RC 21942 (98646) 23 January 2001 **Computer Sciences**

IBM Research Report

Challenges in Flexible Aggregation of Pervasive Data

Norman H. Cohen, Apratim Purakayastha,
John Turek, Luke Wong, Danny Yeh
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Challenges in Flexible Aggregation of Pervasive Data

**Norman H. Cohen, Apratim Purakayastha,
John Turek, Luke Wong, and Danny Yeh**

IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{ncohen,apu,moose,lukew,dlyeh}@us.ibm.com

Abstract. The vision of billions of users connected to millions of services using trillions of devices is fast becoming a reality. The result will be a vast network of mobile communication devices and data sources, including sensors, newsfeeds, web services, and databases. Potential uses of this data span a wide range of application domains, including medical monitoring, traffic routing, proximity detection, electricity management, and service-fleet dispatching. Applications require flexible mechanisms for constructing condensed and refined views of the raw data, possibly in ways unanticipated by the data providers. *Aggregation* comprises collection of high volumes of raw data from data sources, composition of the raw data into less voluminous refined data, and timely delivery of the refined data to applications. There are difficult challenges inherent in creating an *aggregation system* that is sufficiently flexible, scalable, and reliable to address the needs of applications.

1 The aggregation problem

The explosive growth of networked data sources, including sensors, cellular telephones, online services, databases, and data feeds, presents novel opportunities for timely use of the data. The data is heterogeneous and possibly voluminous. Applications will combine raw data from multiple data sources in diverse ways, in a flexible, scalable, and reliable manner. *Aggregation* comprises the collection of raw data from pervasive data sources, the flexible, programmable composition of the raw data into less voluminous refined data, and the timely delivery of the refined data to data consumers. We propose the construction of an *aggregation system* to manage shared resources and facilitate the writing of applications.

Aggregation of data from many known sources in a predetermined manner, although difficult, is not the foremost challenge we face. The need for *flexibility* makes the problem especially challenging. Flexibility has three key aspects:

- Pervasive data sources cannot always anticipate possible uses of the data, and applications cannot always predetermine the sources of data that can be used for a computation.
- Composition of data must be flexible to accommodate diverse characteristics of data sources, mobility of data sources and consumers, and the dynamic nature of network resources.
- Scalability and reliability are intertwined with flexibility: An aggregation system must flexibly choose data sources and manners of composition in order to be both scalable and reliable.

As an example, consider an application that tells drivers the best routes from their current positions to their destinations. The computation is based on the driver's current location and

nearby traffic densities. The driver's location might be obtained from a wireless-enabled GPS receiver in the car or by querying the home location register for the cellular phone in the car. Traffic densities might be inferred from roadway traffic sensors, automated toll collections, cellular-phone data, or traffic newsfeeds. Thus the application may use data sources originally intended for other purposes, and it may use different data sources under different conditions.

Consider the scalability and reliability requirements associated with computing traffic density from sensor data. To improve scalability, the computation of the density metric is best done at the edge of the network, close to the data, and only the density metric should be propagated to the rest of the network. As users of this application move, however, traffic density metrics computed at different edge nodes become relevant to the computation, which the aggregation system must flexibly accommodate. Sensors may fail, or be unavailable in regions. An aggregation system must be flexible enough to select alternative sources of data (such as traffic feeds generated using visual input from traffic helicopters) in the event of sensor failure.

Flexible aggregation of data from pervasive devices can be instrumental in enabling a wide variety of other applications, including:

- **Medical alerts:** Worn or surgically implanted medical instrumentation collects raw data from patients and transmits it to an automated monitoring service. When the service recognizes a worrisome pattern specified earlier by a specialist (e.g., two episodes of tachycardia in an hour) the specialist is alerted. The specialist may remotely alter the patterns that the service tries to detect for a given patient, or alert the patient to seek immediate medical attention.
- **Proximity monitoring:** Based on location information from a variety of sources, a subscriber is alerted when a particular person, or a person satisfying certain attributes (e.g. "plays bridge") comes within a certain distance; or when the subscriber's automobile, laptop computer, or child has wandered beyond a certain distance.
- **Central control of electricity usage:** Based on real-time information about the amount of electric power available on the grid, temperature and humidity on customer premises, and the amount of money each customer is currently willing to pay to achieve various temperature-humidity-index values, an electric utility adjusts its prices, and the settings of individual customer air conditioners.
- **Service-fleet dispatching:** An enterprise continually monitors the status of equipment installed throughout a region, as well as traffic conditions and the position of each vehicle in the service fleet. When a situation requiring service is detected, the vehicle that can most economically handle the call is automatically dispatched.

There is considerable research underway into the design and deployment of low-power wireless sensors that configure themselves into a network to perform a specific task [Def99, Pot00, Sen00, Est99], typically in a battlefield or emergency-response setting. In contrast, our concern is flexible aggregation of potentially massive amounts of data both from sensors accessible at the edge of the network and from other networked data sources. We presuppose the existence of the network infrastructure and focus on the challenges needed in order to provide a general-purpose data aggregation platform upon which it is easy to build business and consumer applications.

Our model of an aggregation system includes *data sources* such as sensors, network services, and files. It includes *data consumers* such as applications. Finally, it includes *data*

composers, which act as both data consumers and data sources, combining input data to produce output data. Under different conditions, applications may use different data sources for the same computation, so data sources are best specified not by the name of a particular source, but by an *abstract data specification* listing attributes of the data to be retrieved. A process called *resolution* determines a data source from which data described by a given data specification can be obtained. The data may then be *composed* with other data. A data-aggregation application is programmed by specifying combining rules in terms of abstract data specifications.

Section 2 discusses resolution and Section 3 discusses composition. Section 4 addresses data aggregation privacy issues. Section 5 explores programming models for flexible data aggregation. Section 6 surveys related work. Section 7 offers our conclusions and describes our plans for developing an aggregation system called iQueue.

2 Data resolution

An application must, first of all, identify the data to be gathered on its behalf. A global assignment of names to data sources would be at too low a level of abstraction for applications. In the proximity-monitoring application, for example, a person's location may be inferred from different data sources at different times; it is the person, rather than a sensor, that the application should name in a request to the system. Furthermore, a global assignment of names must be relatively static, but the appropriate choice of data source may be highly dynamic. For example, a particular sensor may fail, and another sensor may fill in. Sometimes the entity of interest to an application is not a physical entity, but the *role* being played by the entity. We may wish to receive data from the external heart monitor worn by John Doe, but different physical boxes may play this role at different times.

Thus, sources of input data should be identified abstractly, by notations that describe the properties of the data or the role of the data provider, rather than by specifying a particular physical input source. Estrin *et al*. [Est99] foresee that sensor networks will be *data-centric*, in the sense that applications will request data matching certain properties, rather than data from particular sensors. In the same spirit, the Intentional Naming System [Adj99] is a resource-discovery and service-location system that identifies services according to a description of the required service (i.e., according to the requester's intent), rather than the name or location of a service provider.

Data resolution is a form of service discovery, in which the service to be discovered is a source of data satisfying specified criteria. It is more difficult than traditional forms of service discovery because the search space is so vast and because the correspondence between an abstract data specification and a given data source is potentially so volatile. Given an abstract data specification, an aggregation system must determine a corresponding set of candidate data sources, as discussed in Section 2.1. Then one of these candidates must be selected, as described in Section 2.2. The two-stage process is analogous to a Domain Name System (DNS) server returning two or more resource records, and a DNS resolver selecting one of them, but we use the word *resolution* to apply to the two stages collectively.

2.1 Discovery of data sources

Discovery is the determination of a set of data sources consistent with a given abstract data specification. A data specification may resolve to zero or more data sources, some of which may be external sources of raw data and some of which may be data composers. There are two

principal challenges in the discovery of data sources. The first is the vastness of the search space of potential data sources. The second is that the compatibility of a given data source with a given abstract data specification may vary dynamically.

2.1.1 The vast search space

The search space for data sources includes both raw data sources data composers. The search should encompass not only sources providing exactly the required data in the required format, but also approximate matches, providing data that a dynamically created data composer could easily transform to the required form. The search should also encompass *sets* of data sources that a data composer could combine to deduce the required data. A significant challenge in enabling such a search is the development of a comprehensive, uniform *ontology* [Ont00] of data sources.

Exhaustive search of all potential data sources matching a given specification is impossible. As in DNS, the work of discovery must be partitioned, delegated hierarchically, and facilitated with caching. However, the construction of the delegation hierarchy cannot be a human administrative activity as in DNS: The number of potential data sources is comparable in magnitude to the world's population.

There are several approaches to partitioning the search. Astrolabe [Ren00] uses a hierarchy of domains corresponding to the administrative hierarchy of interest to an application. The DataSpace project [Imi99, Imi00] subdivides a region of three-dimensional space near the surface of the earth into nested *datacubes* defined by either geometric or administrative boundaries. However, a data-aggregation infrastructure should be sufficiently flexible to support a wide variety of yet-to-determined applications, and should not be limited by a predetermined partition such as one based on geometric coordinates. Indeed, the data sources we envision include data composers with no inherent coordinates.

2.1.2 Dynamic suitability of data sources

Rapid, scalable remapping of abstract data specifications to physical data sources is a difficult challenge. The set of data sources referred to by a specification may vary dynamically with the context of the user (as with a specification denoting the best driving route from the user's home to the first appointment on the user's calendar for the day) or with changes in the outside world (as with a specification denoting the set of electricity customers willing to pay 5 cents per kilowatt-hour to obtain lower the temperature-humidity indexes). In particular, mobility of a data source may change the set of data sources described by an abstract data specification in either of two ways: Data sources may move in or out of a region encompassed by a data specification, or the specification may encompass a region defined in relation to a mobile data source (say a circle with a 50-foot radius centered at a given person, vehicle, or piece of equipment).

2.2 Selection of a data source

An abstract data specification may resolve to multiple data sources. For example, a person's location might be inferred from a travel itinerary, an office-building badge reader, a GPS device, or cellular-phone data. Once a set of candidate data sources is discovered, one candidate (either a data composer or a direct source of raw data) must be selected. Challenges in selection include efficiently collecting pertinent information and solving a multidimensional optimization problem.

2.2.1 Dynamic factors affecting data-source selection

Like the discovery of data sources matching an abstract specification, the selection of a data source can depend on the context of the user. To monitor the proximity of someone in the same building, the aggregation system might create a data composer that subscribes to a continuous stream of position reports. To monitor the proximity of someone known to be in a distant country, it would suffice to issue individual requests every few hours, switching to progressively more intense forms of monitoring as the subject draws closer.

The choice of data source can also depend on the nature of the query. A data-aggregation system needs to support both snapshot queries and *continuous queries* [Ter92]. Consider the data source for a pulse reading: Individual requests make sense for low-frequency continuous queries such as average pulse readings retrieved daily. Subscriptions to streams of raw device data make sense for high-frequency continuous queries such as average pulse readings retrieved every 10 seconds. Similarly, individual requests make sense for relatively static data like the locations of nearby highways, but subscriptions to device data streams make sense for rapidly changing data such as temperature-humidity-index measurements.

Finally, the choice of a data source may depend on the state of the network. An aggregation system may select data sources in a manner that minimizes network traffic, avoids congested parts of the network, balances computational load, or allocates network resources fairly among competing clients. An end-user could remain oblivious to these considerations.

The DataSpace proposal [Imi99, Imi00] *illuminates* all the objects in a specified datacube having a specified value for a specified network index, by multicasting a request to each such object to *reflect* a response if the object satisfies a specified condition; the possibility of redundant objects supplying identical information, and thus the necessity to select among them, is not addressed. The Intentional Naming System [Adj99] tracks the locations of resources having an optimal value for some application-defined metric, enabling *intentional anycast*, in which a request is tunneled directly to an optimal resource; only a single metric goes into the selection decision, and the application is responsible for computing it.

2.2.2 Tradeoff among selection criteria

There are many factors that may differentiate candidate data sources , for example quality of information (e.g. precision and refresh rate), latency, impact on overall system load, and monetary charges. Selection of a data source entails a tradeoff among several factors, some relating to the interests of a single user or application and some relating to overall system performance. *Directed diffusion* [Int00] routes data in a manner that optimizes system-performance metrics: the energy dissipated by wireless battery-powered sensors, latency of message delivery, and reliability of message delivery. The Odyssey concept of *application-aware adaptation* [Nob00] provides a useful model for mediating between system-wide resource-management interests and user-specific application-adaptation interests.

3 Data composition

The fundamental task of data aggregation is the flexible composition of high-level data out of lower-level data items. Composition performs a logical conversion of available information to required information. In addition, data composition can compensate for imperfect data sources, reconciling readings from different sources observing the same phenomena and computing a consensus value. Finally, data composers enable aggregation systems to scale to

huge amounts of input data in two ways: first by allowing a computation to be split among many nodes whose results are then composed, and second by performing data reduction that replaces large volumes of data with small amounts of summary data.

A set of data composers supporting a computation has a *logical* and a *physical* configuration. The logical configuration consists of the computation performed by each composer, along with the data flow relationships among composers. The physical configuration consists of the placement of composers in the network. The physical distribution of composers may change based on factors such as system load and data source failures while the logical configuration remains the same. In the following sections, we first discuss considerations that influence the creation of data composers. We then discuss the dynamic factors that influence the composition process and necessitate reconfiguration of composers.

3.1 Flexible creation of data composers

Challenges in creating data composers include support for a wide variety of composers, construction of logical and physical composition topologies, and sharing of composers across applications.

3.1.1 Diversity in composition

Data composers must be able to accommodate a wide variety of data sources—such as raw sensor data, newsfeeds, and responses by network services to queries—and the accompanying access protocols. Some data sources are *active*, providing data as it becomes available, and some are *passive*, providing data upon request.

Application data may have a *lifespan*. Consider a data composer that correlates reports of a hotel guest needing a taxi with reports of a taxi driver looking for a passenger, generating an output value when matching reports are received. The detection of the hotel guest and the detection of the taxi driver may not be simultaneous, but a match should be reported if the detections were within 5 minutes of each other. Thus a detection of either kind has a lifespan of 5 minutes.

The work of a data composer may range from simplistic reformatting to a complicated function of several inputs. It may summarize data received in parallel from several different sources or serially from a single data source. Computations can be *stateless* (e.g. the computation of a temperature-humidity index from temperature and humidity readings), can involve *transient state* (e.g. a sliding average of the last 20 pulse readings) or can involve *stateful processing* (e.g. determining whether number of cars passing through a toll booth in a given hour exceeds the average for that time of day by 20%) [Ban99].

The arrival of one piece of time-sensitive data within the lifespan of another is just one example of an *event pattern*. A data composer might generate output values upon the detection of arbitrarily intricate patterns of events. Such patterns can be built out of simple events—such as the arrival of an input or the reaching of a specified time—and recursively defined composite events—such as the occurrence two specified events in sequence or the occurrence of *n* events from among a specified set of alternatives. Active databases [McC89] recognize such patterns in order to execute *event-condition-action rules*.

3.1.2 Composition topology

A data composer is a particular kind of data source. Thus the resolution of an abstract data specification may yield a data composer, either one that already existed (as will be explained in Section 3.1.3) or one that is created as a side-effect of the resolution process. When a new data composer is created, the resolution process is invoked recursively to find input data sources for it. In this manner, the resolution of an abstract data specification received from an application may result in the creation of a *logical* topology in the form of a hierarchy of data composers.

Optimizations can be applied in the construction of the data-composer hierarchy to reduce the system-wide cost of a computation, including the cost of data transmission among distributed composers. These optimizations include traditional database-query optimizations, such as the distributed semi-join reductions used in COUGAR [Bon99], and compiler optimizations, such as *reduction of strength* [Coc77]. Consider the computation of a list of zip codes for all pending service calls at least 8 hours old. A naive computation of this list would construct a list of all service calls, remove those less than 8 hours old, and extract the zip code from each element of the resulting list. A reduction-of-strength optimization might replace this computation with one in which a lower-level data composer for each zip code determines the age of the oldest service call in that zip code, returning 1 if there is a service call at least 8 hours old and 0 otherwise; and another data composer constructs a list of all zip codes whose lower-level data composers returned 1. Similarly, massive computations can be made scalable by decomposing them hierarchically; commutative, associative computations such as sums and maxima can be distributed flexibly among parallel data composers responsible for arbitrarily grouped subsets of the input values.

A hierarchical configuration implies an acyclic data flow among composers, but there are also contexts in which cyclic data flow may arise. For example, the traffic-avoidance application might direct queries to a composer corresponding to the driver's current position, but all such composers might cooperate (through central coordination or a distributed algorithm) to direct traffic in a globally optimal way, rather than directing all traffic to the currently least congested route, and thus congesting it. Therefore, a mechanism is needed for creating a set of mutually dependent *peer* data composers that obtain data from each other. The specification and synchronization of groups of peer composers are formidable challenges.

Several considerations affect the *physical* realization of a logical composition topology. For example, bandwidth considerations imply that a composer that computes traffic density from sensor data should be placed on a node near the sensors. However, placement is less critical for a logically equivalent composer that computes traffic density by querying cellular-phone data, since bandwidth issues are then delegated to the cellular phone infrastructure. Some computations may involve the organization of composers into affinity groups, characterized by frequent communication within groups and infrequent communication between groups. Composers in the same group should be placed on nearby nodes to minimize latency. In addition, for overall system reliability, stateful composers may be *replicated* on distinct nodes.

3.1.3 Shared Data Composers

Creating and initializing a data composer may be expensive, so it is attractive to share data composers across applications. Some composers may be *transient*, i.e., created in response to a specific application request and dispensable upon completion of that request. Transient

composers can be shared across applications or between different requests from the same application. It is challenging to determine appropriate tradeoffs between the costs of maintaining such composers in the system and the costs of destroying and recreating the composers.

Some composers may be *persistent*. For example, a long-running service-providing application might create several data composers at application start-up time, in anticipation of need; some data composers fundamental to many applications might be permanently installed by a system administrator as part of a service layer; and some active data composers might be public Internet resources. A given data consumer may be interested in only a filtered subset of the output values generated by a shared data composer. The transmission of each item of composed data to only those data consumers interested in receiving it is similar to the content-based matching problem studied in the context of publish/subscribe networks [Ban99].

In addition, a data composer might cache the values it computes. A shared composer that caches computed values faces the challenge of determining an appropriate cache invalidation/refresh policy. Minimization of network traffic would suggest that a cache should be invalidated only when there has been a change to the cached data. However, expiration of cached values after a specified amount of time, and periodic retransmission of the values, prevents the retention of stale data if a data source stops transmitting new values because of a sensor failure; retransmission also corrects for data loss in the network.

3.2 Dynamic reconfiguration of composers

The application domains envisioned for pervasive data aggregation are highly dynamic. Data sources and consumers are mobile, resulting in changing context. The overall system, including data sources, data composers, and network resources, is prone to degradation or failure. The logical and physical configurations of data composers must therefore be dynamic to accommodate the dynamic nature of the domain.

3.2.1 Implications of mobility and contextual variation

Mobility affects both the physical and logical configurations of data composers. Consider the *physical* configuration of data composers in the medical-alert example: High volumes of data might be sent from a device carried by the patient to a nearby edge node, where a data composer would summarize and filter the data, drastically reducing the amount of data to be propagated across the network. The composer performing the data reduction would migrate from one edge node to another as the patient travels, to minimize network congestion. Mobility can also affect the *logical* configuration of data composers: As serviced equipment is moved from one service team's territory to another, its status becomes part of the nearest-service-call computation for a different set of service vehicles. Mobility can render a data source temporarily unreachable, requiring data composers to compensate for the absence of its data, perhaps by interpolation from nearby data sources.

Contextual factors such as *time* may affect configurations of data composers. The most efficient logical (and physical) configuration of composers for the traffic application may vary between rush hours and off-peak hours. For the proximity-monitoring application, if an appreciable number of subscribers turn off detection at some point, the set of data composers supporting the application may need to be reconfigured for efficiency.

3.2.2 Failure and degradation of resources

Sensors may break, lose accuracy, run out of energy, or move out of range of a wireless network connection. Higher levels of a data-aggregation system must be prepared to accommodate such failures. As Estrin *et al.* [Est99] point out, the high ratio of devices to users makes it impossible for humans to monitor and manage devices; the system as a whole must be *exception-free*. In the traffic example, when roadway sensors are unavailable, a different composer may need to be created that computes traffic density from cellular-phone positioning data, although the higher levels of the composition hierarchy remain unchanged. In other cases, composers may provide *fusion* services similar to ones proposed by Castro and Muntz [Cas00] that use Bayesian networks to construct joint probability distributions based on the input from multiple sensors; or cooperating composers might adopt conflict-resolution strategies, such as voting among multiple sensors.

Resources such as network bandwidth may be degraded. The physical topology might be reconfigured to reduce data traffic, perhaps at the cost of some other performance metric (for example, moving computations to processors that are less powerful, but closer to the source of the data). A data composer might respond to network degradation by bundling output values into groups and transmitting only the average value for each group. A similar, but less deterministic, approach is *samplecast* [Imi99, Imi00]: Input sources satisfying a query are directed to respond to the query with probability p, and the number of responses actually received is multiplied by $1/p$ when computing aggregate results.

4 Data security

Applications such as proximity matching and service-call dispatching entail tracking the whereabouts of people, which raises obvious privacy concerns. Some privacy issues may be adequately addressed by existing role-based access control techniques, but a flexible aggregation system will require enforcement of more powerful privacy policies. First, *fine-grain* policies must be enforced that allow users to get degraded views of restricted data. For example, some workers might want to give their families complete access to their whereabouts, give business associates access only to an out-of-the-office flag, and give strangers access to no location information. Some trusted data composers might have authorization to read restricted data and to generate degraded, less restricted data . Formalizing the notion of degraded data and enforcing degradation in data composition is a difficult challenge. Second, *dynamic* policies must be enforced that take broad context into account. For example, tighter restrictions might be imposed on detailed traffic information in a city during the visit of a dignitary, to avoid the distribution of information that could be used to infer the dignitary's location. The aggregation system must also address data-security issues such as confidentiality of composed business data (e.g., the frequency of service calls).

5 A programming model for dynamic aggregation

At the core of an aggregation system is it ability to facilitate the development of applications. A good programming model will make it easy for an application writer to specify the required computations at a high level, yet allow the system to be fine-tuned to optimal efficiency. It will allow applications to evolve through a series of localized changes. A data-aggregation programming model must provide for the abstract specification of data sources, data-combining computations to be performed by a data composer, event sequences to be recognized by a data

composer, distribution of data composers throughout the network, and adaptations to the logical and physical configuration of data composers

A computation combining values in a *time-independent* manner is naturally specified by an expression connecting abstract data specifications with high-level operators. By specifying as little as possible about *how* the value is to be computed, we maximize the freedom of the data-aggregation system to optimize the computation. As Backus [Bac78] notes, a functional specification of a computation is conducive to formal reasoning and transformation. For a computation combining sequences of arriving data items in a *time-dependent* manner, composite-event specification schemes developed for active databases, such as Ode [Geh92], Snoop [Cha94], and SAMOS [Gat94], may be appropriate.

6 Related work

6.1 Underlying technology

Much of the interest in aggregation of sensor data stems from research in Wireless Integrated Network Sensor (WINS) technology [Def99]. Pottie and Kaiser [Pot00] address the electrical engineering and network engineering issues in WINS, and Kahn, Katz, and Pister [Kah99] address such issues for the closely related research in "Smart Dust." Estrin *et al.* [Est99] address the programming of WINS applications, particularly the self-organization of sensors, rapidly deployed in a battlefield or emergency-management situation, to establish communication paths and coordinate to solve a narrowly defined problem, while minimizing energy consumption. Bonnet, Gehrke, and Seshadri [Bon99] presume a WINS architecture for the COUGAR device database system. While the abundance of sensor data enabled by WINS technology is one motivating force behind our work in data aggregation, our work starts at the edge of the network, once sensors have already been manufactured, deployed, and configured. While the mobility of wireless sensors is a key factor in many of the challenges we have discussed, we are equally interested in aggregating data from wired sensors, and from network sources other than sensors.

6.2 Data resolution

Mechanisms for identifying a data source and obtaining data depend deeply on the underlying data model. The COUGAR Device Database System project [Bon99, Bon00] views sensor data readings as rows in a distributed active relational database, and uses a modified form of SQL to retrieve data. The DataSpace proposal [Imi99, Imi00] views sensor readings as attributes of physical objects addressed at the network level, through IPv6 multicast addresses that correspond to the geometric coordinates of these objects and their values for certain attributes that serve as *network indexes*. In the directed diffusion model of data acquisition [Int00], a notice of *interest* in data satisfying certain attributes is *diffused* through a sensor network along paths suggested by the attributes, establishing weighted *gradients* along which data is returned from sensors. None of these approaches seems to envision the potential of a data specification that might be resolved dynamically, without application involvement, to either an external source (such as a sensor) or to a data composer.

The Intentional Naming System [Adj99] is a resource discovery system in which the resources to be discovered are described by abstract specifications. The Intentional Naming System provides applications with the option of binding an abstract specification to a resource at

the last possible moment, message-delivery time, rather than during an earlier look-up step, through intentional anycast. The design point of the current Intentional Naming System implementation is the discovery of physical entities such as printers and cameras within a local network domain. It remains to be seen how this approach scales to a wide-area network with highly dynamic physical and logical resources.

In the Mobile People Architecture [Man99], a globally unique personal identifier is resolved by a simple directory lookup to an *application-specific address* through which that person can currently be reached. This approach does not scale from the number of potential mobile people in the world to the number of potential data sources, or from the specification of an individual to more complex combinations of properties that might be used to specify data sources.

6.3 Composition

The concept of hierarchical data composition is well known. The Astrolabe system [Ren00] is based on a composition hierarchy in which the leaves correspond to directly measured data and internal nodes correspond to *condensation functions* that compute data from the data provided by child nodes. A DataSpace supports *query zooming*, the retrieval of imprecise information about a large datacube or more precise information about a smaller datacube nested within it, implemented by parent datacubes aggregating, summarizing, and caching responses to a query from its child datacubes [Imi99]. Both these approaches use relatively static hierarchies, while we envision highly dynamic, adaptive hierarchies.

Gathercast [Bad98] is a network service providing a form of aggregation at the packet level. It optimizes the transmission of short, highly redundant messages from many points upward through a tree to a single point by passing messages from a node to its parent through optimizing *transformer tunnels*. Such efforts at the network level may effectively supplement the optimizations we envision above the network level, based on the semantics of data composers and active reconfiguration.

The notion of a continuous query was introduced by the Tapestry system [Ter92]. OpenCQ [Liu99] specifies composer-like computation nonprocedurally with a continuous query. OpenCQ does not address dynamic reconfiguration or distribution, although [Liu99] promises future work in distribution.

CORIE [Ste00] is a working system that aggregates environmental sensor data. CORIE must cope with high data volume, strict bounds on latency, quality-of-service adaptations to the needs of currently running applications and the resources currently available, sensor failure, and failure of intermediate nodes in the network. However, the problem solved by CORIE is simpler than the problem we have described, because all CORIE data is directed from fixed sensors to a fixed central node and processed by a small number of well-known applications. Central processing introduces a single point of failure, and we do not believe it will scale, even if data is routed through an efficient content-routing network [Ban99].

7 Conclusions

We have described our vision of the challenges in flexibly, scalably and reliably aggregating massive amounts of data from diverse networked data sources. We propose the

creation of systems that can facilitate the writing of applications that depend on this data. The system will provide basic services to application writers and adapt flexibly to changing conditions.

Applications will refer not to specific data sources, but to abstract data specifications. The data-aggregation system will resolve these specifications, by dynamically discovering data-source candidates and selecting from among the candidates discovered. Data composers will combine input values to produce output values. Such data composers are themselves data sources, and may be created by the process of resolving data specifications. Applications will specify the behavior of data composers nonprocedurally, and the system will apply optimizations based on mathematical properties, the state of the system, and the external context. Both the logical organization of data composers and their physical distribution throughout the network will be highly dynamic. A data-aggregation system will react to changes in the network and in the world being measured, and will reconfigure the data-aggregation machinery accordingly.

We have recently started a project named *iQueue* to address the challenges that we have described. The iQueue system will facilitate the writing of applications aggregating pervasive sources of data and feeding pervasive consumers of data. The system will provide services to resolve abstract source and destination data specifications, compose data, and enforce access controls.

References

[Adj99] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99), December 12-15, 1999, Kiawah Island Resort, South Carolina, published as *Operating Systems Review* **33**, No. 5 (December 1999), 186-201

[Bac78] John Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM* **21**, No. 8 (August 1978), 613-641

[Bad98] B.R. Badrinath and Pradeep Sudame. Gathercast: an efficient multi-point to point aggregation mechanism in IP networks. Technical report DCS-TR-362, Department of Computer Science, Rutgers University, July 29, 1998

[Ban99] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. Proceedings of the 19th International Conference on Distributed Computing Systems, May 31 - June 4, 1999, Austin, Texas, 262-272

[Bon99] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Query processing in a device database system. Technical Report TR99-1775, Cornell University, October 1999

[Bon00] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Querying the physical world. *IEEE Personal Communications* **7**, No. 5 (October 2000), 10-15

[Cas00] Paul Castro and Richard Muntz. Managing context data for smart spaces. *IEEE Personal Communications* **7**, No. 5 (October 2000), 44-46

[Cha94] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim. Composite events for active databases: semantics, contexts and detection. **In** Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, eds., *VLDB '94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, Morgan Kaufmann, San Fransisco, 1994, 606-617

[Coc77] John Cocke and Ken Kennedy. An algorithm for reduction of operator strength. *Communications of the ACM* **20**, No. 11 (November 1977), 850-856

[Def99] Defense Advanced Research Projects Agency. WINS: Wireless Integrated Network Sensors. <URL: http://www.darpa.mil/ito/psum1999/h578-0.html>. December 2000.

[Est99] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: scalable coordination in sensor networks. Proceedings of the fifth annual ACM/IEEE international conference on mobile computing and networking, Seattle, Washington, August 1999, 263-270

[Gat94] Stella Gatziu and Klaus R. Dittrich. Detecting composite events in active database systems using Petri nets. Proceedings, Fourth International Workshop on Research Issues in Data Engineering, Houston, Texas, February 14-15, 1994, 2-9

[Geh92] N. H. Gehani, H. V. Jagadish and O. Shmueli. Event specification in an active object-oriented database. Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992, 81-90

[Imi99] Tomasz Imielinski and Samir Goel. DataSpace - querying and monitoring deeply networked collections in physical space, part I - concepts and architecture. Technical Report DCS-TR-381, Department of Computer Science, Rutgers University, October 14, 1999

[Imi00] Tomasz Imielinski and Samir Goel. DataSpace: querying and monitoring deeply networked collections in physical space. *IEEE Personal Communications* **7**, No. 5 (October 2000), 4-9

[Int00] Chalermek Intanagonwiwat, Ramesh Govindan and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. Proceedings of the sixth annual ACM/IEEE international conference on mobile computing and networking, Boston, Massachusetts, August 6-11, 2000, 56 - 67

[Kah99] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next century challenges: mobile networking for "Smart Dust". Proceedings of the fifth annual ACM/IEEE international conference on mobile computing and networking, Seattle, Washington, August 1999, 271-278

[Liu99] Ling Liu, Carlton Pu, and Wei Tang. Continual queries for Internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering* **11**, No. 4 (July/August 1999), 610-628

[Man99] Petros Maniatis, Mema Roussopoulos, Ed Swierk, Kevin Lai, Guido Appenzeller, Xinhua Zhao, and Mary Baker. The Mobile People Architecture. *Mobile Computing and Communications Review* **3** , No. 3 (July 1999), 36-42

[McC89] Dennis McCarthy and Umeshwar Dayal. The architecture of an active database management system. Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989, 215-224

[Nob00] Brian Noble. System support for mobile, adaptive applications. *IEEE Personal Communications* **7**, No. 1 (February 2000), 44-49

[Ont00] Ontology.Org frequently asked questions. <URL: http://ontology.org/main/papers/faq.html>. December 2000.

[Pot00] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM* **43**, No. 5 (May 2000), 51-58

[Ren00] Robbert van Renesse. Astrolabe. <URL: http://www.cs.cornell.edu/Info/People/rvr/astrolabe/>. December 2000.

[Sen00] Sensoria Corporation. Sensoria: technology. <URL: http://www.sensoria.com/technology.html>. December 2000.

[Ste00] David C. Steere, Antonio Baptista, Dylan McNamee, Calton Pu and Jonathan Walpole. Research challenges in environmental observation and forecasting systems. Proceedings of the sixth annual ACM/IEEE international conference on mobile computing and networking, Boston, Massachusetts, August 6-11, 2000, 292 - 299

[Ter92] Douglas Terry, David Goldberg, David Nichols and Brian Oki. Continuous queries over append-only databases. Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992, 321-330