COMPOSITIONAL MODELS FOR THE INTERNET OF THINGS

Briener, Spencer (1);
Jung, Jooik (1);
Subrahmanian, Eswaran (2,1);
Sriram, Ram (1)

1: National Insititute of Standards and Technology;
2: Carnegie Mellon University

ABSTRACT

The modularity of components has enhanced the ability to create IoT systems by composing them from off the shelf. However, the breadth of technological choices and capabilities of component devices has made designing these systems harder to select, compose, implement and test, especially for dynamic systems. In this paper, we adopt formal tools from category theory (CT), a branch of mathematics whose central tenet is compositionality, to generate models for IoT systems. More specifically, we introduce a port-graph operad to represent the architectural designs of IoT systems. We use presheaf categories to construct generic IoT schemas to support modularity. Given this information, we briefly describe its relationship to control strategies of dynamical systems that model the interaction of components. Our approach balances genericity and specificity, providing interlinked schematic representations of system architecture and component representation.

Keywords: Internet of Things, Systems Engineering (SE), Design methods, Product modelling / models

Contact:
Subrahmanian, Eswaran
Carnegie Mellon University
United States of America
es3e@andrew.cmu.edu

Cite this article: Briener, S., Jung, J., Subrahmanian, E., Sriram, R. (2023) ‘Compositional Models for the Internet of Things’, in *Proceedings of the International Conference on Engineering Design (ICED23),* Bordeaux, France, 24-28 July 2023. DOI:10.1017/pds.2023.104

1 INTRODUCTION

The hallmark of the Internet of Things (IoT) is variety. There is obvious variety among the components themselves, both in their capabilities and their manufacture. There is also a huge variety among IoT applications, where we might design a system for the home (energy management), for the hospital (health monitoring), or for city hall ("smart city" technologies) (da Cruz et al., 2018). Current modeling approaches for IoT fall short (Noura et al., 2019). Some focus only on one aspect of an IoT system, such as sensor data analysis (Yin et al., 2018), but ignore other important aspects of the domain (actuation). Additional problems identified in earlier work include the need for dynamic modeling of the system environment (Weyer et al., 2015) and subsystem interaction in nested IoT systems-of-systems (Breiner et al., 2018).

One problem is that today's IoT systems are usually built on an ad-hoc, one-off basis. Consequently, the solutions developed for one situation rarely generalize to other contexts. To provide a more uniform view of the extreme diversity of IoT systems, we break the problem in two, characterizing two separate axes of variety: global and local. The global/local perspectives encourage very different representations. Local variety can be described by a map of the system's architecture and environment, the types of components involved, and how they interact. Every system has a different map. Global variety, on the other hand, is best represented by a catalog detailing the available options and their relevant performance characteristics. To describe systems in these terms, the system-specific architecture must provide a rubric for mapping component characteristics to system performance.

The approach presented here applies tools from category theory to address these representational needs (Breiner et al., 2016). Architectural descriptions are given as port-graphs, graphical structures where nodes are defined by their interfaces and edges represent interactions. The catalog uses a different representation (presheaves) to encode semantic information about components. Crucially, the composition of an IoT system from its components can be encoded as a functor transforming port-graphs into presheaves. It may help to view our proposed system from the perspective of its users; its goal is to synthesize the information provided by the users into a coherent picture of the system at hand. First, is the group defining and registering component models. These might be provided by the manufacturer or developed by a third party during related modeling work. Critically, a component model only needs to be written (well) once, and it can be shared among others who use the same hardware. For commodity components, a "component model" can be reduced to a simple list of standard specs like the cooling capacity of an air conditioner or the accuracy of a thermometer.

Next, the system architect defines the specifics of the situation at hand: What environmental variables will the system observe or control? What are the relevant parameters of these variables (e.g., control volume)? Generically, what types of components will be used to monitor and influence the system? As we will show, this architecture defines a parametric dynamical model of the system.

Following this, the system designer (who need not be the architect) can instantiate the generic components using choices from the component catalog. This parameterizes the dynamical model of the architecture and generates a behavioral model of the system as designed. This can be used to anticipate system performance against a range of control strategies and exogenous environmental behavior. Finally, in implementation, the system model can be extended with the device- and system-specific information like addresses, IDs, and passwords. These could be used to generate the messaging infrastructure and technology stack configuration required by the components on the ground.

The remaining sections of the paper are organized as follows. In Section 2, a short note on related work and its relation to the paper. Section 3 describes the tools from CT which we use to design system architectures and give descriptions of IoT components. In Section 4, we present the core IoT schema applicable to all components and two specialized schemas for sensors and actuators. . Discussions and conclusions are presented in Section 5.

2 RELATED WORK

An early work on IoT systems modeling is the Lighthouse Project IoT-A (Haller et al., 2013). This project provides a reference architecture model for the IoT, along with the definitions of a set of key building blocks. In order to formalize the model, they propose a domain model using UML diagrams. The domain model defines the main concepts in IoT and their relationships, and serves the role as a common lexicon and taxonomy supporting the design of concrete IoT system architectures. However, the IoT-A model is extremely complex for designing system architectures for responsive environments (Carrez et al., 2013).

Category theory, our main toolbox in this paper, has a long history in engineering model management and model-driven engineering (MDE). A proper introduction to CT and its history is beyond the scope of this paper, and we will generally omit formal definitions in favor of intuition and examples.

For more on CT in general, see (Bradley, 2018; Fong and Spivak, 2019). The discussion of port graphs and dynamical systems relies primarily on (Willems, 2007; Vagner et al., 2015). Presheaves are standard tools in CT, but our usage is most closely linked to the theory of categorical databases (Spivak, 2012), and knowledge representation (Spivak and Kent, 2012).

3 METHODS

In this paper, we aim to illustrate how to build models for IoT systems in a modular, off-the-shelf fashion used to construct the target system. To define these models, we use methods from CT, a branch of mathematics concerned with the formal representation of compositional structures, in which complex systems can be built up by combining smaller, simpler formal structures (Breiner et al., 2018).

Borrowing CT structures, Section 3.1 introduces port graphs, which we use as combinatorial representations for the architecture of IoT systems, and shows how these can be used to compose global dynamical models based on local descriptions. In Section 3.2 we introduce structured data models called presheaves to connect component dynamics with more typical product attributes like power usage and efficiency.

3.1 Port graphs

Port graphs are combinatorial structures that will serve as maps for our IoT systems. An example is shown in Figure 1; nodes represent system components, while edges represent interactions. Both nodes and edges must be labeled, indicating the types of components and interactions involved; we also use solid and dashed lines to distinguish between physical and digital interactions. The presence of both types of edges can characterize sensors and actuators. Computation and dynamical evolution involve only one or the other.

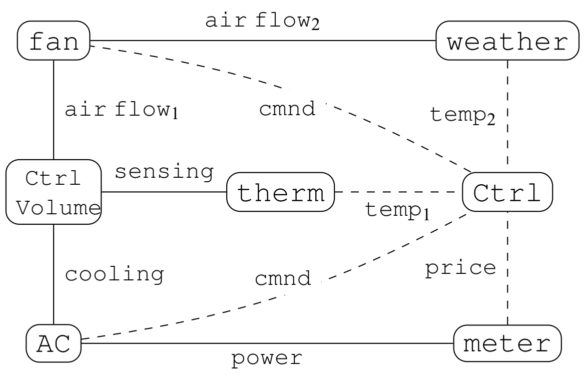

Figure 1. A port-graph model of an IoT cooling system. Solid and dashed lines represent physical and digital interactions, respectively.

https://doi.org/10.1017/pds.2023.104 Published online by Cambridge University Press

Unlike ordinary graphs, the construction of a port graph is highly constrained. Each node type is assigned a set of (typed) ports that it must fill; a port graph is only valid if the incoming edges to each node match its ports exactly. Moreover, if a node has several ports of the same type, the matching between ports and edges matters. For example, the control component `Ctrl` would behave very differently if we accidentally swapped the inside temperature $\texttt{temp}_1$ with the outside temperature $\texttt{temp}_2$. Put another way, every node in a port graph comes with a specified interface that the graph must implement.

The compositional structure of a port graph can be used to construct joint state descriptions of the system. We start by assigning a state space X_i to each (type of) node in the system. These may include both discrete states like the current mode of the AC unit ($X_{\texttt{AC}} = \{\texttt{off}, \texttt{low}, \texttt{high}\}$) and continuous variables like the temperature and humidity of the control volume. Similarly, we assign a state space Y_j to each interaction describing relevant features like the volume displacement of airflow or the (negative) heat transfer of a cooling process.

Given these local descriptions, the port graph provides instructions for assembling a global state space. When two components are linked by an interaction, this amounts to a declaration their port values must be equal. The power drawn by the AC is the same as the power sent from the meter; the temperature measured by the thermometer agrees with the message received by the control algorithm (which, due to error, need not agree with the true temperature of the control volume!).

A joint state for the composed system is a pair of states for the components that project to the same port value. Formally, this corresponds to a categorical operation called the *pullback*. The pullback inherits the port values of its components, allowing us to (carefully[1]) iterate this procedure, resulting in a global state space for the aggregate system.

The same approach can be used to construct dynamical models for the system (Vagner et al., 2015). The key step is to replace states with behaviors, thought of as time-varying states $x(t)$. Now X_j is a set of functions that encodes not only the states of a component but also how future states depend on the past. These specifications may include both differential equations for continuous evolution and state-transition systems (of various types) for discrete phenomena. More generally, these can be combined to provide models for hybrid systems (Lerman and Schmidt, 2019).

Nonetheless, we demand that component behaviors determine port behaviors, defining a functional (function between functions) $p : X_i \to Y_j$.[2] Since these functions are the only ingredients needed in the construction above, we can form a global dynamical system using the same strategy of iterated pullback.

3.2 Presheaves

The previous section was on modeling the architecture of an IoT system using port graphs and using that representation to derive state-based and dynamical models of its behavior. However, the derivation assumes that the individual components already come equipped with dynamical models of local behavior. In this section, we show how to extract these descriptions from more familiar component specifications.

Each component type can be characterized by a set of relevant attributes, which define joint constraints on the simultaneous behaviors at a component's boundaries. For example, an accuracy ϵ for the thermometer shown in Figure 1 indicates that $|T(t) - \mathbb{T}(t)| < \epsilon$, where T and $\mathbb{T}$ represent the true temperature and the measured temperature, respectively. Actuators are a bit more complicated because we have to maintain a model of their modes and commands, encoded as a function $\texttt{Mode}_i \times \texttt{Cmnd}_i \to \texttt{Mode}_i$ for each component i.

[1] If composition involves two (or more) interactions k and ℓ at once, then the pullback should be constructed over the product space $Y_k \times Y_\ell$.

[2] Usually p is limited by a "no-lookahead" condition. For component behaviors $x_i(t)$ and port behaviors $y_i = p(x_i)$

$$\forall t. \Big(\big[\forall t' < t. \, x_1(t') = x_2(t')\big] \Rightarrow y_1(t) = y_2(t) \Big).$$

To construct a categorical model of this data, we rephrase these descriptions in terms of functional (many-one) relationships (written $f : X \to Y$). For example, the numeric attributes of the thermometer (e.g., accuracy) correspond to arrows $\texttt{therm} \to \mathbb{R}^+$. In addition to the geometric structure of a graph, categories also include an algebraic operation of composition. For example, accuracy applies to any measurement, not just temperature, so it is more appropriate to represent ϵ as a composite

$$\begin{tikzcd}
\texttt{therm} \arrow[r, tail, "\texttt{asSensor}"'] \arrow[rr, bend left, "\epsilon=\texttt{asSensor.accuracy}"] & \texttt{Sensor} \arrow[r, "\texttt{accuracy}"'] & \mathbb{R}^+.
\end{tikzcd}$$

The special $\rightarrowtail$ arrow above, called a *monic*, is the categorical generalization of a subset or "*isA*" relation. As above, inheritance across subclass/superclass boundaries becomes a special case of composition.

Formally, a *presheaf model* involves two pieces: a category $\mathcal{S}$, called the *schema*, and a categorical mapping (called a functor, see next section) $P : \mathcal{S} \to \mathbf{Sets}$. While the schema $\mathcal{S}$ defines the generic elements of an IoT systems, a presheaf $P : \mathcal{S} \to \mathbf{Sets}$ specifies a *specific* system by mapping these abstract elements to concrete sets and functions. For example, the system shown in Figure 1 would have $P(\texttt{Comp}) = \{\texttt{AC}, \texttt{fan}, \texttt{meter}, \texttt{therm}, \texttt{weather}\}$.

We can also represent relationships between different IoT systems using two-dimensional mappings called natural transformations which are morphisms between functors:

$$\begin{tikzcd}
\mathcal{S} \arrow[r, bend left, "P"{name=U}] \arrow[r, bend right, "Q"'{name=D}] & \mathbf{Sets} \arrow[from=U, to=D, Rightarrow, "\alpha"]
\end{tikzcd} .$$

For a fixed schema $\mathcal{S}$, natural transformations provide the arrows in a category of presheaves, denoted $\widehat{\mathcal{S}}$. It is traditional to use a double-arrow notation ($\alpha : P \Rightarrow Q$) to emphasize the two-dimensional nature of these maps.

Presheaf models for systems can be composed using operations called colimits, which generalize set-theoretic unions to many other types of mathematical structures. A colimit uses a specification of the *overlaps* between systems in order to generate a model of the whole. Figure 2 shows the composition of two components C and C' along an abstract channel between two systems.

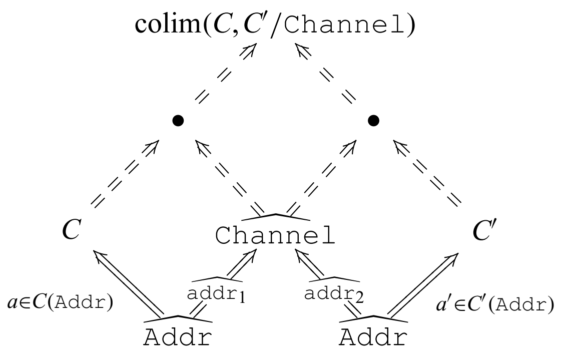

Figure 2. Colimit composition of component presheaves $C, C' \in \texttt{CompCat}$ *along a generic* $\texttt{Channel}$. *Bullets and dashed arrows are additional outputs of the construction.*

4 IOT SCHEMA

In this section, we define an IoT schema, $\mathcal{S}$, which structures our presheaf models for IoT systems. We present the schema in three stages: first a core schema (Section 4.1) relevant for all components, denoted $\mathcal{S}_0$, followed by more specialized models for sensors and actuators (Section 4.2). We will suppress computational/numeric attributes like $\texttt{accuracy} : \texttt{Sensor} \to \mathbb{R}^+$ from our schemas in order to avoid clutter.

4.1 Core schema

We begin with the core schema $\mathcal{S}_0 \subseteq \mathcal{S}$ given in Figure 3, which defines the common structure shared by all IoT components. The central rectangle in Figure 3 associates each component with a state-transition system defined in terms of modes and commands. This is a common design pattern that occurs often in CT, whenever one collection of sets (the commands) can be used to act on a second collection (the modes). The corner notation '$\lrcorner$' indicates that `Transition` is a defined concept (constructed using an operation called *pullback*) that consists of compatible mode-command pairs. The result of each transition is encoded in a second map $\mathtt{post} : \mathtt{Transition} \to \mathtt{Mode}$, restricted by the obvious constraint that transformations do not hop between devices ($\mathtt{pre.compM} = \mathtt{post.compM}$).

$$\begin{tikzcd}
 & \mathtt{Action} \arrow[dl, "\mathtt{hasTime}"'] \arrow[dr, "\mathtt{invokes}"] & \mathtt{Transition} \arrow[r, bend left, "\mathtt{post}"] \arrow[r, "\mathtt{pre}"'] \arrow[d, "\mathtt{cmnd}"] \arrow[dr, phantom, "\lrcorner", very near start] & \mathtt{Mode} \arrow[d, "\mathtt{compM}"'] & \mathtt{Channel} \arrow[d, shift right=1, "\mathtt{addr}_1"'] \arrow[d, shift left=1, "\mathtt{addr}_2"] & \\
\mathtt{Time} & & \mathtt{Cmnd} \arrow[r, "\mathtt{compC}"'] & \mathtt{Comp} & \mathtt{Addr} \arrow[l, two heads, "\mathtt{ofComp}"] \arrow[r, "\mathtt{protocol}"'] & \mathtt{Protocol}
\end{tikzcd}$$

$\mathtt{pre.compM}=\mathtt{post.compM}$ $\quad$ $\mathtt{addr}_1\mathtt{.protocol}=\mathtt{addr}_2\mathtt{.protocol}$

Figure 3. The core schema $\mathcal{S}_0 \subseteq \mathcal{S}$ defines properties shared by all IoT components.

In addition to the state-transition structure for our components, $\mathcal{S}_0$ also encodes a history of control on the left-hand side of the diagram and the networking structure of the system on the right. Both pieces involve a common design pattern called a *span*. Although CT models are expressed in terms of functional (many-one) relationships, we can still represent a many-many relationship using a pair of arrows $\bullet \leftarrow \bullet \rightarrow \bullet$.

The networking elements of the schema are particularly important for linking our presheaf and port-graph representations: a presheaf's channels (resp. addresses) directly correspond to the digital (dashed) interactions (resp. ports) shown in the system map given by the port graph. However, these are only some of the interactions. In the next two sections, we will introduce physical quantities called environmental variables to keep track of physical interactions and model the way that these relate to sensing and actuation.

4.2 Sensing and actuation

Next we extend the `Comp` object, introduced in the core schema, to include more specific information about sensing and actuation. In CT, subclasses are modeled by special arrows called monics, indicated by tailed arrows:

$$\begin{tikzcd}
\mathtt{Sensor} \arrow[r, tail, "\mathtt{asCompS}"] & \mathtt{Comp} & \mathtt{Actuator}. \arrow[l, tail, "\mathtt{asCompA}"']
\end{tikzcd}$$

Both types are characterized by direct interaction with the physical world, a fact which we model through the concept of an *environmental variable* (`EnvVar`). Each variable x is characterized by a space of possible state values $\mathtt{EnvVar}(x)$, which might be scalar- or vector-valued. For example, the link between the AC unit and the control volume can be parameterized by (negative) heat transfer and change in humidity, corresponding to a two-dimensional state space. As we have seen repeatedly, the indexed set of value spaces can be wrapped into a single function $\mathtt{varE} : \mathtt{EnvState} \to \mathtt{EnvVar}$.

The bulk of the sensor schema, which is shown in Figure 4, defines special subsets of actions and commands that are explicitly linked to the environment. First is the subclass itself: $\mathtt{Sensor} \rightarrowtail \mathtt{Comp}$. Next, each sensor has a set of measurement capabilities, called observations, special commands that observe the physical world.

Each observation is characterized by the environmental variable that it observes ($\mathtt{varO} : \mathtt{Obsv} \to \mathtt{EnvVar}$), and a variable x is observable (`ObsVar`) if $x = o.\mathtt{varO}$ for some observation o. Observations are particularly important for our model because they determine the physical ports associated with each sensor in the port-graph representation of the system.

https://doi.org/10.1017/pds.2023.104 Published online by Cambridge University Press

$$\begin{tikzcd}
 & \texttt{value.varE=invokes.varO} & \\
\texttt{EnvState} & \texttt{Query} \arrow[l, "\texttt{value}"'] \arrow[r, tail, "\texttt{asAction}"] \arrow[d, "\texttt{invokes}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \texttt{Action} \arrow[d, "\texttt{invokes}"] \\
\texttt{EnvVar} \arrow[u, leftarrow, "\texttt{varE}"] & \texttt{Obsv} \arrow[l, "\texttt{varO}"'] \arrow[r, tail, "\texttt{asCmnd}"] \arrow[d, "\texttt{sensor}"] \arrow[dl, two heads] & \texttt{Cmnd} \arrow[d, "\texttt{compC}"] \\
\texttt{ObsVar} \arrow[u] & \texttt{Sensor} \arrow[r, tail, "\texttt{asComp}"'] & \texttt{Comp} \\
 & \texttt{sensor.asComp=asCmnd.compC} &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \texttt{postA.mode=trans.post} & \\
 & \texttt{preA.mode=trans.pre} & \\
\texttt{Transition} \arrow[d, bend right, "\texttt{post}"'] \arrow[d, bend left, "\texttt{pre}"] & \texttt{ActTrans} \arrow[l, "\texttt{trans}"'] \arrow[d, bend right, "\texttt{preA}"'] \arrow[d, bend left, "\texttt{postA}"] \arrow[dl, phantom, "\llcorner", very near start] & \texttt{EnvVar} \\
\texttt{Mode} \arrow[d, "\texttt{compM}"'] & \texttt{ActState} \arrow[l, "\texttt{mode}"'] \arrow[d, "\texttt{actS}"'] & \texttt{Infl} \arrow[u, "\texttt{varI}"] \arrow[dl, "\texttt{actI}"'] \\
\texttt{Comp} & \texttt{Actuator} \arrow[l, tail, "\texttt{asComp}"] & \\
 & \texttt{mode.compM=actS.asComp} &
\end{tikzcd}$$

Figure 4. Schema elements associated with the `Sensor` *and* `Actuator` *subclasses.*

Next, we use observations to define queries: a `Query` is an `Action` that invokes an observation; formally, this is expressed as a pullback over `Cmnd`. Like all actions, queries have time stamps. In addition, we represent the measured state of each query as a function `Query` → `EnvState` (constrained by the appropriate `EnvVar`).

Actuation also requires some special features. Like sensors, actuators are explicitly linked to environmental variables, in this case via a span of influences (`Infl`). Formally, influences are analogous to observations: they determine the physical ports associated with each actuator. Because both classes of components are linked to the same set of environmental values, we can compose them to build the feedback loops needed for control.

On the other hand, there is no direct link to `EnvState`. Whereas a sensor reads and reports a value, an actuator typically cannot set a state value unilaterally. Instead, we equip each actuator with its own set of local states (`ActState`). An actuator's state determines its mode, but also includes continuous parameters describing the way that it is embedded in the environment. Air conditioners and fans are too static to provide natural examples; instead, think of the location, altitude, and orientation of a drone, the joint angles of a robotic arm, or the state of charge in a battery. Finally, we assume a transition system for actuator states in parallel to the one for modes, formalized in terms of a CT design pattern known as equivariance, (Bernstein and Lunts, 2006).

5 DISCUSSION AND CONCLUSION

In contrast to existing approaches, we address the problem of component descriptions with compositionality as a primary criterion. The distinction between composition and compositionality may seem subtle at first, but there is a clear difference: Composition is perceived as the question of how to put the pieces together, whereas compositionality defines the semantics of their composed behaviors.

In this paper, we formulate a way to create and adapt descriptions of components to address the problem complexity and interoperability through composition using typed categorical structures as in port graphs, schemas, and presheaves. By nesting different granularity of IoT components, CT allows the components to be easily composed and substituted without having to worry about the entire structure. For instance, you can have multiple control units where several of these control smaller groups of components and another one that supervises and controls the whole system. This concept of "systems of systems" is achieved through CT's exceptional compatibility with compositional systems like those of IoT.

Apart from these, CT's close connections with formal logic and a wide variety of mathematical tools under its belt reflect the breadth of application, now made broader by advances in applied category theory. More specifically, in order to link different representations across multiple domains (ie. sensor specification sheets) as a common representation based on formal logic, CT provides the necessary tools to achieve that. We can also use CT to model and evaluate different behaviors of dynamical systems such as the IoT and hyperdynamical systems.

Another advantage of CT relies on the fact that relationships in CT-based models are considered as “first-class citizens”. With the level of flexibility of CT’s toolbox, we benefit in terms of both information representation and model integration (Wisnesky et al., 2017). Port-graphs allow us to manipulate and understand the interactions among internal representations of components. There is also the ability to encode relationships between systems through natural transformations on presheaves. Natural transformations establish a relationship between the original data and the transformed data. Recall that functors are used to define these data transformations as well as comparing information models. Moreover, colimits allow for the formal integration of information models based on their overlapping components. Composition of IoT system models, for example, `EnvVar` is used as the overlapping entity to connect the sensor and actuator schemas. These exemplars are intended to illustrate the potential of compositionality in CT.

There is a lack of tools to implement the categorical structures and its algebraic power. Early tools in this area are appearing especially for information model and data integration as illustrated in Section 5 of (Breiner et al., 2019). The need for tools is a critical bottleneck for widespread adoption or even illustration through a functioning prototype. The advent of functional programming languages that are categorically inspired also provides another toll in CT’s quiver. The potential for CT in the design complex system is quite substantial and will be realized as the tools develop.

In this paper, we develop an early version of design methodology using a branch of mathematics, category theory, to address the compositionality of models for IoT systems. To this end, we adopt formal tools from CT to introduce port graphs, which represent the architectural designs of IoT systems. we introduce a presheaf category to build generic IoT schemas that encompass the descriptions of IoT components. To tie the system together, we present a control strategy for generalizing the interactions among IoT components. We believe by addressing compositionality as a requirement for system design, we have created the seeds for developing standards for IoT system that is generic but is capable of connecting to the specifics which may include extent standards and tools.

REFERENCES

Bernstein, J. and Lunts, V. (2006), *Equivariant sheaves and functors*, Springer.

Bradley, T.D. (2018), “What is applied category theory?”, *arXiv preprint arXiv:1809.05923*.

Breiner, S., Pollard, B. and Subrahmanian, E. (2019), *Workshop on Applied Category Theory: Bridging Theory and Practice*, Special Publication 1249, US Department of Commerce, National Institute of Standards and Technology, http://doi.org/10.6028/NIST.SP.1249.

Breiner, S., Sriram, R.D. and Subrahmanian, E. (2018), “Compositional models for the internet of everything”, in: *2018 AAAI Spring Symposium Series*.

Breiner, S., Subrahmanian, E. and Sriram, R.D. (2016), “Modeling the internet of things: a foundational approach”, in: *Proceedings of the Seventh International Workshop on the Web of Things*, ACM, pp. 38–41.

Carrez, F., Bauer, M., Boussad, M., Bui, N., Jardak, C., De Loof, J., Magerkurth, C., Meissner, S., Nettsträter, A., Olivereau, A. et al. (2013), “Internet of things—architecture iot-a, deliverable d1. 5—final architectural reference model for the iot v3. 0”, *European Union, 7th Framework Programme*.

da Cruz, M.A., Rodrigues, J.J.P., Al-Muhtadi, J., Korotaev, V.V. and de Albuquerque, V.H.C. (2018), “A reference model for internet of things middleware”, *IEEE Internet of Things Journal*, Vol. 5 No. 2, pp. 871–883.

Fong, B. and Spivak, D.I. (2019), *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*, Cambridge University Press.

Haller, S., Serbanati, A., Bauer, M. and Carrez, F. (2013), “A domain model for the internet of things”, in: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, IEEE, pp. 411–417.

Lerman, E. and Schmidt, J. (2019), "Networks of hybrid open systems", *arXiv preprint arXiv:1908.10447*.
Noura, M., Atiquzzaman, M. and Gaedke, M. (2019), "Interoperability in internet of things: Taxonomies and open challenges", *Mobile Networks and Applications*, Vol. 24 No. 3, pp. 796–809.
Spivak, D.I. (2012), "Functorial data migration", *Information and Computation*, Vol. 217, pp. 31–51.
Spivak, D.I. and Kent, R.E. (2012), "Ologs: a categorical framework for knowledge representation", *PLoS One*, Vol. 7 No. 1, p. e24274.
Vagner, D., Spivak, D.I. and Lerman, E. (2015), "Algebras of open dynamical systems on the operad of wiring diagrams", *Theory and Applications of Categories*, Vol. 30 No. 51, pp. 1793–1822.
Weyer, S., Schmitt, M., Ohmer, M. and Gorecky, D. (2015), "Towards industry 4.0-standardization as the crucial challenge for highly modular, multi-vendor production systems", *Ifac-Papersonline*, Vol. 48 No. 3, pp. 579–584.
Willems, J.C. (2007), "The behavioral approach to open and interconnected systems", *IEEE Control Systems Magazine*, Vol. 27 No. 6, pp. 46–99.
Wisnesky, R., Breiner, S., Jones, A., Spivak, D.I. and Subrahmanian, E. (2017), "Using category theory to facilitate multiple manufacturing service database integration", *Journal of Computing and Information Science in Engineering*, Vol. 17 No. 2, p. 021011.
Yin, H., Wang, Z. and Jha, N.K. (2018), "A hierarchical inference model for internet-of-things", *IEEE Transactions on Multi-Scale Computing Systems*, Vol. 4 No. 3, pp. 260–271.

CAMBRIDGE
UNIVERSITY PRESS