 The Registry

We refer to producers and republishers together as *publishers*. Consumer agents need to find publishers that can contribute to answering their query. This is facilitated by R-GMA's *registry*, which records all publishers and consumers that exist at any given point in time. Publishers and consumers send a heartbeat to the registry at predefined intervals to maintain their registration.

When a new publisher is created, its agent contacts the registry to inform it about the type of that publisher and, if it is a stream publisher, whether it maintains latest-state or history pools. If the publisher is a producer, the agent registers its local relations together with the views on the global schema that describe their content. If it is a republisher, the agent registers its queries. Similarly, when a consumer is created, the consumer's agent contacts the registry with the consumer's query.

The registry cooperates with the consumer agent in constructing a query plan. It identifies publishers that can contribute to the answers of that query, called the *relevant* publishers. In the current implementation, the registry informs the agent of *all* relevant publishers. Due to the existence of republishers, there may be some redundancy among the relevant publishers. In the future, the registry may exploit this to reduce the amount of information sent to the agent. It could do so by informing the agent only of the publishers that contribute maximally to the query, called *maximal* relevant publishers, while ignoring those that are subsumed by a maximal one. Based on the list of publishers it has received, the agent constructs a query plan. (In Section 6 we discuss how to choose maximal relevant publishers and how to create plans for the case of continuous selection queries over streams.)

When a consumer registers a continuous query, R-GMA ensures that during the entire lifetime of the consumer it can receive all the data the query asks for. At present, whenever a new producer registers, the registry identifies the consumers to which this producer is relevant and notifies their agents. Then the

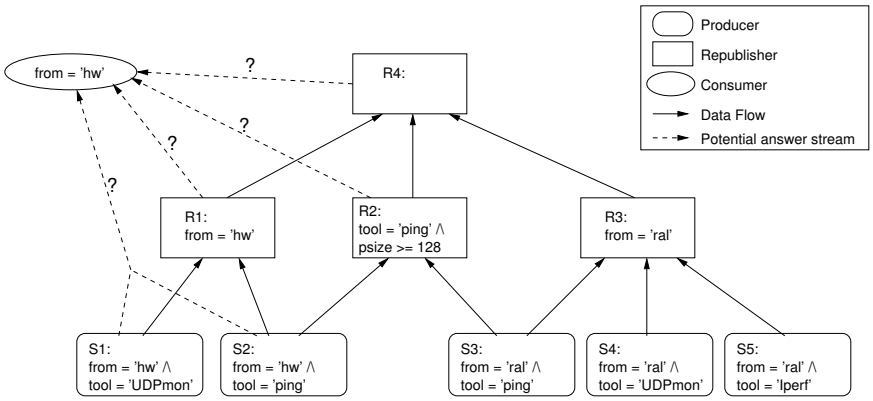

Fig. 5. A hierarchy of stream republishers for `tp`.

agent integrates the new producer into its query plan. A refined approach would consist in informing a consumer only about a new producer if the producer is not subsumed by a relevant republisher (see Section 6 for further details). Consumer agents need to be informed as well when a republisher goes offline because then the consumer may miss data that it has received via that republisher. Similarly, the registry has to contact a consumer agent if a new relevant republisher is created and when a producer goes offline.

4 A Formal Model of Stream Queries in R-GMA

Most often, applications seeking information require the latest state or the history of several stream relations to be joined and aggregated. As an example, consider the needs of the resource broker (Section 2.2, use case 1). Such a query can be answered efficiently by setting up a hierarchy of stream republishers to collect the data needed. The user query can then be answered by the top level republisher, with the help of a DBMS that holds a pool of latest-state or history values.

For the monitoring applications we have encountered so far it is sufficient if consumers and republishers pose simple continuous queries that are selections over a single relation. Although this is a very restricted form of query, its semantics in a data integration scenario like R-GMA's is not straightforward.

In this and the subsequent sections we introduce a formalism to define the meaning of simple stream queries that are posed against a global schema, while data are provided by stream producers and republishers. Since data can only be obtained from publishers, a global query has to be translated into a plan, that is, a query over the local stream relations of the publishers. Part of the difficulty of the problem stems from the fact that a republisher offers a stream of data, but at the same time has to run a plan to acquire its data.

Fig. 5 shows a consumer, producers, and a hierarchy of republishers for the throughput relation `tp`. For each component, the query or descriptive view, re-

spectively, is indicated by a condition involving the attributes of tp. The bold lines leading to each republisher indicate how data can flow through the hierarchy, while the dashed lines leading to the consumer represent publishers that are relevant to the consumer query. We will refer to the situation depicted in the figure to illustrate the query plans that our techniques will generate.

In the present section we define a formal framework for publishing and querying distributed streams with the help of a global schema. This will be used in Section 5 to develop characterisations of query plans. In Section 6 we discuss how to compute query plans for consumers and republishers.

4.1 Streams and Their Properties

We formalise data streams as finite or infinite sequences of tuples. To capture the idea that a stream consists of readings, each of which is taken at a specific point in time, we assume that one attribute of each stream tuple is a timestamp.

More precisely, suppose that T is the set of all tuples. Then a stream s is a partial function from the natural numbers $\mathbb{N}$ to T,

$$s \colon \mathbb{N} \hookrightarrow T,$$

such that, if $s(n)$ defined for some $n \in \mathbb{N}$, the tuple $s(m)$ is defined for all $m < n$. Thus, $s(n)$ denotes the n^{th} tuple of s. We write $s(n) = \perp$ if the n^{th} tuple of s is undefined. A special case is the empty stream, also denoted as $\perp$, which is undefined for every $n \in \mathbb{N}$.

We have chosen to model data streams in this way as it allows different tuples to have the same timestamp and tuples to arrive in an order independent of their timestamp. Thus, we have no requirements about how regularly a reading can be taken nor do we require that readings are published in chronological order.

Suppose r is a relation with a relation schema, specifying a type for each attribute. We define as usual when a tuple satisfies the schema. A stream satisfies the schema if all its tuples satisfy it. We assume from now on that a relation schema is declared for every stream and that the stream satisfies its *local* schema.

Properties of Data Streams. As in Section 3, we assume that the attributes of a stream relation are split into three parts: key attributes, measurement attributes and the timestamp.

We specify the following shorthands for the subtuples of $s(n)$ relating to these three parts:

$s^{\kappa}(n)$ for the values of the key attributes;
$s^{\mu}(n)$ for the values of the measurement attributes;
$s^{\tau}(n)$ for the timestamp of $s(n)$.

We use this notation to formalise the channels of a stream. We say that a stream s_1 is a *substream* of s_2 if s_1 can be obtained from s_2 by deleting zero or more tuples from s_2. A *channel* of s is a maximal substream whose tuples

agree on the key attributes of s. For every tuple t occurring in s, where t^κ is the subtuple of t that contains the values of the key attributes, the substream of s consisting of the tuples with $s^\kappa(n) = t^\kappa$ is the *channel* of t^κ.

The following properties of streams are central to our discussion of the semantics of stream queries.

Duplicate Freeness: A stream s is *duplicate free* if for all m, n with $m \neq n$ we have that $s(m) \neq s(n)$, that is, if no tuple occurs twice in s.

Weak Order: A stream s is *weakly ordered* if for all m, n with $s^\kappa(m) = s^\kappa(n)$ and $m < n$ we have that $s^\tau(m) < s^\tau(n)$. This means that in every channel of s, tuples appear in the order of their timestamps. Note that this definition is equivalent to requiring that for all m, n with $s^\kappa(m) = s^\kappa(n)$ and $s^\tau(m) < s^\tau(n)$ we have that $m < n$.

Disjointness: Two streams s_1 and s_2 are *disjoint* if for all m, n we have that $s_1(m) \neq s_2(n)$, that is, if s_1 and s_2 have no tuples in common.

Operations on Streams. We define two simple operations on streams. Let s be a stream and suppose that C is a condition involving attributes of the schema of s, constants, operators "=", "$\leq$", "$\geq$", and boolean connectives. Then the *selection* $\sigma_C(s)$ of s is the stream that consists of the tuples in s that satisfy C where those tuples appear in the same order as they do in s.

Let $s_1, \ldots, s_n$ be streams for relations with compatible schemas. A stream s is a *union* of $s_1, \ldots, s_n$ if s can be obtained by merging these streams, i.e., if each s_i contributes all its tuples to s, and the tuples of s_i occur in s in the same order as they do in s_i.

Note that the result of a selection is unique, while this is not the case for a union. Note also that (i) streams resulting from these operations are weakly ordered if the argument streams are, and that (ii) the result of a union is duplicate free it the argument streams are mutually disjoint.

4.2 Stream Producers

A *stream producer* is a component that is capable of producing a data stream. Every stream producer has a local relation schema. We denote both, stream producers and their local relations, with the letter S.

Local Queries over Stream Producers. We want to pose queries over stream producers. We call such queries *local queries* as opposed to *global queries,* which are posed over a global schema.

The queries we consider are unions of selections of the form

$$Q = \sigma_{C_1}(S_1) \uplus \ldots \uplus \sigma_{C_m}(S_m), \tag{1}$$

where $S_1, \ldots, S_m$ are distinct stream producers whose schemas are mutually compatible. A special case is the empty union, written ε.

To define the semantics of such a query, we have to associate a stream to each producer. A *stream assignment* over a set of producers is a mapping $\mathcal{I}$ that associates to each producer S a stream $S^{\mathcal{I}}$ that is compatible with the schema of S. A stream s is an *answer* for Q w.r.t. $\mathcal{I}$ if s is a union of the selections $\sigma_{C_1}(S_1^{\mathcal{I}}), \ldots, \sigma_{C_m}(S_m^{\mathcal{I}})$. We remind the reader that an answer is not uniquely defined, since there is more than one way to merge the selections $\sigma_{C_i}(S_i^{\mathcal{I}})$. The empty union ε has only one answer, namely the empty stream $\perp$.

Producer Configurations. We want to formalise collections of stream producers as they can be created in R-GMA. We assume there is a *global schema* $\mathcal{G}$, which is a collection of stream relations. A *producer configuration* consists of a finite set $\mathcal{S}$ of stream producers and a mapping v that associates to each producer $S \in \mathcal{S}$ a query v_S over the global schema $\mathcal{G}$ such that v_S is compatible with the schema of S. If no confusion arises we will also denote the producer configuration with the letter $\mathcal{S}$. In R-GMA, producer configurations are represented in the schema and the registry.

We call v_S the *descriptive view* of the producer S. In this paper we limit ourselves to descriptive views that are selections, that is, they have the form $\sigma_D(r)$ where D is a condition and r is a global relation.

To keep things simple we require, if S is described by the view $\sigma_D(r)$, that S and r have the same attributes and the same type and key constraints. We also require that the condition D in $\sigma_D(r)$ involves *only key attributes* of r. Thus, the view restricts the channels of a producer, but not the possible measurements of the readings.

Instances of Producer Configurations. A producer configuration is similar to a database schema. It contains declarations and constraints, but no data. We want to define which streams are the possible instances of such a configuration.

We say that a stream s is *sound* w.r.t. a query $\sigma_D(r)$ over the global schema if the schema of s is compatible with the schema of r and if every tuple $s(n)$ satisfies the view condition D.

An assignment $\mathcal{I}$ for the producers in a configuration $\mathcal{S}$ is an *instance* of $\mathcal{S}$ if for every $S \in \mathcal{S}$ the stream $S^{\mathcal{I}}$ is (i) sound w.r.t. the descriptive view $v(S)$, (ii) duplicate free and (iii) weakly ordered and if, moreover, (iv) distinct producers have disjoint streams.

4.3 Global Queries and Query Plans

Consumer components in R-GMA pose queries over the global schema and receive a stream of answers. The only queries over the global schema that we consider in this section are selections of the form

$$q = \sigma_C(r),$$

where r is a global relation.[1] Since the relation r does not refer to an existing stream it is not straightforward what the answer to such a query should be.

Intuitively, we understand that query q is posed against a virtual stream, made up of all the small streams contributed by the producers. We say that a producer S *produces for* the relation r if S is described by a view over r. If $\mathcal{I}$ is a producer instance then an *answer* for q w.r.t. $\mathcal{I}$ is a duplicate free and weakly ordered stream that consists of those tuples satisfying C that occur in streams $S^{\mathcal{I}}$ of producers S which produce for r. Note that, according to our definition, there can be infinitely many different answer streams for a query q. Any two answer streams consist of the same tuples, but differ regarding the order in which they appear.

Note also that we have not postulated that tuples occur in the same order as in the original producer streams. We only require that the tuples of a channel appear in the same order as in the stream of the publishing stream producer. This makes it possible for streams to be split and then re-merged during processing.

Since global queries cannot be answered directly, they need to be translated into local queries. We say that a local query Q is a *plan* for a global query q if for every producer instance $\mathcal{I}$ we have that all answer streams for Q w.r.t. $\mathcal{I}$ are also answer streams for q w.r.t. $\mathcal{I}$.

The following proposition gives a characterisation of plans that use only stream producers.

Proposition 1 (Plans Using Producers). *Let $Q = \sigma_{C_1}(S_1) \uplus \ldots \uplus \sigma_{C_m}(S_m)$ be a local query where each S_i is described by a view $\sigma_{D_i}(r)$ and let $q = \sigma_C(r)$ be a global query. Then Q is a plan for q if and only if the following holds:*

1. *for each $i \in 1..m$ we have that*

$$C_i \wedge D_i \models C \qquad \text{and} \qquad C \wedge D_i \models C_i; \tag{2}$$

2. *every stream producer S with a descriptive view $\sigma_D(r)$ such that $C \wedge D$ is satisfiable occurs as some S_i.*

Proof. The first condition ensures that any producer occurring in Q contributes only tuples satisfying the query and that it contributes all such tuples that it can possibly produce. The second condition ensures that any producer that can possibly contribute occurs in the plan.

Since by assumption all S_i are distinct and the streams of distinct producers are disjoint, all answers of Q are duplicate free. Also, by the definition of union of streams, all answers are weakly ordered. □

The proposition can be immediately translated into an algorithm to compute plans. For instance, in the scenario of Fig. 5, it would yield the plan $Q = \sigma_{true}(\mathsf{S1}) \uplus \sigma_{true}(\mathsf{S2})$. We discuss in Section 6 how to compute plans in the presence of republishers.

[1] It would be straightforward to generalise our work to unions and projections, although it would complicate the presentation.

5 Query Plans Using Republishers

We introduce republishers and generalise query plans accordingly. Then we develop characteristic criteria that allow one to check whether a local query over arbitrary publishers is a plan for a glolal query.

5.1 Republishers and Queries over Republishers

A *republisher* R is a component that is defined by a global query $q_R = \sigma_D(r)$. For a given instance $\mathcal{I}$ of a producer configuration the republisher outputs a stream that is an answer to q_R w.r.t. $\mathcal{I}$. The descriptive view $v(R)$ of a republisher is identical to the defining query q_R. A *republisher configuration* $\mathcal{R}$ is a set of republishers.

Publisher Configurations. Since both producers and republishers publish streams, we refer to them collectively as *publishers*. Ultimately, we want to answer global queries using arbitrary publishers.

We define a *publisher configuration* as a pair $\mathcal{P} = (\mathcal{S}, \mathcal{R})$ consisting of a producer and a republisher configuration. By abuse of notation, we shall identify $\mathcal{P}$ with the set $\mathcal{S} \cup \mathcal{R}$.

A stream assignment $\mathcal{J}$ for publishers in $\mathcal{P}$ is an *instance* of $\mathcal{P}$ if (i) the restriction $\mathcal{J}_{|\mathcal{S}}$ of $\mathcal{J}$ to $\mathcal{S}$ is an instance of $\mathcal{S}$ and if (ii) for every republisher R the stream $R^{\mathcal{J}}$ is an answer for the global query q_R w.r.t. $\mathcal{J}_{|\mathcal{S}}$. Thus, an instance $\mathcal{J}$ is "essentially" determined by $\mathcal{J}_{|\mathcal{S}}$. Note that $R^{\mathcal{J}}$ being an answer for a global query implies that $R^{\mathcal{J}}$ is duplicate free and weakly ordered.

Local Queries over Publishers. In the presence of republishers we generalise our local queries, which had the form (1), in such a way as to allow them to be posed over arbitrary publishers. Thus, general local queries have the form

$$Q = \sigma_{C_1}(P_1) \uplus \ldots \uplus \sigma_{C_m}(P_m), \tag{3}$$

where $P_1, \ldots, P_m$ are distinct publishers.

A stream s is an *answer* for Q w.r.t. $\mathcal{J}$ if s is a union of the selections $\sigma_{C_1}(P_1^{\mathcal{J}}), \ldots, \sigma_{C_m}(P_m^{\mathcal{J}})$.

Similarly as before, we say that a local query Q as in Equation (3) is a *plan* for a global query q if for all instances $\mathcal{J}$, every answer for Q is an answer for q.

We are interested in characterising when a local query over a publisher configuration is a plan for a global query. Republishers add to the difficulty of this task because they introduce redundancy. As a consequence, answers to such a query need not be duplicate free or weakly ordered.

5.2 Properties of Plans

We first identify the characteristic properties of plans. They are defined in terms of the properties of the answers to a query.

Consider a fixed publisher configuration $\mathcal{P}$ and let Q be a query over $\mathcal{P}$ as in Equation (3). We say that Q is *duplicate free* if for all instances $\mathcal{J}$ of $\mathcal{P}$ all answer streams for Q w.r.t. $\mathcal{J}$ are duplicate free. In a similar way, we define when Q is *weakly ordered.* Let q be a global query. We say that Q is *sound* for q if for all instances $\mathcal{J}$ of $\mathcal{P}$ all answer streams for Q w.r.t. $\mathcal{J}$ are sound for q. A stream s is *complete* for q w.r.t. a producer instance $\mathcal{I}$ if every tuple in an answer stream for q w.r.t. $\mathcal{I}$ occurs also in s. We say that Q is *complete* for q if for all instances $\mathcal{J}$ all answer streams for Q w.r.t. $\mathcal{J}$ are complete for q w.r.t. $\mathcal{J}_{|\mathcal{S}}$.

Clearly Q is a plan for q if and only if Q is (i) sound for q, (ii) complete for q, (iii) duplicate free, and (iv) weakly ordered. For soundness and completeness one would expect characterisations similar to those in Propostion 1. However, with republishers there is the difficulty that the descriptive views do not accurately describe which data a republisher offers in a given configuration. For instance, a republisher may always publish the empty stream if the configuration does not contain any producers whose views are compatible with the republisher's query.

Given a publisher configuration $\mathcal{P}$, we derive for every republisher R, defined by the query $\sigma_D(r)$, a new condition D' as follows. Let $S_1, \ldots, S_n$ be all producers for r in $\mathcal{P}$, where $v(S_i) = \sigma_{E_i}(r)$. Then we define

$$D' = D \wedge (\bigvee_{i=1}^{n} E_i).$$

Intuitively, D' describes which of the tuples that can actually be produced in $\mathcal{P}$ will be republished by $\mathcal{R}$. We call $v'(R) := \sigma_{D'}(r)$ the *relativisation* of $v(R)$ w.r.t. $\mathcal{P}$. For a producer S we define the relativisation $v'(S)$ to be equal to $v(S)$. Note that an empty disjunction is equivalent to *false* and therefore the relativised condition for a republisher that does not have producers is *false*.

5.3 Soundness

First, we give a characterisation of soundness.

Theorem 1. *Let $\mathcal{P}$ be a publisher configuration, $q = \sigma_C(r)$ a global query, and*

$$Q = \sigma_{C_1}(P_1) \uplus \cdots \uplus \sigma_{C_m}(P_m) \tag{4}$$

be local query over $\mathcal{P}$. Suppose that the descriptive view of P_i is $v(P_i) = \sigma_{D_i}(r)$ and that the relativisation is $v'(P_i) = \sigma_{D'_i}(r)$. Then Q is sound *for q if and only if for each $i \in 1..m$ we have that*

$$C_i \wedge D'_i \models C. \tag{5}$$

Proof. Clearly, if Equation (5) holds, then every tuple in an answer to $\sigma_{C_i}(r)$ over $\mathcal{P}$ satisfies C, and so does every tuple in an answer to Q over $\mathcal{P}$.

Conversely, if Equation (5) does not hold, then there is a tuple t that satisfies some C_i and D'_i, but not C. Since the argument is simpler if P_i is a producer, we assume without loss of generality that P_i is a republisher.

Since t satisfies D'_i, there is a producer S with $v(S) = \sigma_E(r)$ such that t satisfies D_i and E. Let $\mathcal{J}$ be an instance where the stream $S^{\mathcal{J}}$ contains t. Then the stream $P_i^{\mathcal{J}}$ contains t as well, because $P_i^{\mathcal{J}}$ is an answer for $\sigma_{D_i}(r)$. Then t is in every answer stream for $\sigma_{C_i}(P_i)$ and therefore in every answer stream for Q w.r.t. $\mathcal{J}$. However, t does not occur in any answer stream for Q because t does not satisfy C. □

It is easy to see that the criterion of the theorem above can be weakened to a sufficient one if instead of Equation (5) we require that for each $i \in 1..m$ we have

$$C_i \wedge D_i \models C, \tag{6}$$

where D_i is the original condition in the descriptive view of P_i.

5.4 Completeness

To characterise completeness, we distinguish between the producers and the republishers in a local query. The reason is that the stream of a republisher is always complete for its descriptive view while this need not be the case for a producer.

Let Q be a query as in Equation (4) and suppose that $R_1, \ldots, R_k$ are the republishers and $S_1, \ldots, S_l$ the stream producers among $P_1, \ldots, P_m$. Then we can write the query as $Q = Q^{\mathbf{R}} \uplus Q^{\mathbf{S}}$ where

$$Q^{\mathbf{R}} = \sigma_{C_1}(R_1) \uplus \cdots \uplus \sigma_{C_k}(R_k) \tag{7}$$

$$Q^{\mathbf{S}} = \sigma_{C'_1}(S_1) \uplus \cdots \uplus \sigma_{C'_l}(S_l). \tag{8}$$

Suppose that the republishers have the descriptive views $v(R_i) = \sigma_{D_i}(r)$.

We define a condition $C_Q^{\mathbf{R}}$, which summarises the conditions in the selections of the republisher part $Q^{\mathbf{R}}$ of Q, as follows:

$$C_Q^{\mathbf{R}} = \bigvee_{j=1}^{k} (C_j \wedge D_j). \tag{9}$$

Theorem 2. *Let $\mathcal{P}$ be a publisher configuration, $q = \sigma_C(r)$ a global query, and $Q = Q^{\mathbf{R}} \uplus Q^{\mathbf{S}}$ a local query where $Q^{\mathbf{R}}$ and $Q^{\mathbf{S}}$ are as in Equations (7) and (8). Then Q is* complete *for q if and only if for every stream producer $S \in \mathcal{P}$, where S is described by the view $\sigma_E(r)$, one of the two following statements holds:*

1. *$S = S_i$ for some producer S_i in $Q^{\mathbf{S}}$ and*

$$C \wedge E \models C_Q^{\mathbf{R}} \vee C'_i; \tag{10}$$

2. S does not occur in $Q^{\mathbf{S}}$ and

$$C \wedge E \models C_Q^{\mathbf{R}}. \tag{11}$$

Proof. We only give a sketch. A full proof is not difficult but tedious.

To see that the criterion is sufficient note that any tuple in an answer for q must satisfy C and must originate from some producer for r with view condition E. Let S be such a producer. A tuple returned by Q can occur either as an element of an answer for $Q^{\mathbf{R}}$ or as an element of an answer for $Q^{\mathbf{S}}$. If S is present in Q, then Equation (10) guarantees that a tuple produced by S is either returned by $Q^{\mathbf{R}}$ or by $Q^{\mathbf{S}}$. If S is not present in Q, then Equation (11) guarantees that a tuple produced by S is returned by $Q^{\mathbf{R}}$.

To see that the criterion is necessary, assume that there is producer S for which none of the two statements holds. Suppose that S occurs in $Q^{\mathbf{S}}$. Then there is a tuple t such that t satisfies $C \wedge E$, but satisfies neither $C_Q^{\mathbf{R}}$ nor C_i'. There exists an instance $\mathcal{J}$ of $\mathcal{P}$ such that t occurs in the stream $S^{\mathcal{J}}$. Every answer for q w.r.t. $\mathcal{J}$ contains t. However, t does not occur in any answer for Q w.r.t. $\mathcal{J}$. With a similar argument one can show that t does not occur in any answer for Q if S does not occur in $Q^{\mathbf{S}}$. In summary, this proves that Q is not complete for q. □

5.5 Duplicate Freeness

Next, we give a characterisation of duplicate freeness.

Theorem 3. *Suppose $\mathcal{P}$ is a publisher configuration and Q a local union query over publishers $P_1, \ldots, P_m$ as in Equation* (3)*. Suppose that the relativised descriptive view of each P_i is $v'(P_i) = \sigma_{D_i'}(r)$. Then Q is* duplicate free *if and only if the condition*

$$(C_i \wedge D_i') \wedge (C_j \wedge D_j') \tag{12}$$

is unsatisfiable for each republisher P_i and publisher P_j where $i \neq j$.

Proof. If the statement is true, then for any instance $\mathcal{J}$, the streams $\sigma_{C_i}(P_i^{\mathcal{J}})$ are mutually disjoint and every answer of Q is duplicate free because the streams $\sigma_{C_i}(P_i^{\mathcal{J}})$ are duplicate free.

If the statement is not true, then there are i and j with $i \neq j$ and a tuple t such that t satisfies both $C_i \wedge D_i'$ and $C_j \wedge D_j'$. Suppose that P_i is a republisher and P_j is a producer. Consider an instance $\mathcal{J}$ where t occurs in the stream $P_j^{\mathcal{J}}$ of the producer P_j. Since P_i is a republisher, t occurs also in the stream $P_j^{\mathcal{J}}$. Finally, since t satisfies both C_i and C_j, the tuple occurs in both streams, $\sigma_{C_i}(P_i^{\mathcal{J}})$ and $\sigma_{C_j}(P_j^{\mathcal{J}})$. Hence, there is an answer to Q where the tuple t occurs twice.

If both P_i and P_j are republishers, one can show that there is a producer S with view $\sigma_E(r)$ such that $D_i \wedge D_j \wedge E$ is satisfiable. Then one chooses a satisfying tuple t and considers an instance $\mathcal{J}$ where $S^{\mathcal{J}}$ contains t. The rest of the argument is analogous to the first case. □

Similar to Theorem 1, we can turn the criterion of the above theorem into a sufficient one if we replace in Equation (12) the relativised conditions D_i' by the view conditions D_i, that is, if we require that

$$(C_i \wedge D_i) \wedge (C_j \wedge D_j) \tag{13}$$

is unsatisfiable for each republisher P_i and publisher P_j where $i \neq j$.

5.6 Weak Order

The following lemma gives a semantic characterisation of weakly ordered queries.

Lemma 1. *Let $\mathcal{P}$ be a publisher configuration and $Q = \sigma_{C_1}(P_1) \uplus \cdots \uplus \sigma_{C_m}(P_m)$ be a local query. Then Q is weakly ordered if and only if for all publishers P_i, P_j with $i \neq j$ occurring in Q and for every instance $\mathcal{J}$ of $\mathcal{P}$ the following holds:*

If t and t' are tuples occurring in the two streams $\sigma_{C_i}(P_i^{\mathcal{J}})$ and $\sigma_{C_j}(P_j^{\mathcal{J}})$, respectively, then t and t' disagree on their key attributes.

The lemma holds because otherwise the two streams in question could be merged in such a way that t and t' occur in an order that disagrees with their timestamps. The lemma excludes, for instance, the possibility to use two republishers $R_{>10}$ and $R_{\leq 10}$ with views $\sigma_{\mathsf{latency}>10}(\mathsf{tp})$ and $\sigma_{\mathsf{latency}\leq 10}(\mathsf{tp})$, respectively, for answering the query $\sigma_{true}(\mathsf{tp})$. The reason is that, $\mathsf{latency}$ being a measurement attribute, some tuples of a given channel could end up being republished by $R_{>10}$ and others by $R_{\leq 10}$.

Since in the end, we are interested in characterising plans for global queries, we ask next when a local query is weakly ordered *and* complete for some global query q. Considering these two properties together has the advantage that it leads to a characterisation in terms of the individual disjuncts that make up a union query.

Lemma 2. *Let $\mathcal{P}$ be a publisher configuration and $Q = \sigma_{C_1}(P_1) \uplus \cdots \uplus \sigma_{C_m}(P_m)$ be a local query. Suppose that Q is complete for the global query $\sigma_C(r)$. Then Q is weakly ordered if and only if for every publisher P_i occurring in Q and every instance $\mathcal{J}$ of $\mathcal{P}$ the following holds:*

If the stream $\sigma_{C_i}(P_i^{\mathcal{J}})$ contains some *tuple t that satisfies C, then this stream contains* every *tuple t' that is generated by a producer for r such that t' satisfies C and t' agrees with t on the key attributes.*

This lemma follows immediately from the preceding one: if it is impossible for two publishers to publish tuples from the same channel, then all tuples of one channel must come from the same publisher.

Lemma 2 can be formalised in logic. We write the condition C of query q as $C(x, y)$, where x stands for the vector of key attributes of r, which identifies a channel, and y for the non-key attributes. Similarly, we write the conditions C_i in query Q and D_i' in the relativised descriptive views as $C_i(x, y)$ and $D_i'(x, y)$ and we abbreviate the conjunction $C_i(x, y) \wedge D_i'(x, y)$ as $F_i(x, y)$.

Theorem 4. *Let $\mathcal{P}$ be a publisher configuration, Q a local query over $\mathcal{P}$, where $Q^{\mathbf{R}} = \sigma_{C_1}(R_1) \uplus \cdots \uplus \sigma_{C_k}(R_k)$, and $q = \sigma_C(r)$ a global query. Suppose that Q is complete for q w.r.t. $\mathcal{P}$. Then Q is* weakly ordered *if and only if for all $i \in 1..k$ we have*

$$\exists y.\,(C(x,y) \wedge F_i(x,y)) \models \forall y.\,(C(x,y) \rightarrow F_i(x,y)). \tag{14}$$

Proof. Suppose that Equation (13) holds for all $i \in 1..k$. Consider an instance $\mathcal{J}$ of $\mathcal{P}$. We want to show the claim using Lemma 2.

Suppose that $t = (t_x, t_y)$ is a tuple in the stream $\sigma_{C_i}(R_i^{\mathcal{J}})$ obtained from a republisher R_i. Then t_x satisfies $\exists y.\,(C(x,y) \wedge F_i(x,y))$. By Equation (13), it follows that t_x also satisfies $\forall y.\,(C(x,y) \rightarrow F_i(x,y))$. Let $t' = (t_x, t'_y)$ be a tuple that is generated by a producer for r and agrees with t on the key attributes. Suppose that t' satisfies C. Then, since t_x satisfies $\forall y.\,(C(x,y) \rightarrow F_i(x,y))$, it follows that t' also satisfies F_i. Hence, t' occurs also in the stream $\sigma_{C_i}(R_i^{\mathcal{J}})$.

Since producer streams do not share channels, Lemma 2 yields the sufficiency of the criterion.

We now show the necessity. Suppose that Equation (13) does not hold for some $i \in 1..k$. Then there is a tuple $t = (t_x, t_y)$ that satisfies $C \wedge F_i$ and a tuple $t' = (t_x, t'_y)$ such that t' satisfies C, but not F_i. By definition of F_i, the tuple t satisfies C_i, D_i, and some condition E for a stream producer S with descriptive view $\sigma_E(r)$. We construct an instance $\mathcal{J}$ where both t and t' occur in the stream of S. Then t occurs in every answer to $\sigma_{D_i}(r)$, the defining query of R_i, and thus in $R_i^{\mathcal{J}}$. Moreover, t occurs in the stream $\sigma_{C_i}(R_i^{\mathcal{J}})$. However, since t' does not satisfy F_i, it does not occur in that stream. Hence, by Lemma 2 it follows that Q is not weakly ordered. □

We note that the proof above would go through as well if we changed Equation (13) into

$$\exists y.\,(C(x,y) \wedge C_i(x,y) \wedge D'_i(x,y)) \models \forall y.\,(C(x,y) \rightarrow C_i(x,y) \wedge D_i(x,y)),$$

that is, if we replace D'_i by D_i on the right hand side of the entailment. This formulation, however, is less concise.

Let us review that part of the proof above that shows the sufficiency of the fact that Equation (13) holds for all $i \in 1..k$ for the claim of Theorem 4. It turns out that it goes through as well if we define

$$F_i(x,y) = C_i(x,y) \wedge D_i(x,y), \tag{15}$$

that is, if we replace relativised by original view conditions. Thus, Equation (**??**) leads to a simpler albeit sufficient criterion for weak order.

The entailment in Equation (13) of Theorem 4 is in general difficult to check because of the universal quantifier. However, it can be simplified if in queries and in descriptive views the conditions on key and on non-key attributes are decoupled, that is, if every condition $C(x,y)$ can be written equivalently as $C^{\kappa}(x) \wedge C^{\mu}(y)$ (and analogously C_i and D_i, and therefore also F_i). This restriction is likely not to cause difficulties in practice.

Theorem 5. *Suppose* $C(x,y) \equiv C^\kappa(x) \wedge C^\mu(y)$ *and* $F_i(x,y) \equiv F_i^\kappa(x) \wedge F_i^\mu(y)$. *Then*

$$\exists y.\,(C(x,y) \wedge F_i(x,y)) \models \forall y.\,(C(x,y) \rightarrow F_i(x,y))$$

holds if and only if one of the following holds:

1. $C^\kappa(x) \wedge F_i^\kappa(x)$ *is unsatisfiable;*
2. $C^\mu(y) \wedge F_i^\mu(y)$ *is unsatisfiable;*
3. $C^\mu(y) \models F_i^\mu(y)$.

We omit the proof of the theorem, since it is elementary, but tedious. Again, we obtain a sufficient criterion if in the definition of the F_i we replace the relativised view conditions by the original ones.

The theorems in this subsection contain characterisations that allow us to verify whether a local query is a plan for a global query. As we have seen, the characterisations can be simplified to yield sufficient criteria for soundness, duplicate freeness and weak order.

In the next section we discuss how the characterisations can be used to compute query plans over a publisher configuration. Specifically, these techniques can be used to realise hierarchies of republishers where republishers consume from other republishers.

6 Computing Query Plans

Based on the characterisations in the previous section, there is a straightforward approach to constructing a plan Q for a global query $q = \sigma_C(r)$. If $S_1, \ldots, S_n$ is a sequence comprising all stream producers in a configuration $\mathcal{P}$ that publish for relation r, then by Proposition 1 the query

$$\sigma_C(S_1) \uplus \cdots \uplus \sigma_C(S_n) \tag{16}$$

is a plan for q. This plan, however, may access a higher number of publishers than necessary because it does not make use of republishers. The question arises when a publisher is potentially useful for a query.

General Assumption. *We assume from now on that in global queries and descriptive views the conditions on key and non-key attributes are decoupled, that is, every condition* C *can be equivalently rewritten as* $C^\kappa \wedge C^\mu$, *where* C^κ *involves only key attributes and* C^μ *involves only non-key attributes.*

6.1 Relevant Publishers

We want to find out which publishers can potentially contribute to a query plan.

We say that a publisher P is *strongly relevant* for a query q w.r.t. to a configuration $\mathcal{P}$ if there is a plan Q for q that contains a disjunct $\sigma_{C'}(P)$ such that for some instance $\mathcal{J}$ of $\mathcal{P}$ the stream $\sigma_{C'}(P^{\mathcal{J}})$ is non-empty.

Proposition 2 (Strong Relevance). *Let $\mathcal{P}$ be a publisher configuration and P a publisher with view $\sigma_D(r)$, where $D = D^\kappa \wedge D^\mu$, and where D' is the relativised view condition. Let $q = \sigma_C(r)$ be a global query where $C = C^\kappa \wedge C^\mu$. Then P is* strongly relevant *for q w.r.t. $\mathcal{P}$ if and only if*

1. *$C \wedge D'$ is satisfiable, and*
2. *$C^\mu \models D^\mu$.*

Proof. If P is strongly relevant, then Statement 1 holds because P contributes some tuple to q and Statement 2 holds by Theorem 4 because the plan containing P is complete and weakly ordered.

Conversely, suppose the two statements hold. If P is a producer we construct an instance where P produces a tuple satisfying C. Then P can be part of a plan as in Equation (**??**). Because of Statement 1 there is an instance where P contributes at least one tuple to the answer of the plan.

If P is a republisher, we consider the query $Q = \sigma_C(P) \uplus \sigma_{C'}(S_1) \uplus \cdots \uplus \sigma_{C'}(S_n)$, where $S_1, \ldots, S_n$ are all producers for r in $\mathcal{P}$ and $C' = C \wedge \neg D$. Then it is easy to check that Q is duplicate free and sound and complete for q. Moreover, because of Statement 2, Q is weakly ordered. Finally, Statement 1 allows us to construct an instance of $\mathcal{P}$ where P actually contributes to Q. □

Criterion 1 of Proposition 2 involves relativised views. In practice, this is hard to check because there may be a large number of producers in a configuration and producers may come and go. We therefore generalise the criterion in such a way that it depends solely on the publisher and the query. We say that a publisher P with view $\sigma_D(r)$, where $D = D^\kappa \wedge D^\mu$, is *relevant* for a query $\sigma_C(r)$ with $C = C^\kappa \wedge C^\mu$ if it has the following two properties:

1. $C \wedge D$ is satisfiable (Consistency);
2. $C^\mu \models D^\mu$ (Measurement Entailment).

Intuitively, the first property states that P can potentially contribute values for *some* channels requested by q, while the second states that for those channels *all* measurements requested by q are offered by P.

Clearly, strong relevance implies relevance. Also, a relevant republisher may become strongly relevant if the right producers are added to the current configuration.

Consider the scenario in Fig. 5. Let $q = \sigma_{\mathsf{from}=\text{'}\mathtt{hw}\text{'}}(\mathsf{tp})$ be the query of the consumer. Then S1, S2, R1, R2 and R4 are the relevant publishers for q. They are also strongly relevant.

6.2 Subsumption of Publishers

In principle, there is a wide range of possibilities to construct query plans in the presence of republishers. We want to give preference to republishers over producers, since one of the main reasons for setting up republishers is to support more efficient query answering. Among the republishers, we want to prefer those

that can contribute as many channels as possible to a query. In order to be able to rank publishers we introduce a subsumption relationship.

We say that a stream s_1 is *subsumed* by a stream s_2 if for every channel c_1 in s_1 there is a channel c_2 in s_2 such that c_2 is a substream of c_1. A publisher P is *subsumed* by a republisher R w.r.t. a configuration $\mathcal{P}$, if for every instance $\mathcal{J}$ of $\mathcal{P}$ the stream $P^{\mathcal{J}}$ is subsumed by $R^{\mathcal{J}}$. Since $\mathcal{P}$ is usually clear from the context, we denote this simply as $P \preceq R$. We say that P is *strictly subsumed* by R and write $P \prec R$ if P is subsumed by R but not vice versa.

The definition entails that if P has the view $\sigma_{D^\kappa \wedge D^\mu}(r)$ and R the view $\sigma_{E^\kappa \wedge E^\mu}(r)$, then P is subsumed by R if and only if

$$D^\kappa \models E^\kappa \qquad \text{and} \qquad E^\mu \models D^\mu. \tag{17}$$

Consider a query $q = \sigma_C(r)$, where $C = C^\kappa \wedge C^\mu$. We want to rank relevant publishers for q also according to the channels they can contribute to q. If P is a relevant publisher for q and R a relevant republisher, then we say that P is *subsumed by R w.r.t. q,* and write $P \preceq_q R$, if for every instance $\mathcal{J}$ of $\mathcal{P}$ the stream $\sigma_C(P^{\mathcal{J}})$ is subsumed by $\sigma_C(R^{\mathcal{J}})$. We write $P \prec_q R$ to express that P is strictly subsumed by R w.r.t. q.

If the descriptive view of P is $\sigma_{D^\kappa \wedge D^\mu}(r)$ and the one of R is $\sigma_{E^\kappa \wedge E^\mu}(r)$, then $P \preceq_q R$ if and only if

$$D^\kappa \wedge C^\kappa \models E^\kappa. \tag{18}$$

The property $C^\mu \wedge E^\mu \models C^\mu \wedge D^\mu$ is always satisfied, since the relevance of R and P implies that $C^\mu \models E^\mu$ and $C^\mu \models D^\mu$.

In the scenario of Fig. 5, among the relevant publishers for q we have the subsumption relationships S1 $\prec_q$ R1, S2 $\prec_q$ R1, R2 $\prec_q$ R1, R1 $\preceq_q$ R4, and R4 $\preceq_q$ R1.

6.3 Plans Using Maximal Relevant Republishers

We present a method for constructing query plans that consist of publishers that are maximal with regard to the subsumption relation "$\preceq_q$". We suppose that the publisher configuration and the query $q = \sigma_C(r)$ are fixed.

A relevant publisher is *maximal* if it is not strictly subsumed by another relevant publisher. Let M_q be the set of maximal relevant publishers for q. We partition M_q into the subsets $M_q^{\mathbf{S}}$ and $M_q^{\mathbf{R}}$, consisting of stream producers and republishers, respectively.

We write $P_1 \sim_q P_2$ if $P_1 \preceq_q P_2$ and $P_2 \preceq_q P_1$. Note that a producer is never equivalent to another publisher because it cannot subsume the other publisher. Thus, the relation "$\sim_q$" is an equivalence relation on the set of republishers $M_q^{\mathbf{R}}$ and we say that R_1 is *equivalent to R_2 w.r.t. q* if $R_1 \sim_q R_2$. We denote the equivalence class of a republisher R w.r.t. q as $[R]_q$. Clearly, if P_1 and P_2 are two distinct maximal relevant publishers, and $P_1 \preceq_q P_2$, then $P_1 \sim_q P_2$.

We call

$$\mathcal{M}_q^{\mathbf{R}} = \left\{ \; [R]_q \;\middle|\; R \in M_q^{\mathbf{R}} \; \right\}$$

the *meta query plan* for q. The set $\mathcal{M}_q^{\mathbf{R}}$ consists of equivalence classes of maximal relevant republishers. A sequence $\langle R_1, \ldots, R_k \rangle$ of republishers that is obtained by choosing one representative from each class of republishers in $\mathcal{M}_q^{\mathbf{R}}$ is called a *supplier sequence* for q.

Let $\langle R_1, \ldots, R_k \rangle$ be a supplier sequence for q and $S_1, \ldots, S_l$ be the stream producers in $\mathcal{M}_q^{\mathbf{S}}$. Suppose the descriptive views of the R_i have the conditions D_i. We define the *canonical republisher query* for the sequence as

$$Q^{\mathbf{R}} = \sigma_{C_1}(R_1) \uplus \cdots \uplus \sigma_{C_k}(R_k), \tag{19}$$

where $C_1 = C$ and $C_i = C \wedge \neg(D_1 \vee \cdots \vee D_{i-1})$ for $i \in 2..k$. Moreover, we define the *canonical stream producer query* as

$$Q^{\mathbf{S}} = \sigma_{C'_1}(S_1) \uplus \cdots \uplus \sigma_{C'_l}(S_l), \tag{20}$$

where $C'_j = C \wedge \neg(D_1 \vee \ldots \vee D_k)$ for $j \in 1..l$.

The selection conditions on the disjuncts in $Q^{\mathbf{R}}$ ensure that R_i only contributes channels that no $R_{i'}$ with $i' < i$ can deliver, and the ones in $Q^{\mathbf{S}}$ that producers only contribute channels that cannot be delivered by the republishers.

Note that the conditions C_i depend on the order of republishers in the sequence, but once the order is fixed, they do not depend on which republisher is chosen from an equivalence class. This is due to the fact that for relevant republishers R and R' with descriptive conditions D and D', respectively, we have that $R \sim_q R'$ if and only if $C \wedge D$ is equivalent to $C \wedge D'$, that is, if and only if $C \wedge \neg D$ is equivalent to $C \wedge \neg D'$. Similarly, for all supplier sequences the conditions C'_j are the same up to equivalence.

Theorem 6. *Let q be a global query a $Q^{\mathbf{R}}$ and $Q^{\mathbf{S}}$ be the canonical republisher query and stream producer query for some supplier sequence for q. Then*

$$Q = Q^{\mathbf{R}} \uplus Q^{\mathbf{S}} \tag{21}$$

is a plan for q.

Proof. We only sketch the proof. To prove that Q is a plan, we have to show that Q is sound and complete for q, duplicate free and weakly ordered.

The conditions in the selections of Q satisfy Equation (6) and thus ensure soundness. They also satisfy Equation (**??**) and thus ensure duplicate freeness. Completeness is guaranteed because Q satisfies the properties stated in Theorem 2 because maximal republishers are chosen for Q, together with producers that are not subsumed by a republisher. Finally, Q is weakly ordered because the republishers used in Q are relevant and thus satisfy the Measurement Entailment Property. □

Continuing the discussion of the example in Fig. 5, we see that R1 and R4 are the maximal relevant publishers for the consumer query. Since both republishers are equivalent w.r.t. q, the local queries $\sigma_{\mathsf{from}=\texttt{'hw'}}(\mathsf{R1})$ and $\sigma_{\mathsf{from}=\texttt{'hw'}}(\mathsf{R4})$ are both plans for q.

As another example, let us consider the query $q' = \sigma_{true}(\mathsf{tp})$, by which the top level republisher is defined. For this query, all publishers are relevant (except R4, for which we do the planning). Moreover, subsumption and subsumption w.r.t. q' coincide. There are three maximal publishers, R1, R2, R3, none of which is equivalent to another publisher. Thus $\langle \mathsf{R1}, \mathsf{R2}, \mathsf{R3} \rangle$ is a supplier sequence, and the corresponding plan is

$$\sigma_{true}(\mathsf{R1}) \uplus \sigma_{\mathsf{from} \neq \texttt{'hw'}}(\mathsf{R2}) \uplus \sigma_{\mathsf{from} \neq \texttt{'hw'} \wedge (\mathsf{tool} \neq \texttt{'ping'} \vee \mathsf{psize} < 128)}(\mathsf{R3}).$$

Computing plans that use maximal republishers involves satisfiability and entailment checks. Clearly, this makes the task intractable in the worst case if we admit arbitrary conditions. However, if conditions in queries and views are of the restricted form that R-GMA supports currently, namely conjunctions

$$attr_1\ op_1\ val_1 \wedge \ldots \wedge attr_n\ op_n\ val_n,$$

where $op_i \in \{ <, \leq, =, \geq, > \}$, then both satisfiability and entailment checks are polynomial. They remain polynomial if one allows for slightly more general conditions by admitting also comparisons between attributes of the form "$attr_1\ op\ attr_2$" or limited disjunctions of the form "$attr$ in $\{ val_1, \ldots, val_n \}$".

We foresee that the technique presented above will be the basis for planning consumer queries in R-GMA. For planning republisher queries, however, some modifications are needed to ensure that plans do not introduce cyclic dependencies between republishers. To avoid checking for possible cycles whenever a new republisher query is planned, it seems more feasible not to consider all republishers relevant to the query, but only those that are strictly subsumed with respect to the general subsumption relationship.

In an implementation, the responsibility for planning can be divided between registry and consumer agents. A possible way to do this is for the registry to hand the list of maximal publishers over to the consumer agent, which then decides which republisher in each class to contact. Such an approach could also reduce the communication between registry and consumer agent because an agent would only need to be notified of new producers that are not subsumed by the republishers in the meta query plan.

6.4 Irredundant Plans

A natural question to ask about the planning technique presented in the previous section is how good the plans it produces are. A way to approach this question is to ask whether or not the plans contain redundancies. To simplify the discussion, we focus only on redundancies among republishers.

We say that a local query Q *covers* a global query $q = \sigma_C(r)$ w.r.t. a publisher configuration if Q is complete for q and for every relevant republisher P for q we have

$$C \wedge D \models C_Q^{\mathbf{R}}, \tag{22}$$

where D is the condition in the descriptive view of R and $C_Q^{\mathbf{R}}$ is defined as in Equation (9). While completeness requires a query to "cover" all relevant

producers, a covering query has to "cover" also all relevant republishers. Plans based on maximal republishers as introduced in the preceding subsection are always covering by construction.

Given q, a covering query Q using republishers $R_1, \ldots, R_k$ is *irredundant* if there is no covering query Q' that uses only a strict subset of $R_1, \ldots, R_k$.

We say that a condition is *simple* if it is a conjunction of comparisons of the form "*attr op val*", where *op* is one of $\leq$, $<$, $=$, $>$, or $\geq$. A global query is simple if its condition is simple. A configuration is *simple* if all its descriptive views are simple. A local query is *simple* if all its disjuncts have simple conditions.

Proposition 3 (Irredundancy is NP-hard). *Checking whether a covering query is irredundant is* NP*-hard. This is still the case if we consider only local queries, global queries, and configurations that are simple.*

Proof. We prove the claim by a reduction of the irredundancy problem for propositional clauses. A set of clauses Γ is said to be irredundant if there is no clause $\gamma \in \Gamma$ such that $\Gamma \models \gamma$. Irredundancy of clause sets is known to be NP-complete [14].

Suppose Γ is a clause set and $p_1, \ldots, p_n$ are the propositional atoms occurring in Γ. Let r be a relation with key attributes $a_1, \ldots, a_n$, ranging over the rational numbers. We define conditions G_i as $a_i > 0$ and $\bar{G}_i$ as $a_i \leq 0$. For every clause $\gamma \in \Gamma$ we define D_γ as the conjunction of all G_i such that $\neg p_i \in \gamma$ and all $\bar{G}_i$ such that $p_i \in \gamma$.

For each γ, let R_γ be a republisher defined by the query $\sigma_{D_\gamma}(r)$ and let $\mathcal{P}$ be the configuration consisting of all republishers R_γ. Let $q = \sigma_{true}(r)$ and let Q be the union of all $\sigma_{true}(R_\gamma)$ where $\gamma \in \Gamma$. Then Q is clearly covering q w.r.t. $\mathcal{P}$. Moreover Q is irredundant if and only if for all $\gamma \in \Gamma$ we have that

$$D_\gamma \not\models \bigvee_{\gamma' \in \Gamma \setminus \{\gamma\}} D_{\gamma'},$$

which holds if and only if $\bigwedge_{\gamma' \in \Gamma \setminus \{\gamma\}} \neg D_{\gamma'} \not\models \neg D_\gamma$. Now, it is easy see that this is the case if and only if $\bigwedge_{\gamma' \in \Gamma \setminus \{\gamma\}} \gamma' \not\models \gamma$ for all $\gamma \in \Gamma$, that is, if Γ is irredundant. □

We note without proof that there are situations where all plans based on maximal relevant republishers are irreducible. Let us a call a set of operators *one-sided* if it is a two-element subset of $\{<, \leq, =\}$ or $\{>, \geq, =\}$. If all conditions in queries and views are simple and built up from a one-sided set of operators ranging over the rational numbers, then plans of the form (18) are irreducible. The reason is that if D, E, and E' are such conditions, then we have that $D \models E \vee E'$ holds if and only if $D \models E$ or $D \models E'$ holds.

6.5 Open Problems

We have set up a framework to model distributed streams that are queried via a global schema. A feature of the approach are so-called stream republishers,

a view-like mechanism that allows one to create new streams that are defined by a continuous query. We have restricted the form of queries to enable us to concentrate on semantic issues, such as the meaning of a query or characteristic properties of query plans. The restrictions are reflecting the current requirements of the R-GMA Grid monitoring system, to the development of which we are contributing.

Even in this restricted framework there are a number of open issues that we have not addressed in the current paper. The most important one is how to switch from one plan to another. In its simplest form it involves replacing a republisher by an equivalent one, for instance, when a republisher experiences a failure. A slightly more difficult task is to replace a republisher by the producers that feed into it or vice versa. In doing so it is crucial that no tuples are lost and no duplicates are introduced into the answer stream.

To achieve this a consumer agent has to know where to stop consuming from old input streams and where to start consuming from new ones. We envisage protocols that make the switch on a channel by channel basis and exploit the fact that channels are weakly ordered: For each publisher in a plan, the consumer agent remembers the timestamp of the last tuple from each channel that it has received. Then it can switch that channel to a new supplier as soon as it receives from it a tuple that the old one has already delivered. Tricky situations may arise if a component fails during such a transition period.

In our discussion we have assumed that the publisher configuration is fixed. In reality, however, it changes continuously, as publishers come and go. This requires plan modification techniques that subject existing plans to changes that are as small as possible.

7 R-GMA Implementation

We have been working together with Work Package 3 of the DataGrid project to implement the architecture and techniques presented in this paper. By the end of the DataGrid project, a working R-GMA system had been deployed on a testbed containing 25 sites, and was attracting a growing user base. In this section, we describe this implementation, and indicate some of the ways the system is being used.

The system continues to be developed in a follow-on project called EGEE (= "Enabling Grids for E-Science in Europe") [6]. One of the aims of EGEE is to prepare Grid middleware components developed in DataGrid, including R-GMA, for use in a much larger Grid being set up to support physicists working with the Large Hadron Collider (LHC) at CERN.

7.1 Overall Approach

The Grid components that play the role of producer or consumer in R-GMA are applications that are coded in several programming languages. To support these, R-GMA offers APIs in C, C++, Java, Perl and Python. As it would not

make sense to duplicate all of the core functionality of R-GMA in each of these languages, these APIs communicate with agents. The agents are realised using servlet technology, and are hosted by web servers.

We have seen in Section 2 that the system should be resilient to failures in the network. R-GMA achieves this using a simple heartbeat registration protocol. Heartbeats are sent from API to agent, and from agent to registry. If a problem occurs, and heartbeats fail to arrive within some time interval, the receiver can assume that the sender is no longer around.

Another requirement, that the system should not have a single point of failure, was not met by the end of DataGrid. Work was started on protocols for replicating the registry and schema; however, this had not been finalised by the end of the DataGrid project and is continuing in the EGEE project.

7.2 Use of R-GMA Components in DataGrid

In DataGrid, R-GMA was mainly used for publishing network monitoring data and for providing information on resources for resource brokers. In addition, it was tested for monitoring batch jobs.

Schema. R-GMA ships with a set of core relations that describe the components of a Grid and how they are related to each other. These relations are derived from a conceptual model called GLUE [10], which was defined by a number of Grid projects, including DataGrid. Users may also introduce new relations into the global schema.

To keep the system simple, R-GMA currently only supports publication of stream relations. Nevertheless, care was taken when designing the schema to separate data that changes rapidly (such as the number of free CPUs of a computing element) from more static data (e.g. lists specifying access rights of users).

Producers. As the schema only supports stream relations, R-GMA currently offers just a stream producer API for publishing data. All stream producers can answer continuous queries. In addition, the user can configure the agent to maintain the history or the latest state of the stream in a database.

In DataGrid, one use of stream producers was to periodically publish information about Grid resources for resource brokers. Dynamically changing relations were published every 30 seconds, whereas static relations were resent every hour. Fortunately, the volume of static information was moderate, as the testbed was small, and so this approach worked. However, as EGEE's testbed grows in size, mechanisms may be needed that forward only changes to "static" data sets.

Consumers. The consumer API allows users to pose continuous, latest-state and history queries in SQL. In principle, arbitrary one-time queries can be posed. However, these can only be answered if a republisher for all the tables in the

query can be located. Select-project queries, on the other hand, can always be answered by contacting all of the relevant producers.

Resource brokers have complex latest-state queries. In DataGrid's testbed, each resource broker was provided with a dedicated republisher. A consumer agent acting on behalf of a resource broker then has a choice. The closest republisher is always chosen for query answering, but if that cannot be contacted for some reason, then the agent automatically switches to using another republisher.

Republishers. Stream republishers have proved useful in DataGrid, for both query answering and for archiving. However, the system does not currently support the republisher hierarchies discussed earlier. There has not been a need for these so far, as DataGrid's testbed was small.

A use case for republisher hierarchies is emerging in the EGEE project. A job submissions accounting service is needed for the LHC Grid that logs all of the jobs that run on the Grid. Data about jobs that run at individual sites is to be collected into site databases, whereas a global history is needed in a central database. Such a system could be easily configured if R-GMA supported republisher hierarchies. Also, if global republishers stream from site republishers, then it becomes easier to automatically recommence a stream that is interrupted by network failure, as the site republisher maintains a history in a database.

Registry. Currently, publishers can only register views over single relations with conditions that are conjunctions of equalities of the form "$attr = val$". In this simple case, views can be stored in a structured form in a relational database and the satisfiability and entailment tests for finding relevant publishers can be expressed as SQL queries.

For large Grids, support for views involving aggregation might prove useful. Republishers could then republish summaries of streams rather than whole streams, and so the volume of information flowing through a hierarchy of republishers would be reduced.

8 Conclusions

We have introduced the problem of monitoring a computational Grid and discussed the requirements that arise from it. Grid monitoring involves publishing data from independent sources and querying it in a transparent manner. These are characteristics of a typical information integration setting. A number of challenges, though, that complicate other information integration problems do not exist here.

Since Grids are dynamic environments where the data of interest is relatively short lived, it is not necessary to deal with legacy data. Moreover, components of a Grid obey certain naming conventions to be interoperable. This vastly simplifies the task of matching of entities in different sources when joining information and eliminates the need for sophisticated matching algorithms. Thus, in a nutshell, the case of Grid monitoring is close to the abstract settings that have been

considered in theoretical work on information integration, which makes it an ideal candidate for applying these techniques [13].

Together with WP3 of DataGrid, we have developed and implemented the R-GMA architecture, which creates an environment where components pose queries in terms of a global schema and publish their data as views on that schema. Although the current implementation of R-GMA makes only limited use of advanced data integration techniques, such as query rewriting using views, there will be a need for them as the Grids on which R-GMA is being used grow.

This is only now becoming realistic. Query rewriting using views translates a global query into a query over a distributed set of sources, which has to be executed by a distributed query processor. Such facilities are now being made available in the public domain by the OGSA-DAI project although the code has not yet reached production quality [15].

The R-GMA approach also raises new research questions. Most notably, since monitoring data often comes in streams, formal models for integrating data streams is needed. In the present paper we have defined a framework for approaching this task and applied it to a simple type of continuous queries. The generalisation to more expressive queries will be a topic of future work.

Acknowledgement

The work reported in this paper was supported by the British EPSRC under the Grant GR/R74932/01 (Distributed Information Services for the Grid). The R-GMA architecture was developed jointly with the members of WP3 of DataGrid.

References

1. F. Berman. From TeraGrid to Knowledge Grid. *Communications of the ACM*, 44(11):27–28, 2001.
2. The CrossGrid Project. http://www.crossgrid.org, April 2004.
3. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proc. 10th IEEE International Symposium on High Performance Distributed Computing*, 2001.
4. The DataGrid Project. http://www.eu-datagrid.org, April 2004.
5. DataGrid WP3 Information and Monitoring Services. http://hepunx.rl.ac.uk/edg/wp3/, April 2004.
6. Enabling Grids for E-science in Europe. http://public.eu-egee.org/, April 2004.
7. I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2: Computational Grids, pages 15–51. Morgan Kaufmann, 1999.
8. Global Grid Forum. http://www.ggf.org, April 2004.
9. Globus Toolkit. http://www.globus.org, April 2004.
10. High Energy Nuclear Physics InterGrid Collaboration Board. http://www.hicb.org/glue/glue.htm, April 2004.
11. Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

12. W.E. Johnston, D. Gannon, and B. Nitzberg. Grids as production computing environments: The engineering aspects of NASA's Information Power Grid. In *Proc. 8th IEEE International Symposium on High Performance Distributed Computing*, pages 197–204. IEEE, 1999.
13. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. 21st Symposium on Principles of Database Systems*, pages 233–246. ACM, June 2002.
14. P. Liberatore. The complexity of checking redundancy of CNF propositional formulae. In *Proc. 15th European Conference on Artificial Intelligence*, pages 262–266. IEEE, IOS Press, July 2002.
15. Open Grid Services Architecture–Data Access and Integration (OGSA-DAI). http://www.ogsadai.org.uk, April 2004.
16. R.L. Ribler, J.S. Vetter, H. Simitci, and D.A. Reed. Autopilot: adaptive control of distributed applications. In *Proc. Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 172–179, 1998.
17. W. Smith. A system for monitoring and management of computational Grids. In *ICPP-31*, pages 55–, 2002.
18. B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid monitoring architecture. Global Grid Forum Performance Working Group, March 2000. Revised January 2002.
19. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.