

# Towards Middleware Components for Distributed Actuator Coordination

Joel W. Branch and Boleslaw Szymanski  
*Department of Computer Science  
Rensselaer Polytechnic Institute  
{brancj, szymansk}@cs.rpi.edu*

Chatschik Bisdikian, Norman Cohen, John S.  
Davis, Maria R. Ebling, and Daby M. Sow  
*IBM T. J. Watson Research Center  
{bisdik, ncohen, davisjs, ebling,  
sowdaby}@us.ibm.com*

## Abstract

*The pervasive presence and availability of sensor and actuator networks creates the potential for widely distributed Internet-scale control systems. Given these systems' potential magnitude and complexity, new tools will be required to facilitate their composition and operation, especially as they relate to distributed actuator coordination. This paper introduces and advocates our approach to implementing distributed actuator coordination algorithms using a middleware framework.*

## 1. Introduction

For decades, sensors and actuators have been used in implementing various forms of automated control systems (e.g., climate control or industrial plant control). Recent technology advances (and associated cost reductions), however, have enabled sensors and actuators to become much more connected and pervasive. These trends are further accelerated by factors such as the increasing pervasiveness of Internet access and new wireless networking standards. These factors have enabled sensors and actuators to communicate with each other with growing ease within ever growing networked infrastructures forming sensor and actuator networks, or *SANETs*. Depending on the application, other networked entities such as databases, web services, and messaging systems may also be included as sensors or actuators. When combined with large-scale networked systems, the culmination of these advancements define a new generation of highly pervasive, dynamic, and heterogeneous Internet-scale control systems, which we refer to as *SANET control systems*.

Along with the proliferation of *SANET* control systems comes a wealth of new research challenges. One significant challenge, and the focus of this paper, is *distributed actuator coordination*, which involves managing the behavior of actuators that are related via overlapping *treatment domains*. Treatment domains are defined by the spatial region over which an actuator affects an environment. Focusing on the details of actuator coordination in emerging *SANET* control systems is important for several reasons. First, overlapping treatment domains create both

the potential for actuator interference and opportunity for cooperative shared task execution. While redundancy among sensors is typically not considered a negative feature, overlapping treatment domains could affect a *SANET* control system's ability to satisfy local and global application goals. Second, the relationship between actuators and their corresponding treatment domains may be dynamic. This may depend on factors such as the actuator(s) chosen to perform a specific task, inconsistent actuator performance, or the nature of the medium to be controlled. To compound these two challenges, *SANET* control systems will have a magnitude that far exceeds the scale of control systems encountered in prior generations. The number of distributed, transient, and potentially heterogeneous actuators and sensors that will populate *SANET* control systems will require novel programming abstractions that were previously unnecessary.

Given the motivations above, we propose to address distributed actuator coordination via the Java-based *Sentire* framework [2], which supports the composition of extensible middleware for *SANET* control systems in a high-level, platform-independent manner. This paper presents the next step in the evolution of *Sentire* and introduces its extensions for supporting the implementation of algorithms for distributed actuator coordination. We concentrate our initial research in defining suitable actuator programming abstractions and a service for facilitating coordination among actuators with interfering treatment goals. To the best of our knowledge, there is no previous proposal to address such a coordination scenario using a high-level middleware approach. This paper proceeds as follows. Section 2 describes our research scope and pertinent challenges. Section 3 describes *Sentire*'s initial approach to supporting actuator coordination. Section 4 illustrates this approach's utility in building an example of a *SANET* control system. Section 5 concludes with a discussion of related works and future research endeavors.

## 2. Research scope and challenges

Our research focuses on providing a middleware framework, called *Sentire*, for facilitating distributed actuator coordination. We target *Sentire* towards supporting

the composition of large-scale SANET control systems composed of distributed autonomous sub-controllers. Each sub-controller is responsible for controlling a prescribed treatment domain by issuing commands to one or more attached SANETs. Sub-controllers may also operate either singularly or collectively to enforce either local or global application goals. A significant challenge for such systems is managing the interaction (or, coordination) between various sub-controllers since their treatment domains may intentionally or unintentionally overlap given their individual actuators' treatment domains and the behavior of the environmental processes to be controlled. Overlaps could be beneficial, especially in helping to complete shared tasks in a resource-balanced manner. However, overlaps could also potentially degrade an application's effectiveness if the related sub-controllers are attempting to achieve interfering goals.

The validity of the above descriptions and concerns is illustrated by the following descriptions of two examples of next-generation SANET control systems: *vehicle traffic management* and *distributed energy management*. Multiple use cases help to identify salient research challenges and potential middleware-centric solutions for distributed actuator coordination.

**Vehicle traffic management.** Advanced vehicle navigation systems provide drivers with routes based on both their destinations and the collection of live traffic data (e.g., reports of congestion and accidents) using roadside sensors and other data sources [10][11]. We envision next-generation traffic systems to evolve with two characteristics. First, we expect the development of *decentralized* traffic management systems as a solution to administrative domains of control (e.g., the separate control of adjacent municipalities), in support of reduced and balanced network traffic load, and due to physical network partitioning. Furthermore, distributed control will avoid single points of failure in the face of large metropolitan areas with growing traffic systems. Second, we expect the implementation of globally optimal routing algorithms that reduce aggregate trip delay for a population of users rather than individualistic, greedy algorithms. If congestion is encountered on a roadway, such systems should re-route traffic so as to reduce the probability of creating *new* sources of congestion elsewhere in the system.

The composition of both decentralized and globally optimal routing algorithms require a great deal of coordination. When separate jurisdictions compose the overall system, they must coordinate to prevent their local routing decisions from interfering with the decisions of other domains. For instance, routing decisions for suburban New Jersey, Connecticut, and New York could affect traffic in New York City.

**Distributed energy management.** It has been argued that the current centralized structure of the national power grid is inadequate to reliably sustain the increasing de-

mand for energy. A proposed solution is the microgrid paradigm [5]. A microgrid is a semiautonomous system in which a group of power supplying entities (e.g., micro-turbines or fuel cells) is managed to intelligently distribute the larger grid's energy to a subset of end-users (e.g., those within office buildings and residential districts). The microgrid can operate in two ways. First, it can operate interconnected with the larger grid and economically purchase energy (e.g., electricity or natural gas) from it so as to efficiently meet users' demands. Second, it can disconnect from the larger grid, such as in the case of a blackout, and intelligently distribute stored and locally generated energy to users based on their usage behavior and priority of demands (possibly measured in terms of consumers' willingness to pay premium prices). We envision separate disconnected microgrids coordinating purchases of energy from each other to help meet users' demands distributed across multiple microgrid domains. In general, embedded sensors can be used to monitor and predict energy usage behavior and support energy purchasing and distribution, i.e., collectively make *actuation* decisions. Actuation decisions can also involve selectively controlling the operating state of appliances to control energy usage.

The previous usage scenarios call attention to three research challenges that should be addressed by a middleware framework, such as Sentire, for supporting the systematic implementation of algorithms for distributed actuation coordination:

1. *Actuator programming abstraction:* Designing a programmable abstraction and other middleware support for selecting appropriate underlying actuators for a task and controlling their actions in an abstract, platform-independent, and scalable manner is the most fundamental challenge.
2. *Actuator coordination service:* An extensible middleware service must be designed to enable the implementation of customized algorithms for coordinating actuation decisions among related sub-controllers in order to reduce actuator interference and support the enforcement of local or global application goals.
3. *Actuator treatment domain tracking:* Middleware services must be designed to track actuators' treatment domains. These can change over time in potentially unpredictable ways and are important to track since they determine sub-controllers' treatment domains and influence actuator coordination.

Our initial research largely focuses on item 1 and (with a specific focus on actuator interference) item 2. Research regarding item 3 is left for future study.

### 3. Sentire approach to actuator coordination

We augment Sentire for supporting distributed actuator coordination by first proposing an extended architec-

ture of middleware components as illustrated in Figure 1. Within the Sentire middleware, two major components are shown: *virtual actuator* programming abstractions and the *actuator coordination service manager*. Note that other Sentire components described in [2] reside within the intermediary cloud surrounding the actuator coordination service manager; these other components will not be discussed in this paper. Next, we describe further how we envision the proposed components supporting distributed actuator coordination algorithms.

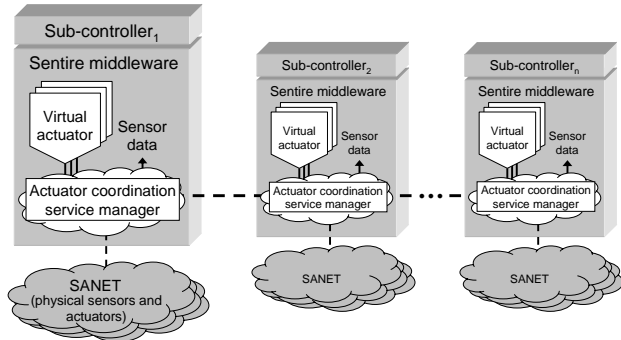


Figure 1. Sentire actuator coordination architecture

### 3.1. Virtual actuator

The *virtual actuator* is an abstraction for receiving programmer-level commands and passing them to individual (or, groups of) underlying actuators residing within the same sub-controller’s domain<sup>1</sup>. Supporting the functionality of virtual actuators consists of two fundamental tasks: *resource discovery* and *command translation*.

*Resource discovery* matches and logically binds a virtual actuator with an appropriate underlying actuator associated with the Sentire middleware. Two attributes of virtual actuators help Sentire determine which underlying actuator it binds to: *action type* and *treatment domain*. *Action type* describes the action that the underlying actuator should be capable of performing (e.g., adjusting temperature or messaging PDAs). *Treatment domain* describes the environmental region(s) that the underlying actuator should be able to affect. Because treatment domains can change over time, the value of this attribute is dynamically updated. In the simplest case, one underlying actuator, fitting the virtual actuator’s criteria, will be bound to the virtual actuator, creating a *one-to-one* (1:1) binding. However, multiple candidates could be found, in which case the most appropriate one must be selected. To cope with such a case, in addition to the two attributes mentioned, Sentire will also maintain the *resource availability* and *expected treatment latency* of each underlying

actuator. *Resource availability* describes the amount of resources (e.g., energy or treatment supplies) that are allotted to an actuator. *Expected treatment latency* describes the mean time elapsed before a desired treatment takes effect. Expected treatment latency depends both on system factors (e.g., network latency) and physical characteristics of the actuator and the environment (e.g., how long it will take to heat up a room to the desired temperature, given the current outdoor temperature). The computation of expected treatment latency will depend on feedback from sensors, or even from end-users. These attributes can be used to select the most appropriate actuator to bind to a virtual actuator, based on either user preferences or system-defined policies. When a virtual actuator’s underlying actuator becomes unavailable (e.g., due to network or device faults), Sentire will also attempt to rebind the virtual actuator to a new comparable actuator.

If the virtual actuator’s treatment domain is larger than that of any available underlying actuator, Sentire will attempt to use multiple underlying actuators to collectively affect the treatment domain. This requires creating a *one-to-many* (1:n) binding. Even in this case, the previous descriptions regarding 1:1 bindings are still applicable.

*Command translation* defines how the commands that are received by virtual actuators are passed to underlying actuators. In the case of a 1:1 binding, Sentire will first approve the command based on the actuator’s resource availability before translating it and forwarding it to the bound actuator. However, in some cases, using 1:n bindings may be more efficient than using 1:1 bindings. This is because 1:n bindings enable abstracted actuator control at varying scopes, which is beneficial for large-scale systems for which it would be inefficient to control multiple underlying actuators individually. Unlike 1:1 bindings, command translation with 1:n bindings requires more sophisticated command *decomposition* operations. Therefore, in its current implementation, Sentire supports 1:n bindings by simply distributing replicated commands to individual actuators; more sophisticated implementations are left for future research.

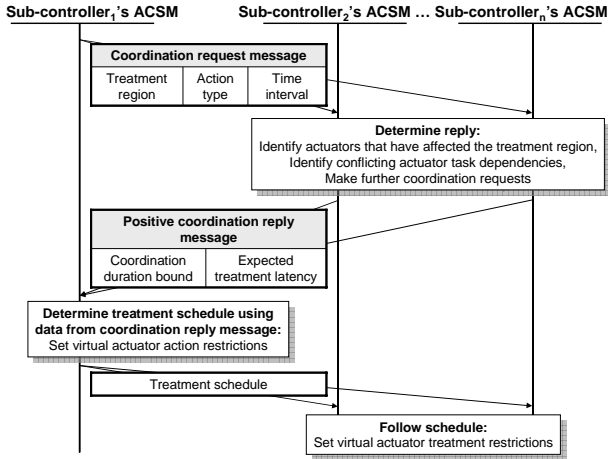
### 3.2. Actuator coordination service manager

The *actuator coordination service manager* (ACSM) provides a messaging service and other mechanisms to support distributed coordination among actuators associated with different sub-controllers. We currently focus on enabling the ACSM to support the implementation of algorithms that attempt to reduce actuator interference.

Preventing actuator interference involves three general steps. First, the ACSM determines if an actuator’s treatment is being hindered. Second, the source of interference (i.e., other actuator(s)) is identified. Third, the relevant parties negotiate how to prevent interference. Regarding

<sup>1</sup> Underlying actuators may be realized as hardware or software-based entities.

the first step, the ACSM interfaces with a virtual actuator (as shown in Figure 1) to determine the expected treatment latency and outcome for the currently desired treatment. Using sensor data corresponding to the virtual actuator's treatment domain, the ACSM determines if the expected treatment outcome is met within the latency constraints while also accounting for variation due to network delays, jitter, etc. If it is not, and assuming that the underlying actuator(s) has adequate resources to complete the treatment, the ACSM will assume that there is a source of interference and initiates a negotiation process.



**Figure 2. ACSM coordination scheme for preventing actuator interference**

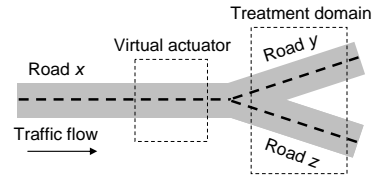
For this paper, we propose the coordination scheme illustrated in Figure 2. First, the ACSM will send a coordination request message to all neighboring sub-controllers describing the pertinent treatment domain, action type, and time interval during which the interference was believed to have occurred. Using the information in this message, the solicited sub-controllers' ACSMs will identify any of their actuators that have affected the pertinent treatment domain during the identified time interval. If any are found, the ACSM will check whether these actuators are committed to any other significant tasks (e.g., cooperating with other actuators for shared task execution). Note that the ability to coordinate may be beyond the sub-controller's control; this may occur in flow-based SANET applications where decisions regarding the flows of a resource are distributed among various sub-controllers (e.g., traffic management). In this case, the solicited sub-controller's ACSM will also attempt to coordinate with other sub-controllers that are indirectly interfering with the actions of the original solicitor. A programmer-defined policy uses the information gathered from these steps to determine if the ACSM originally solicited is able to coordinate. If so, the ACSM will reply with a coordination reply message including the duration of time it commits to coordinating and the expected treat-

ment latency of its actuator(s) that are involved in the interference. Once the original soliciting ACSM has received this information from all sub-controllers willing to participate, it determines a schedule under which all participating sub-controllers are allowed in time-divided slots to sequentially affect the pertinent treatment domain. A sub-controller's respective slot is determined by its expected treatment latency plus a prescribed constant allowing the desired treatment goal to be sustained for a period of time. While waiting for its respective slot, the sub-controller's relevant virtual actuator will automatically be restricted from affecting the pertinent treatment domain. The total length of this schedule is determined by the smallest duration of time indicated in the coordination reply messages.

This scheme represents an *initial* step in preventing distributed actuator interference; we make no claims that it is effective for all application scenarios. Therefore, strengthening this scheme and researching possible alternatives is a subject of future research.

#### 4. Example Sentire usage scenario

We describe a scenario that illustrates how Sentire's extensions can be utilized to implement a SANET control system. We focus on the vehicle traffic management system described in Section 2. This is only a conceptual example, as a rigorous solution for distributed traffic management is beyond the scope of this paper.



**Figure 3. Logical virtual actuator definition for vehicle traffic management system**

The general architecture would consist of a series of sub-controllers (belonging to different municipalities) interfaced with vehicle navigation messaging systems and roadside sensors, acting as actuators and sensors respectively. Sub-controllers control traffic according to road congestion, drivers' destinations, and the drivers' current locations. A virtual actuator is logically defined as a location corresponding to a road segment immediately preceding a major road division; the road segments resulting from this division characterizes the treatment domain (see Figure 3). The virtual actuators are defined in this way because it is at these *decision points* (the road divisions) where drivers' actions can significantly affect traffic. In reference to Figure 3, an example virtual actuator command would be to route 20% of all vehicles with destination  $b$  (not shown) onto road  $z$  because of congestion on

road  $y$ . The underlying messaging system would then transmit the corresponding instructions *only* to the vehicles in the virtual actuator location shown in Figure 3. Note that other traffic authorities' routing decisions could also affect congestion on road  $y$  or  $z$ , creating overlapping treatment domains. In this case, the process described in Section 3.2 can be used to enable the authorities to alternately route traffic in an attempt to prevent creating new sources of congestion (or interference) and hence reduce travel delay on a global application scale.

## 5. Concluding remarks

Our research is related to the field of control systems. Research in control systems (or, control theory) has established a wealth of knowledge for automatically facilitating control over some system state(s) [4]. Sentire complements this field by providing an intuitive model to incorporate knowledge from control theory to build and operate distributed control systems in an ad hoc manner with pervasive and heterogeneous devices/services. Sentire is also related to other research activities which have addressed high-level SANET middleware. The research activities described in [3], [7], and [12] propose middleware that focus primarily on providing facilities to adjust resource management, quality of service, and quality of sensed information regarding wireless sensor networks and related systems. Other activities go beyond sensor networks to also provide support for actuator management. For example, [8] describes a system in which a high-level data-stream-centric composition model is used to build SANET middleware. In [9], a high-level file system abstraction model is used for SANET application development. The research described in [1] uses sentient object and event interaction model to provide high-level SANET composition support. While the above research activities have made important contributions, we distinguish ourselves by addressing more sophisticated aspects of actuator programming abstractions and explicitly offering support distributed actuator coordination. To our knowledge, the only activity that addresses actuator coordination is described in [6]. However, this activity focuses on a specific algorithm for coordinating actuators for shared task execution while optimizing energy usage and task completion time.

In general, Sentire exemplifies an on-going research effort to systematically build middleware for SANET control systems. We have identified several directions for future research. First, we intend to research more sophisticated techniques for  $1:n$  actuator command decomposition.  $1:n$  sensor bindings have been proposed and supported mainly using data aggregation techniques. However, since actuators deal with command structures that fan out instead of coalesce, virtual actuator commands must be properly decomposed according to the connec-

tivity and capabilities of the underlying actuators. Second, while the ACSM's current actuator coordination scheme is somewhat customizable, we intend to research how to further extend the ACSM to support swappable actuator coordination algorithms. This mainly involves researching alternative actuator coordination schemes and identifying common primitive operations that the middleware should offer to support their implementation. Third, we intend to research autonomous techniques for tracking actuators' treatment domains. This process involves an analysis of actuator causal effects given environmental context and behavior and may be particularly challenging to track across different sub-controller's domains. Finally, other on-going work involves intelligently determining actuator treatment latency and designing a substantial simulation test-bed for further evaluation purposes.

## 6. References

- [1] G. Biegel and V. Cahill, "A framework for developing mobile context-aware applications," in *Proc. of the 2<sup>nd</sup> IEEE PERCOM*, pp. 361-365, March 2004.
- [2] J. W. Branch, J. S. Davis, D. M. Sow, and C. Bisdikian, "Sentire: a framework for building middleware for sensor and actuator networks," in *Proc. of the 3<sup>rd</sup> IEEE PERCOM Workshops*, pp. 396-400, March 2005.
- [3] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," in *IEEE Network*, vol. 18, no. 1, pp. 6-14, Jan./Feb. 2004.
- [4] B. C. Kuo and F. Golnaraghi, *Automatic Control Systems*, John Wiley & Sons, Inc., Hoboken, NJ, 2003.
- [5] C. Marnay and O. C. Bailey, "The CERTS microgrid and the future of the macrogrid," in *Proc. of the ACEEE 2004 Summer Study on Energy Efficiency in Buildings*, Aug. 2004.
- [6] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz, "A distributed coordination framework for wireless sensor and actor networks," in *Proc. of the 6<sup>th</sup> ACM MobiHoc*, pp. 99-110, May 2005.
- [7] M. Modahl, I. Bagrak, M. Wolnetz, P. Hutto, and U. Ramachandran, "MediaBroker: an architecture for pervasive computing," in *Proc. of the 2<sup>nd</sup> IEEE PERCOM*, pp. 253-262, March 2004.
- [8] L. St. Ville and P. Dickman, "Garnet: a middleware architecture for distributing data streams originating in wireless sensor networks," in *Proc. of the 23<sup>rd</sup> IEEE ICDCS Workshops*, pp. 235-240, May 2003.
- [9] S. Tilak, B. Pisupati, K. Chiu, G. Brown, and N. Abu-Ghazaleh, "A file system abstraction for sense and respond systems," in *Proc. of the 2005 ACM MobiSys EESR Workshop*, pp. 1-6, June 2005.
- [10] Traffic.com, <http://www.traffic.com>.
- [11] XM NavTraffic, <http://www.xmradio.com/xmnavtraffic>.
- [12] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," in *IEEE Network*, vol. 18, no. 1, pp. 15-21, Jan./Feb. 2004.