Flow Semantics for Intellectual Control in Smart City IoT Systems

Richard Linger
Chief Technology Officer
Lenvio Inc.
Manassas, VA
richard.linger@lenvio-inc.com

Alan Hevner
Muma College of Business
University of South Florida
Tampa, FL
ahevner@usf.edu

Abstract: Internet of Things (IoT) systems for Smart Cities will be comprised of massive numbers of hardware, software, and network components, all sensing, computing, communicating, and controlling in dynamic architectures and state spaces of extraordinary complexity. Nodes, connections, and users will come and go, quality and reliability will be unpredictable, and failures and intrusions will be ever-present. It is vital that these systems be rigorously designed, developed, and governed; but how can a rigorous engineering discipline be defined for an environment where the capability, connectivity, and integrity of components will vary from moment to moment? Current IoT reference architectures specify system structures, but provide little guidance for the semantic foundations required to create and verify system functionality, security, and quality attributes. In this paper, we focus on Flow Semantics as a rigorous engineering foundation for analysis, development, evolution, operation, and governance of IoT systems in such unpredictable environments. Flow Semantics are overarching engineering artifacts that exhibit desirable properties for IoT system development and operation. They are based on scale-free mathematical foundations of functions and relations, and capitalize on recent advances in automated computation of software behavior. The goal of this research is to develop engineering principles and practices for maintaining intellectual control in Smart City IoT systems.

Keywords: Internet of Things, Smart Cities, Reference Architecture, Governance, Intellectual Control, Flow Semantics, Quality Semantics, Engineering Foundations, Software Behavior Computation

IoT System Realities

Internet of Things (IoT) systems exhibit unprecedented levels of scale and complexity (Atzori et al. 2010). These systems are characterized by very-large-scale heterogeneous networks with often unknown and unknowable boundaries and components. Nodes, connections, configurations, and capabilities come and go in unpredictable ways, and failures and compromises are ever-present. Dynamic interconnectivity of systems-of-systems limits visibility and intellectual control of security and functionality. User transaction flows traverse systems and components of varying quality and reliability. Additional complications arise from the variety of architectures, platforms, languages, protocols, organizations, and users that will be involved. Yet IoT systems must provide unprecedented levels of reliability and dependability for effective Smart City operations (Gil-Castineira et al. 2011; Vlacheas et al. 2013). A critical question is how a rigorous engineering discipline can be defined for designing, developing, evolving, and operating such massive, complex, and unpredictable systems.

The burden of un-mastered complexity leads to loss of intellectual control when it exceeds human capabilities for reasoning and analysis. Intellectual control means understanding IoT system behavior at all levels in all circumstances of use. It does not mean the absence of uncertainty – that will always be with us – but rather the capabilities, through engineering and management processes, to deal with it.

The essential role of governance of IoT systems addresses resource facilitation along with organizational authorities, responsibilities, and accountabilities (Weber 2013). IoT governance requires a clear and detailed reference architecture as a basis for analyzing, designing, regulating and communicating IoT activities. Current IoT reference architecture proposals, such as the EU's Architectural Reference Model (ARM 2013) and Cisco's Internet of Things Reference Model (Cisco 2014), provide extensive detail on

structural requirements for IoT systems, but say little with respect to requirements for behavioral semantics of IoT system applications. This is a gap that we attempt to understand and address in this research stream. Our motivation is simple: without rigorous behavioral semantics as a foundation for IoT applications such as Smart Cities, attempts to develop and deploy these systems may pose unacceptable risks. To build our argument, we begin by identifying the semantic deficiencies in current IoT reference architectures. Then we propose a set of mathematical bases for flow semantics to support an engineering foundation for IoT governance and application development.

IoT Reference Architectures

An IoT reference architecture provides a common understanding and vocabulary for interoperability and communication across various platforms in an IoT system. Given a standard reference architecture, businesses and developers can create compliant IoT solutions for specific application ecosystems, such as Smart Cities. The complexities and compromises in developing an acceptable IoT reference architecture are enormous, and we are just beginning to see initial proposals from standards organizations (e.g. ARM 2013) and industry (e.g. Cisco 2014). We observe that these initial reference architectures have a primary focus on IoT structures, including:

- Components – Sensors, Computers, Data Repositories, Servers, Platforms, etc.
- Connectors – Networks, Pipes, Telecommunications, etc.
- Configurations – Patterns for components and connectors in well-defined arrangements.
- Protocols – Detailed support for data flows and control flows within and between configurations.

While such architectural details are essential for providing and enforcing the syntactical structures of IoT systems, there is little attention to the behavioral semantics needed to support engineering of IoT applications. A recent survey of Smart Cities software architectures finds a great disparity in the range of requirements considered important for inclusion in the various architecture descriptions (da Silva et al. 2013). They conclude that no current Smart City architecture fulfills all essential requirements for application development. Further, they identify an overriding focus on technical issues with a significant gap on human-centered (i.e. social and behavioral) issues of the Smart City applications.

Computational Semantics for IoT Systems

IoT application development requires stable and dependable anchors for specification, design, and verification in a unified engineering discipline. In addressing this need, we build on our previous research stream on Flow-Service-Quality (FSQ) technologies, which we adapt here to IoT Smart City applications (Hevner et al. 2002; Linger et al. 2002; Hevner et al. 2009). We propose three overarching semantic concepts that contribute to foundations for IoT development. They prescribe both engineering processes and new forms of computational automation to support them.

Flow Semantics

IoT transactions are composed of flows of control and data through IoT architectural structures (i.e. components, connectors, and configurations). Methods for creation, instantiation, execution, monitoring, completion, and deletion of IoT transactions are required for any application domain. User task flows and their refinements into invocations of other flows and system services can provide a unified engineering foundation for analysis, specification, design, and verification of required functionality and quality attributes. Engineering development of task flows can be augmented by computational automation for behavioral analysis of flows and their components.

Flow Semantics are compositions of components that carry out user tasks to accomplish enterprise missions. Flows are composed of control and data movement among nodes and communication links in IoT systems, and may employ other flows in carrying out user tasks. They define and implement business value for IoT users and sponsors, and can represent the overarching engineering and operational artifacts of IoT systems. Flows employ special semantic structures that preserve deterministic properties for human understanding and machine analysis, despite the underlying asynchronous and unpredictable behavior of IoT systems, as described below.

Quality Semantics

Quality attributes can be associated with flows and the system services they invoke, and are specified as dynamic mathematical properties to be computed, rather than as a priori predictions of limited value in real-time operations. Quality Semantics can be dynamically managed in operational use to provide required levels of availability, quality, and functionality (Walton et al. 2009).

Three semantic quality objectives can be identified. (1) A flow transaction will require minimum levels of quality to be successfully performed. How do we define the quality requirements of a designed user transaction on an IoT architecture? (2) As a transaction is instantiated on a particular IoT system at a given point in time, how can we predict the quality levels that are available? This will depend on the current state of the IoT system in terms of capacity, load, reliability, and many other system state variables. If the required levels of quality cannot be currently provided, a decision must be made whether to perform the transaction or wait until a sufficient level of quality can be obtained in the system. (3) What management mechanisms are needed in an IoT system to monitor the progress of an executing transaction to ensure that the quality levels are achieved? If certain qualities are falling short, dynamic flow management may be required to alter the flow path or abort the transaction and reinitiate it at a later time. In essence, quality attribute status must be known and dynamically managed in IoT systems for all user flows.

Evolution Semantics

Flow Semantics and Quality Semantics prescribe requirements for IoT architecture semantics focused on management and governance of user functionality and data in a bounded application domain. In complex IoT environments, however, it is not possible to completely predict all possible system behaviors. New and unpredictable behaviors will emerge over time and must be recognized and managed. Also, an IoT architecture must evolve and adapt over time based on changes in goals, ecosystems, and technologies. Evolution Semantics will prescribe processes for managing and directing this evolution while maintaining current operational capabilities.

In the remainder of this paper, we elaborate on application of Flow Semantics to Smart City IoT systems. Issues of Quality Semantics and Evolution Semantics will be topics for future research.

Foundations for IoT Flow Semantics

The IoT Uncertainty Factors

Given the need for an engineering discipline that can explicitly accommodate development for the operational variability of IoT systems, it is useful to focus on sources of this variability in what are termed Uncertainty Factors. These factors capture the unpredictable outcomes of system service uses. For example, a service may be unavailable, unreliable, incorrect, partial, or compromised, to name a few areas of uncertainty. In addition, services will typically be operating asynchronously, thereby creating an additional level of engineering complexity.

Uncertainty Factors are an important concept that helps guide definition of the semantic properties that IoT components must possess to permit effective operational control. An engineering discipline for IoT systems must provide systematic means for developers to deal effectively with the Uncertainty Factors in a particular domain in addition to provision of required functionality, quality, and performance.

The Idea of Flow Semantics

A flow is a traversal of IoT network components to perform a specific task for a user (person or machine) or another flow. A flow combines functional capabilities of components such sensors, controllers, computations, protocols, communications, and services, any of which can be operating asynchronously, in order to complete its specified task. Flows can range from large and complex to small and simple. Flows can invoke other flows, and can be grouped into flowsets that implement particular mission objectives.

We identify two overarching engineering requirements for flows and their constituent components:

- Flows must be deterministic to permit design and verification under intellectual control despite the underlying complexity and asynchronism of service uses.
- Flows must provide suitable responses despite the Uncertainty Factors (i.e. robustness) to permit dependable operational use.

In informal terms, flows and their components must be self-contained in a semantic sense to permit localized development and verification under intellectual control, yet still embody all uses of external flows and services required to complete their specified tasks, but whose responses cannot be predicted.

In addition, because of the Uncertainty Factors, an IoT system can serve up absolutely anything to a flow when it invokes external flows and services. To achieve a self-contained and localized view of a flow for development while accommodating necessary use of external capabilities, it is necessary that a flow "doesn't care" what the system serves up, but will always employ what is served up to carry out its task. If what is served up is complete and correct, the flow satisfies its nominal specification. If what is served up is incorrect or incomplete, the flow does not fail, but rather communicates its status for other flows to take action, and satisfies its specification in this way. That is, both ordinary and extraordinary behaviors, and everything in between, are defined in its specification. This is a critical property for flows and their components – it localizes reasoning for development yet permits use of external functionality for task completion. This property is a cornerstone of Flow Semantics engineering. It prescribes a special semantic model for development and verification that can be standardized across all flows in an IoT system, as described here.

The Mathematics of Flow Semantics

The mathematical foundations of Flow Semantics are defined to support development and verification of flows for uncertain environments as a standard engineering practice. To accommodate the unpredictable behavior of other flows and services, flow specifications permit only definition of the processing that the flow itself performs, and not the processing of the services it invokes. To achieve this, Flow Semantics require definition of appropriate actions by a flow for all possible responses from invoked services, both desired and undesired. Thus, if the behaviors of invoked services change for any reason, the specification and verification of the invoking flow need not change.

This approach offers important advantages. It requires for mission completion that the Uncertainty Factors be explicitly dealt with in flow design. It permits reasoning about flows to be localized yet complete. And it permits flows to be defined by simple deterministic structures despite the underlying asynchronous behavior of their constituent services. These deterministic structures can be refined, abstracted, and verified using straightforward compositional methods for human understanding and intellectual control.

These IoT objectives require extension of the traditional functional semantics model of programming. That model is based on denotational semantics (Stoy 1977) and the well-known concept of programs as rules for mathematical functions or relations, that is, mappings form domains to ranges, or stimuli to responses, as specified by transition functions (Linger et al. 1979).

Briefly, the principal extension required to ensure that flows are deterministic and deal with the Uncertainty Factors is to make the histories of service invocations themselves part of the specified behavior of their using flows. This is achieved by including the invocation stimulus history (ISH) of every service in the range (responses) of the function that represents the specification of a flow. Because subsequent flow processing can depend on the responses from these invocations, the invocation response history (IRH) must be part of the domain (stimuli) of the mathematical function that represents the specification of a flow. This extension is labeled Response-Based Semantics (RBS), as illustrated in Figure 1.

The transition function is thus f: (S x IRH) → (R x ISH). This counterintuitive inclusion of service responses in the domain of a flow and service stimuli in the range allows flows to deal with the Uncertainty Factors. IRH represents the range of possible service responses, and thus embodies the Uncertainty Factor possibilities that must be accommodated in flow design. This requires designing for all possible responses that external flow and service invocations can produce, including missing, incorrect, or partial responses, to name a few. Additional information on this semantic model can be found in (Hevner et al. 2009). The model is characterized by theorems that inform engineering practices in its application, including the flow

structure theorem, abstraction/refinement theorem, flow implementation theorem, and flow verification theorem.

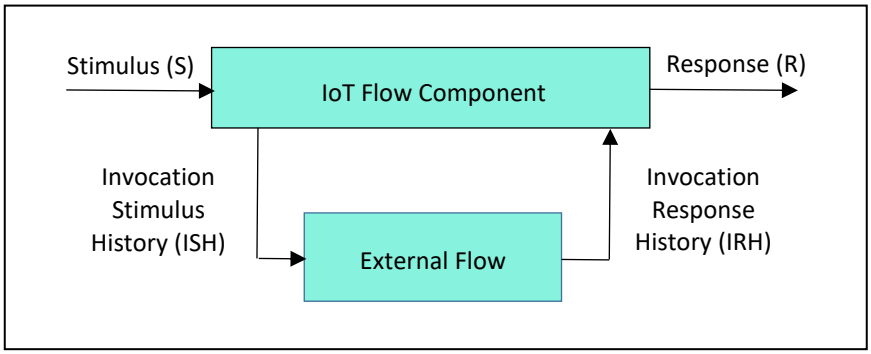

Figure 1. Response-Based Semantics

This approach to specification is important for maintaining intellectual control. Deterministic flows that invoke non-deterministic services can be specified by deterministic mathematical functions, making human reasoning and analysis much simpler. These methods differ from conventional functional semantics that require that a specification of a flow must include the full functional effects of all external flow and service invocations. The key point is that Response-Based Semantics localize the design and verification of a given flow, independent of other flows and service uses, and permit full accommodation of the Uncertainty Factors. The ability to superimpose a deterministic flow on an asynchronous IoT network and effectively deal with the Uncertainty Factors facilitates human reasoning and control.

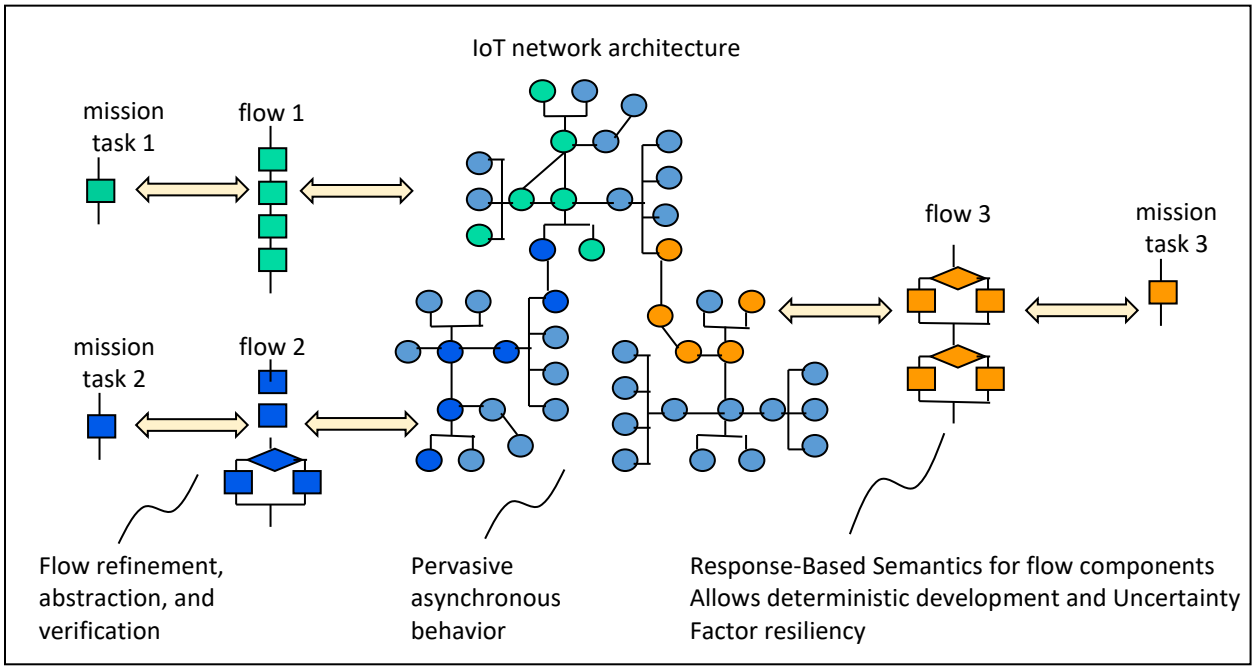

Figure 2. IoT Flow Illustration

IoT Flow Engineering

Flows are defined by specifications based on mission objectives. Specifications are refined into function-equivalent, architecture-independent user task flows of components and data, then into compositions of function-equivalent, architecture-dependent components and data in a stepwise process. At each step, designs are expressed as compositions of single-entry, single-exit control structures, including sequence, alternation, and iteration, and their variants. Local sub-specifications are defined and documented with

each refinement for use in verification. Components can invoke other flows and system services through stimulus/response interactions in a hierarchical structure as defined by Response-Based Semantics. Flow designs are implemented within IoT architectures. The union of flows for a given application can suggest a sufficient, but likely inefficient, IoT architecture. Figure 2 above illustrates flow development and operations in an IoT environment. Based on this basic description of flow engineering, the following concepts can be addressed in IoT applications.

Flow Abstraction. Existing flows can be abstracted for analysis and evolution through a process of stepwise abstraction that reverses the flow development process.

Flow Engineering for Uncertainty Factors. Uncertainty Factor engineering requires that flows conform to Response-Based Semantics in correctly processing all possible responses (IRH) from invoked flows and services they employ. This aspect of the design process deals with risk management and mission survivability, as well as management and governance of IoT operations. Definition of standards for Uncertainty Factor detection and responses can simplify and organize this aspect of design.

Flow Verification. Because flows are deterministic, traditional function-theoretic verification can be employed, as defined in the following equations that enumerate equivalences between functional specifications and their procedural refinements. (Linger, et al. 1979) The functional forms represent the behavior signatures of the control structures. They can be obtained through function composition and case analysis as described below (for control structure labeled P, operations on data labeled g and h, predicate labeled p, and program function (specification) labeled f). These function equations are independent of language syntax and program subject-matter, and define the mathematical starting point for automated behavior computation. The behavior signature of a sequence control structure

P: g; h

can be given by

f = [P] = [g; h] = [h] o [g]

where the square brackets denote the behavior signature of the enclosed program and "o" denotes the composition operator. That is, the program function of a sequence can be calculated by ordinary function composition of its constituent parts. The behavior signature of an alternation control structure

P: if p then g else h endif

can be given by

f = [P] = [if p then g else h endif]
= ([p] = true → [g] | [p] = false → [h])

where | is the "or" symbol. That is, the program function of an alternation is given by a case analysis of the true and false branches. The behavior signature of an iteration control structure

P: while p do g enddo

can be expressed using function composition and case analysis in a recursive equation based on the equivalence of an iteration control structure and an iteration-free control structure (an if then structure):

f = [P] = [while p do g enddo]
= [if p then g; while p do g enddo endif]
= [if p then g; f endif]

Flow designs incorporating Response-Based Semantics will include statements to invoke external flows and check responses. These statements are specified by behavior functions that are extended to include lower-level histories. In particular, a statement to "invoke flow f1 with stimulus s1" will require a behavior function that appends S1 to ISH. A statement to "receive response R1 from flow F1" will require a behavior function that sets local variable R1 to the corresponding element of IRH. The correctness relations described here can be verified in team reviews, and also form the basis for automated analysis, as described below.

Computing IoT Flow Behavior

Recent advances in automated computation of software behavior (Pleszkoch et al. 2012, Prowell et al. 2012) open opportunities for verification of IoT flow component functionality and security properties. Behavior computation is based on the function-theoretic equations defined above. It operates at the level of deep

functional semantics, not surface syntax, and is implemented through compositional methods that abstract program functionality into equivalent specifications. These computed specifications can in turn be analyzed for the presence of any malicious content, as well to validate required legitimate operations. The computed behavior of a program can be expressed as a set of disjoint cases, each consisting of a predicate that triggers it and a definition of how the case uses and changes the state of a system. Initial application of this technology has been directed to detection and analysis of malware at the binary level in both cybersecurity and software assurance contexts.

Behavior computation does not look for things in code or in executions, both of which are subject to fundamental limitations. Rather, it computes the net effects of programs and analyzes those results for behaviors of interest, whether legitimate or malicious. In the context of malware detection, it is easy to hide malicious content at the syntactic level, but very difficult to hide at the semantic level.

Behavior computation technology can be applied to the architectures and languages employed in IoT systems. It can support development and verification of component software, as well as integration and composition of component behaviors in flows. In turn, governance and management of deployed IoT systems can benefit from periodic behavior computation for operational flows to validate current functionality and absence of malicious content. It is increasingly clear that IoT development, with its massive numbers of components and interactions, will not be business as usual and traditional methods and tools for system development may prove insufficient. Next-generation forms of computational automation such as behavior computation will be required to leverage human capabilities for IoT system engineering. In essence, development processes and automation must evolve to keep pace with the scale and complexity of IoT systems for Smart City operations.

A Smart City Illustration

In order to briefly illustrate a few of these concepts, imagine a Smart City IoT system and the Smart Grid (SG) application within it (Zanella et al. 2014). The SG itself is a complex system-of-systems, supporting sensing, communication, analysis, and actuation within a mission-centric management and governance structure. Thousands of software-defined components and connections are involved, from smart meters to power stations, all heavily instrumented (the sensor aspect of IoT) and actuated (the control aspect of IoT), participating in real-time, asynchronous network-spanning flows. Massive amounts of data are created, aggregated, analyzed, and acted upon in closed-loop control patterns by humans and machines, often through computation-intensive big data analytics.

In organizing this functionality, it is useful to group related flows that support particular operations into what are termed *flowsets*. Figure 3 depicts an imagined high-level flowset grouping for specification and refinement in an iterative design process, based on the NIST Smart Grid Interoperability Framework (Fitzpatrick and Wollman 2010). Each box represents a flowset addressed to a particular mission function, with example high-level flows, actually hierarchical flowsets in their own right, listed as illustrations of functionality. Each flowset is in fact a complex system-of-systems of its own.

These flows invoke each other in real-time, protocol-based inter-communications across large-scale networked systems, and invoke *reach-back*[1] and supply chain flowsets at lower levels in virtually endless interactions with economic and societal entities. At first thought, the complexity of such IoT systems may seem overwhelming, however, deterministic flows implementing Response-Based Semantics and design for Uncertainty Factors can help maintain intellectual control by localizing reasoning and isolating yet providing required functionality.

As described above, flows can depend on other flows distributed across an IoT network in order to achieve successful completion. Transitivity analysis can reveal such dependencies for analysis of survivability, resiliency, and other quality attributes. Given a primary flow, the first step in transitivity analysis is to determine what other flows must be invoked to complete its required functionality. In addition, every flow can exhibit both desired and undesired outcomes as defined, say, by equivalence classes that cover all possible responses. It is also possible that primary flows can depend on successful completion of flows not directly invoked, but rather temporally separated while nevertheless producing required data or system

[1] Reach-back is defined as the process of obtaining products, services, applications, forces, equipment, or material from organizations that are not forward deployed.

states to permit primary flows to proceed. Basically, the Smart City flowsets of Figure 3 are all interdependent at some level.

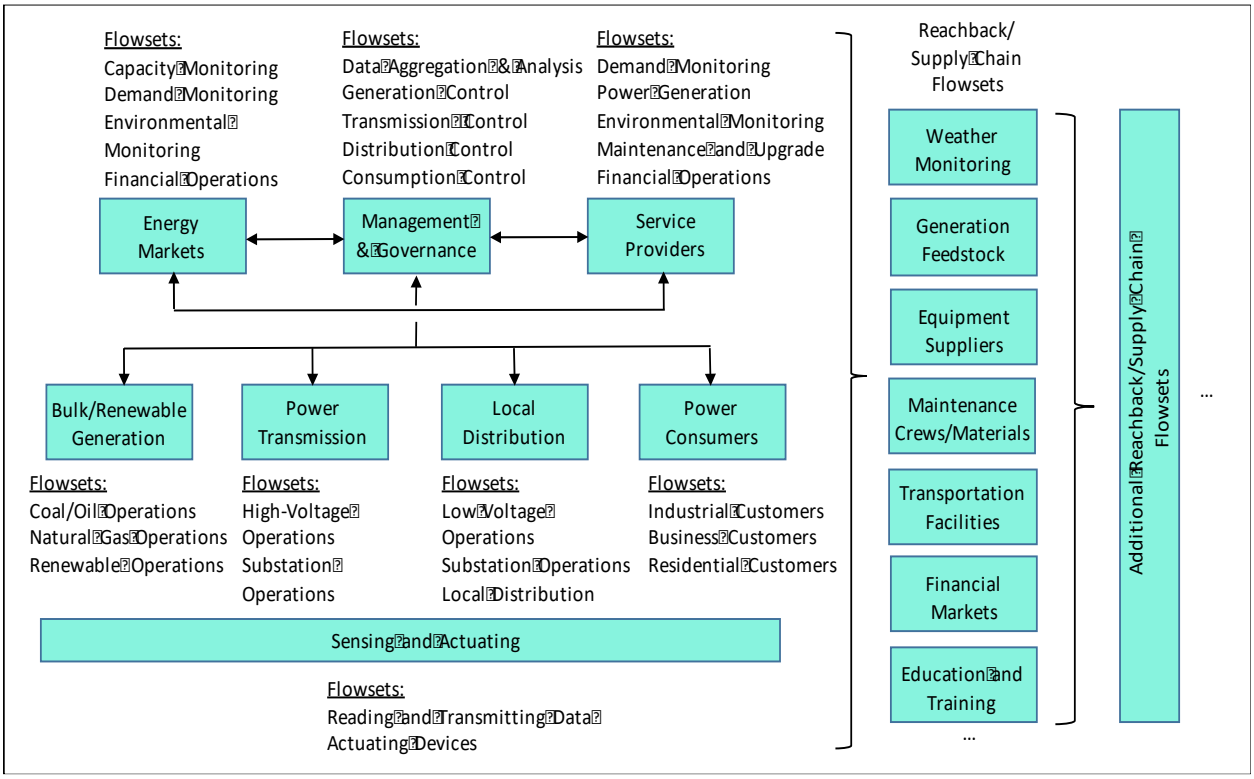

Figure 3. IoT Smart Grid Flowset Illustration

Discussion and Future Research

Research in Flow Semantics is complementary to IoT architecture research, which currently focuses primarily on structural properties. While structural aspects are important for IoT development, the semantics of IoT operations, which are software-defined at virtually every level, are essential to developing and governing these systems. We believe Flow Semantics can play an important role in a unified engineering discipline for IoT systems, with substantial impact on their specification, development, verification, operation, and survivability. The following observations summarize key aspects of Flow Semantics:

- Flows help organize and aggregate IoT functionality around mission objectives at all levels, from systems down to sensors and controllers.
- Flows can be defined to be deterministic for localized reasoning and verification while nevertheless embodying dependencies on other flows in a system.
- Flows can provide deterministic operations despite the underlying asynchronism of IoT systems.
- Flows can be designed to deal with Uncertainty Factors through implementation of Response-Based Semantics for operational resiliency and survivability.
- Flows are scale-free, with the same principles and processes involved in high-level and low-level flow development and operation.
- Flows are implementation-independent, and can serve as first-class concepts for IoT system specification and development.

Future research on Flow Semantics will include case studies dealing with real-world problems in IoT system development. Research in Flow Semantics automation will help drive adoption of this technology. Of particular importance in this regard is extension of the science and technology of automated software behavior computation (e.g. Linger et al. 2007) for development and verification of flowsets across IoT

architectures and languages. Further, we will extend the reach of IoT semantics to Quality Semantics and Evolution Semantics and apply these new ideas to Smart City IoT applications.

Acknowledgements

It is a pleasure to acknowledge the contributions of Mark Pleszkoch and Matt Mullarkey to the ideas in this paper.

References

ARM. 2013. Introduction to the Architectural Reference Model for the Internet of Things, downloaded from http://www.iot-a.eu/arm .

Atzori, L., Iera, A., and Morabito, G. 2010. "The Internet of Things: A Survey," *Computer Networks* (54), pp. 2787-2805.

Cisco. 2014. "The Internet of Things Reference Model," Cisco White Paper.

Da Silva, W., Tomas, G., Dias, K., Alvaro, A. Afonso, R., and Garcia, V. 2013. "Smart Cities Software Architectures: A Survey," *Proceedings of the 28th ACM Symposium on Applied Computing*, Portugal.

Fitzpatrick, G.J. and Wollman, D.A. 2010. NIST Interoperability Framework and Action Plans. *IEEE Power and Energy Society General Meeting*, Minneapolis.

Gil-Castineira, F., Costa-Montenegro, E., Gonzalez-Castano, F., Lopez-Bravo, C., Ojala, T., and Bose, R. 2011. "Experiences inside the Ubiquitous Oulu Smart City," *IEEE Computer* (44:6), pp. 48-55.

Hevner, A., Linger, R., Sobel, A., and Walton, G. 2002. "The Flow-Service-Quality Framework: Unified Engineering for Large-Scale, Adaptive Systems," *Proceedings of the 35th Annual Hawaii International Conference on System Science* (HICSS35), Hawaii.

Hevner, A., Linger, R., Pleszkoch, M., Prowell, S., and Walton, G. 2009. "Flow-Service-Quality (FSQ) Engineering: A Discipline for Developing Systems of Systems," Chapter 2 in *Systems Analysis and Design: Techniques, Methodologies, Approaches, and Architectures*, Edited by R. Chiang, K. Siau, and B. Hardgrave, Advances in Management Information Systems Monograph Series, M.E. Sharpe, Inc.

Linger, R., Mills, H., and Witt, B. 1979. *Structured Programming: Theory and Practice*. Reading, MA: Addison Wesley.

Linger, R., Pleszkoch, M., Walton, G., and Hevner, A. 2002. "Flow-Service-Quality (FSQ) Engineering: Foundations for Network System Analysis and Development," Technical Note CMU/SEI-2002-TN-019, Software Engineering Institute, Carnegie Mellon University.

Linger, R., Pleszkoch, M., Burns, L., Hevner, A., and Walton, G. 2007. "Next-Generation Software Engineering: Function Extraction for Computation of Software Behavior," *Proceedings of the 40th Annual Hawaii International Conference on System Sciences* (HICSS40), Hawaii.

Pleszkoch, M., Linger, R., Prowell, S., Sayre, K., and Burns, L. 2012. "Automated Behavior Computation for Software Analysis and Validation," *Proceedings of Hawaii International Conference on System Sciences (HICSS-45)*.

Prowell, S., Pleszkoch, M., Sayre, K., and Linger, R. 2012. "Automated Vulnerability Detection for Compiled Smart Grid Software," *Proceedings of Innovative Smart Grid Technologies (ISGT), IEEE PES,* Washington, DC.

Stoy, J. 1977. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press.

Vlacheas, P., Giaffreda, R., Stavroulaki, V., Kelaidonis, D., Foteinos, V., Poulios, G., Demestichas, P., Somov, A., Biswas, A., and Moessner, K. 2013. "Enabling Smart Cities though a Cognitive Management Framework for the Internet of Things," *IEEE Communications Magazine* (51:6), pp. 102-111.

Walton, G., Longstaff, T., and Linger, R. 2009. "Computational Evaluation of Software Security Attributes," *Proceedings of the 42nd Hawaii International Conference on System Sciences* (HICSS-42), Hawaii.

Weber, R. 2013. "Internet of Things – Governance Quo Vadis?" *Computer Law & Security Review* (29), pp. 341-347.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., and Zorzi, M. 2014. "Internet of Things for Smart Cities," *IEEE Internet of Things Journal* (1:1), pp. 22-32.

