

A System Architecture for Distributed Control Loop Applications (Extended Abstract)

Umakishore Ramachandran

Phillip Hutto

Bikash Agarwalla

Matthew Wolenetz

College of Computing
Georgia Institute of Technology
Atlanta GA 30332-0280
rama@cc.gatech.edu

Abstract

In this position paper we motivate an important emerging class of applications that cooperate across a complex distributed computational fabric containing elements of widely varying capabilities, including physical and virtual sensors, actuators, and high-performance computational clusters and grids. We identify typical requirements of such applications and identify several novel research challenges that such applications pose. We sketch an evolving architecture developed as part of the Media Broker project at Georgia Tech that solves a subset of the problems presented.

1 Introduction

Hardware developers continue to press the extremes of large and small, making available a complex *computational fabric* marked by a pervasive, networked hardware continuum including sensors, embedded controllers, handhelds, wearables, laptops, workstations, servers, clusters, and grids. We are interested in developing seamless programming support for computations on this “broad spectrum” computational fabric. We are particularly interested in the confluence of two well-developed but seemingly disparate “worlds” of computational applications and research: high-performance and sensor-based computing.

High-performance computing has become a mainstream discipline concerned with the efficient processing of high-value data and utilizes techniques such as transactions, checkpointing, and consistency maintenance to ensure data integrity, reliability and recoverability. Sensor-based computing emerged historically from hardware-centric environments and military applications, often involving significant signal processing components. Sensor processing environments feature high-volume, transient data with varying data rates and accuracies. Techniques such as aggregation, sampling, data reduction, and fusion are frequently used to reduce data volume and summarize information content. These two worlds are clearly and inevitably merging with the increasing prevalence and ubiquity of sensors and embedded systems.

There is an interesting philosophical conflict in this merger. The sensor processing world understands the value of “throwing away” data as well as the value of the expeditious processing of time-sensitive data. Techniques for skillfully handling approximate data and data of limited or unknown accuracy are well-developed. In contrast, the mainstream high-performance computing world is very much concerned about keeping data around, expecting everything to be used! Given the meeting of sensor-driven and high-performance computing, there is a need to merge the techniques and idioms used in the two worlds and perhaps discover unifying tricks for the merged sensor-based distributed computing fabric. In this merged world, interfacing and integrating with the physical environment will be a key property of high-performance distributed computing.

We expect more and more applications of the future to take on a “control loop” flavor in the following sense: inputs from a variety of distributed sensors may be gathered, filtered, processed, correlated, and fused to facilitate higher levels of hypotheses and inferences (implying heavy duty processing) which may then be used to drive distributed actuators to affect the physical world in some application-specified manner. This mirrors the classic control loop idea, borrowed from control theory, in which an application attempts to maintain a “setpoint” through a combination of sensors and actuators. A familiar example of such a system is a home heating system featuring a sensor (thermostat) and actuator (heater) which attempts to maintain a fixed temperature setpoint. Examples of distributed applications with this control loop character include mobile robots, smart vehicles, aware environments, disaster response, surveillance, and environmental monitoring. We believe this is an important emerging application class.

This extended abstract is an exposition of the programming models and higher level abstractions that help applications deal with a world of distributed sensors and actuators. We structure this abstract as a “position paper” instead of just reporting on a particular research artifact in keeping with the theme of the workshop. The primary contribution of this paper is in articulating and clarifying the application domain, research questions and issues associated with the uni-

fied programming of environments that merge sensors and high-performance computing. In addition, we offer some initial thoughts on appropriate infrastructure for such an environment in the way of a general architecture.

Research is already underway on several of the problems enumerated in this abstract as part of the *Ubiquitous Presence* project (<http://www.cc.gatech.edu/ubiq-presence>) at Georgia Tech. This paper contains a sketch of one of the research artifacts we are currently working on in the context of this project called the *Media Broker*, a clearing house for sensors and actuators that offers a variety of capabilities mentioned in the next section. A detailed exposition of the Media Broker is forthcoming.

2 Research Problems

Complex control loop applications require the acquisition, processing, synthesis, and correlation (often *temporally*) of streaming high bandwidth data such as video and audio, as well as low-bandwidth data such as temperature readings or haptic sensor pressure samples. These applications often require the coordinated processing of multiple streams of data of differing bandwidth and frequency, and with differing failure, accuracy, and latency characteristics. Such applications usually span a multitude of devices that are *physically distributed* and *heterogeneous*. Different kinds of sensors (and data aggregators located near them) collect raw data and perhaps do limited processing such as filtering. However, extraction of higher order information content from such raw data requires significantly more processing power. For example, microphones may collect audio data, but higher order processing is needed for voice recognition. Thus there is a continuum of computation and communication resources from tiny computationally limited sensors to high-performance back-end clusters. A complex optimization problem arises in attempting to efficiently map processing elements to available resources while meeting application requirements. This optimization problem is made more complex with regards to power-utilization because the environments we consider contain a mixture of wired and wireless components. Distributed control loop applications are also highly dynamic. For example, in a telepresence application participants may join and leave a chat session at different times requiring adaptation and restructuring of ongoing computation.

We aim to support mid- to large-scale computations, including hundreds to tens of thousands of elements (sensors, actuators, computational resources). Solutions should be easily scalable from a room to a small home up to campus or corporation scale. As a specific example, consider a sensed environment and associated computational infrastructure on the order of the space shuttle which contains something like 8000 sensors. We imagine similarly instrumented environ-

ments in more mundane settings with much more generalized and changing application requirements. Indeed, a single instrumented environment should support simultaneous access of multiple distinct applications, cooperatively sharing sensors and actuators. If we think of endpoints producing sensor data as servers or services and applications consuming sensor data as clients, our environment commonly gives rise to a “multi-client, multi-server” architecture in which a single client access multiple sensor sources and a single sensor source might be simultaneously accessed (shared) by multiple clients.

An important and often forgotten challenge is the massively distributed programming problem that such a vision engenders. The unique characteristics of this emerging class of massively distributed applications calls for novel solutions. We present this *distributed programming challenge* as a set of research questions that need investigation.

- **Programming Abstractions.** What are the right programming abstractions that seamlessly transcend the hardware continuum from sensors in the environment to powerful back-end clusters in machine rooms? Issues include temporal guarantees for data, efficient support for stream data, and distributed synchronization. The primary goal is simplify the development of applications spanning such a distributed continuum easy. Some sensors may be mobile and may have limited computational capabilities and energy resources. Yet we may want a uniform programming environment that allows development and deployment of distributed applications spanning this computation and communication resource continuum. These environments seem to require multi-paradigm programming supporting fine- and coarse-grained parallelism, distribution, real-time characteristics, and low-level device access and control across computational platforms of widely varying characteristics.
- **Naming and Resource Discovery.** How do we name entities in this massively distributed environment? There have been several recent research ideas at the networking and sensor level to name entities that are not based on static or global identifiers such as IP addresses. These ideas need to be elevated and supported by high-level abstractions in the programming language or runtime system. Indeed, there may be several layers to the naming problem. For example, a computation may specify that it wants to get video input from the north-west part of campus. Such a high-level programmatic intention should be transparently translated to an appropriate logical *channel* defined by the programming abstraction. The logical channel should translate to the low level address associated with the physical camera that provides this video input. Naming schemes should be conducive to associated, flexible, scalable authorization and

resource scheduling components. For example it should be possible for groups or zones of devices to have access authorized to individuals or groups of interested applications. Resources should be addressable by a variety of pre-programmed or dynamically acquired and changing attributes. For example (combining the various ideas presented so far), applications should be able to request access to idle or lightly utilized resources of a particular type (e.g. pan-tilt-zoom camera) in a particular geographic region (e.g. first-floor atrium of the Computer Science building on the Georgia Tech campus) controllable by a particular authorization group (e.g. Computer Science faculty members). Note that in this example the device is both a sensor and an actuator. Such combinations are not unusual. For example, the speaker and microphone attached to a particular workstation are often controlled and accessed as a unit.

- **Adaptive Data Fusion.** The environment we envision consists of a highly complex, distributed collection of fixed and mobile sensors and actuators that are adaptively organized over time and space. A supporting architecture should enable the large-scale collection, archiving, and processing of live streams of sensor data for on-line and off-line analysis, decision making, and dissemination. A common technique, heavily utilized in the sensor processing world, is the filtering and fusion of streams of sensor-produced data. Filters take a single input stream and produce a single output stream by sampling, selecting, aggregating or, more generally, transforming the input stream. Data reduction is a common use of filtering.

A related concept is *sensor fusion*. Fusion is a generalization of filtering that applies an arbitrary computation to “correlated” inputs from two or more input streams to produce a single output stream. Thus, fusion is fundamentally a multiplexing or “fan-in” operation. Fusion supports spatial aggregation while filtering supports temporal aggregation. A equivalent demultiplexing or “fan-out” operation is possible but less common and is used for sharing of sensor data streams. Fusion operations with physical sensor inputs can be viewed as producing or defining a new “virtual” sensor with its own distinct characteristics and uses. Typically sensors are virtualized for several reasons. Readings from identical (replicated) sensors can be combined to enhance accuracy and precision or to increase fault-tolerance. Readings from similar sensors with overlapping “fields of view” can be collaged to create a virtual sensor with enhanced range of coverage. Finally, multiple sensors of distinct types or modalities can be composited to produce enhanced information. For example, consider a location tracking system that combines audio, video, thermal and infrared tag-tracking inputs to pro-

duce a highly accurate “sample” of a subject’s current location. Finally, note that various “software sensors” are being used to sense, monitor, and track software-generated activities. Examples include intrusion detection systems, web update monitoring, and network traffic management. Our application domain requires support for all these forms of filtering, transformation and fusion, of both real and virtual sensor data.

These types of sensor processing must often be performed in constrained environments with limited resources. Also to facilitate higher level inferencing (a common use of sensor data) there is a need for application-specific data fusion in the presence of insufficient and error-prone data. Inferences from fused data should support risk analysis and should allow the computation of confidence interval on resulting conclusions. Analysis and interpretation of collected data in conjunction with information databases (that could often be out of date and/or have partial or incorrect information) is essential in enabling critical decision making often involving humans in the loop. Other essential components of the information architecture include novel data models, segmentation, and organizational methods for storing and mining collected sensor data in conjunction with existing databases.

- **Adaptive Distributed Plumbing.** How do we orchestrate the dynamic connectivity of the components participating in such a massively distributed computation? Sensor filtering, sharing, fusion can give rise to complex data flow graphs that may need to change dynamically in response to application imperatives, adding or removing devices, or to optimization communication or power. With advances such as aspect oriented programming, we are just beginning to tackle the problem of plumbing programming components in a dynamic setting. The .NET framework from Microsoft and supporting third party tools are approaching the level of maturity for component-based programming on the internet with the Web as the primary focus. We are not even at the stage of clearly articulating the plumbing problem in massively distributed computations. For one thing, the end point of communication may not always be the same physical device. The camera that was supplying video input in the above naming example may fail. In this case, transparent to the application, the programming system should change the end point of communication to another camera which is available in the vicinity, priming the new camera with the state from the old one.
- **Failure Semantics.** What is the right computational model for dealing with failures? Real failures and perceived failures (due to poor response times) are expected to be normal occurrences in such an environment. Indeed, physical sensors routinely degrade and fail.

Battery-powered devices are even more prone to (temporary) failure (although it may be possible to anticipate such failure.) Traditional approaches to fault tolerance and recovery may be neither applicable nor scalable to a setting in which there are thousands of sensors and actuators. How do we design self-monitoring, self-tuning, and self-healing computational abstractions? Fundamentally new computational models are needed that will allow reasoning about software in the presence of failures. As mentioned previously, an individual sensor sample is often of low-value and an occasional lost or corrupted sample is expected. High rates of corrupt data may indicate sensor damage or failure. Communication to actuators usually required higher reliability guarantees. Also, sensors and actuators should be protected from unauthorized access or use. So the presence of malicious applications should be addressed with appropriate restraints.

- **Runtime Mechanisms.** What new runtime mechanisms are needed to support such massive distributed programming? To support adaptive plumbing the runtime has to be able to package the state information from a resource-constrained or a failed component and move it to another end point. This is just one instance of new runtime mechanisms that may be needed. There may be several new mechanisms needed to support such a programming continuum including program and data transformations, computation elision where applicable, and state reconstruction in the presence of failures.
- **System Evaluation.** How do we evaluate such large-scale systems? There are several axes along which such evaluation needs to be done: ability to meet application level guarantees, availability in the presence of real and perceived failures, and performance and scalability of the runtime mechanisms. While none of these dimensions are fundamentally new for system evaluation, the scale of future systems poses interesting challenges in coming up with new techniques for carrying them out. For example, the processing speeds of sensor-based task graphs are often limited by the lowest frequency input sample rate. Showing that a support infrastructure can process data significantly faster than the highest input sample rate is of limited value.

3 Solution Approach

The solution approach is two-fold: First, we are asking a number of research questions that will help us arrive at intuitive data abstractions and an appropriate API for manipulating them. Second, we are working with application researchers in vision, robotics, and human-computer interfaces to deploy the systems technologies we

develop and evaluate them in an application context. We have testbeds available for this exercise such as the *aware home* (<http://www.cc.gatech.edu/fce/ahri>) at Georgia Tech.

3.1 Media Broker Architecture

In this subsection, we present the preliminary architecture of a subsystem we are building called the *Media Broker* that is aimed at addressing a subset of the problems listed in Section 2 including adaptive naming, resource discovery and data fusion. The Media Broker (see Figure 3.1) acts as a clearinghouse for sensors and actuators. It supports data and device naming and efficient resource discovery and provides support for efficient sensor data fusion. The Media Broker is being built on top of the D-Stampede [1] system but the architecture is not specific to the implementation environment. We have chosen to focus on two issues that we believe to be fundamental. We feel that naming and resource discovery should be elevated to a higher level of programming abstraction than is typical. Data fusion is also an important and natural metaphor and a convenient way to realize virtual sensors.

The Media Broker architecture embodies several key features:

- **Fusion Channels and Virtual Sensors.** The Media broker architecture provides an abstraction called fusion channel. A fusion channel is a named, global entity that abstracts a set of inputs and encapsulates a programmer-supplied fusion function. Inputs to a fusion channel can come from a variety of sources (including other fusion channels). Item fusion is automatic and is performed according to a programmer-specified policy either on request (demand-driven, lazy, pull model) or when input data is available (data-driven, eager, push model). Items are fused and accessed by timestamp (usually the capture time of the incoming data items). An application can request an item with a particular timestamp or in a more general way (earliest item, latest item). Requests can be blocking or non-blocking. To accommodate failure and late arriving data, requests can include a minimum number of inputs required and a timeout interval. If sufficient inputs are not available by timeout expiry, partial fusion can be performed on the available inputs. Fusion channels have a fixed capacity specified at creation time. Finally, inputs to a fusion channel can themselves be fusion channels, creating fusion networks or pipelines. The fusion channel takes a number of inputs, applies an application-specified fusion function to them, and generates a fused output. A fusion channel can also be used to transform a single input into a different type via the fusion function.

Creation of virtual sensors may require dynamic code loading in the media broker. The application program-

mer can, at run time, load code that fuses data from one or more sensors. Such virtual sensors are addressable by higher level names available globally. In addition to allowing dynamic fusion channel creation, the Media Broker also supports dynamic resource discovery and sharing among many concurrently running applications.

- **Resource Discovery and Sharing.** The Media Broker architecture provides a resource discovery mechanism that allows application level queries to result in connections to the appropriate logical channel, which is either a real sensor's output or a virtual sensor's fused output. Any sensor, real or virtual, in this architecture can elect to produce more than one kind of output. For example, a camera could register multiple resolution, compression, and pixel format capabilities with the Media Broker. Similarly, an image analysis virtual sensor (fusion channel) could register as a face recognizer and a face detector. Sensors register their output format hierarchies, including any available virtual sensors that perform fusion function transformations between native sensor formats. Sensors are notified by the Media Broker to produce an output compatible with all current requests by consumers, where consumers in this context may be an actuator, an input to a fusion channel, or even a feedback input into a sensor. This process streamlines sharing of a real or virtual sensor's constrained resources.

- **Sensor-Provided Attributes.** Relying on a single communication protocol such as TCP/IP to handle all potential forms of Media Broker communication with sensors and actuators may limit the degree to which application level guarantees are satisfiable. For example, latency-sensitive audio conferencing applications may benefit from using RTP[2] to maintain realtime guarantees with the risk of losing audio quality. The benefit of using RTP instead of TCP/IP as the underlying protocol in a Media Broker audio conferencing application grows when the communication occurs over a wireless link. Anticipating the need for heterogeneous communication protocols, each real sensor may include protocol differentiation options as part of the capabilities it registers with the Media Broker.

A sensor may provide other attributes such as a location based name, membership in a sensor group, and hardware capabilities via a flexible registration schema. A sensor may also provide the Media Broker with dynamic attributes such as remaining energy and processor load. A potential consumer may query the media broker based on these attributes to decide which sensor it wants to request.

- **Sensor-Derived Attributes.** The Media Broker monitors each sensor to determine latency, bandwidth, and usage, including current consumers and output format.

These dynamic attributes are included in the set of attributes available for queries by potential consumers. For example, a potential consumer may query to find the most popular sensor within a sensor group.

- **Scalability.** To support scalability, multiple Media Brokers can be federated with peer-to-peer interactions within logical domains or zones. We plan on providing the ability for clients to migrate between Media Brokers for load balancing and to allow active Media Brokers to be deactivated. Media Brokers participating in resource discovery can be requested to forward unsatisfied queries to peer Media Brokers, in a manner similar to recursive Domain Name System queries. In addition, Media Broker zones can be arranged in a hierarchical fashion to support cross-zone queries and client migration.
- **Access Control and Resource Scheduling.** Resources are attributed with access lists that identify subjects (individuals, roles, groups) and associated access permissions. Access permissions include capabilities for sampling, actuating, and configuring. Subject attributes include an access priority. Access requests specify shared or exclusive access with access granted using standard reader/writer semantics. Conflicting accesses are resolved by subject priority. Competing accesses at the same priority are scheduled. This requires access requests to include start and end times. The system supports resources status queries and access requests include options for blocking, non-blocking, and asynchronous access. Asynchronous access in the Media Broker API generates a notification to the requester when the resource is available and access is granted. Expected availability is returned synchronously when the resource is currently in use for fixed time period. Access revocation rights can be granted to certain subjects to supercede current user access. To avoid deadlock and handle failure, the API uses a transparent leasing protocol between clients and the Media Broker to ensure clients continue interest and participation. Deadlocks are not currently detected between clients with exclusive access to resources.
- **Security.** We envision the Media Broker to span a wide area network. Especially in this context, security and privacy become important issues. The Media Broker architecture has built-in authentication of applications when they make their initial connections. All of the communications are done securely.

4 Conclusion

We expect many mainstream distributed applications of the future to take on a distributed control loop flavor

involving sensors, actuators, and a complex, dynamic task graph of processing, filtering, transformation and fusion, often requiring complex cluster-based computations. In this extended abstract, we have presented a system architecture that embodies many salient issues relevant to such future systems. We have also presented a preliminary *Media Broker* architecture that seeks to address a subset of these issues including adaptive naming, resource discovery and data fusion.

References

- [1] S. Adhikari, A. Paul, and U. Ramachandran. D-Stampede: Distributed programming system for ubiquitous computing. In *Proc. 22nd International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.
- [2] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. *RFC 1889*, January 1996. URL: <http://www.ietf.org/rfc/rfc1889.txt>.

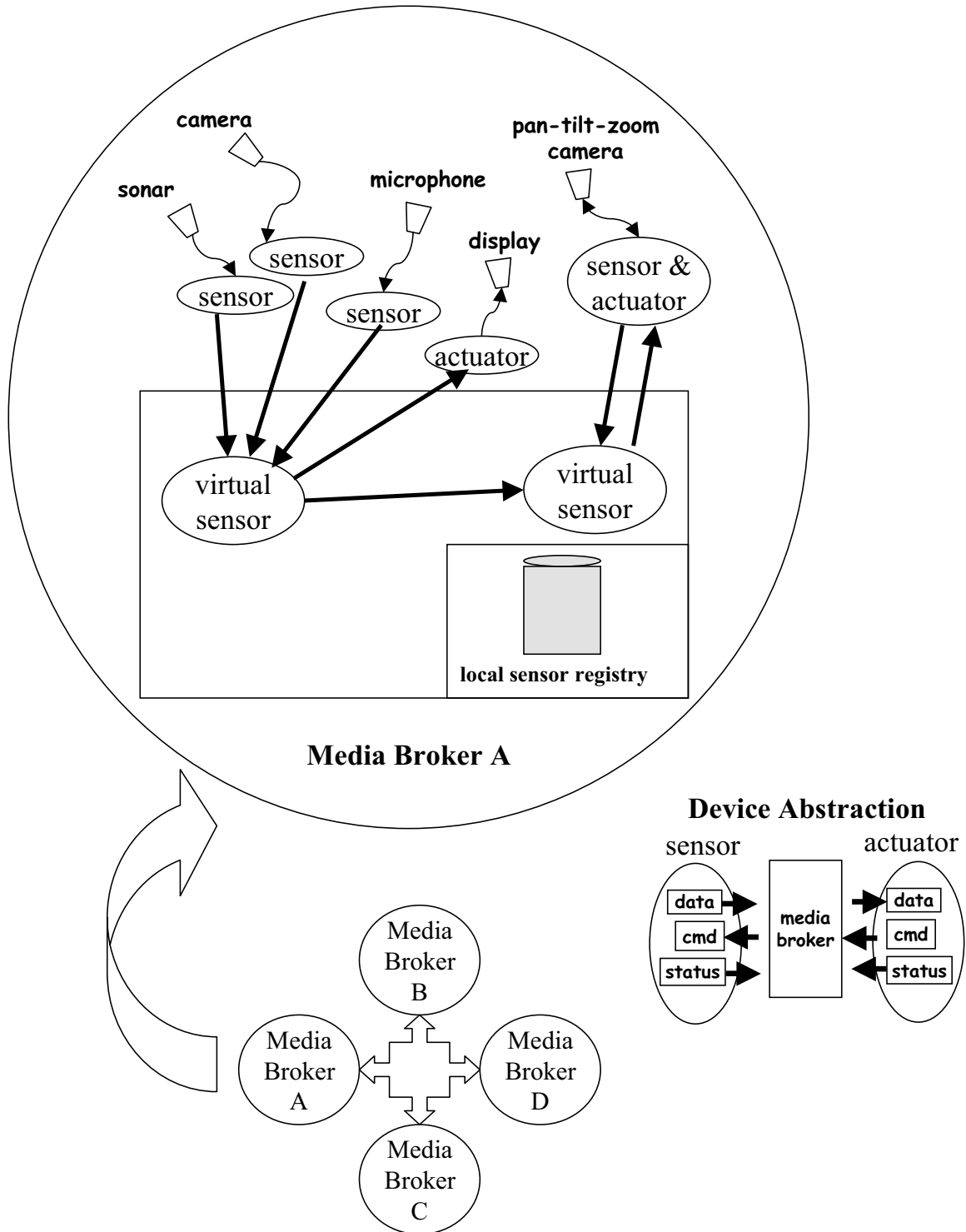


Figure 1: Media Broker Architecture