EAI Endorsed Transactions
on Context-aware Systems and Applications

Research Article ICST.ORG

Representing and Reasoning with the Internet of Things: a Modular Rule-Based Model for Ensembles of Context-Aware Smart Things

S. W. Loke[1,*]

[1]Department of Computer Science and Information Technology, La Trobe University, Kingsbury Drive, VIC 3086, Australia

Abstract

Context-aware smart things are capable of computational behaviour based on sensing the physical world, inferring context from the sensed data, and acting on the sensed context. A collection of such things can form what we call a *thing-ensemble*, when they have the ability to communicate with one another (over a short range network such as Bluetooth, or the Internet, i.e. the Internet of Things (IoT) concept), sense each other, and when each of them might play certain roles with respect to each other. Each smart thing in a thing-ensemble might have its own context-aware behaviours which when integrated with other smart things yield behaviours that are not straightforward to reason with. We present Sigma, a language of operators, inspired from modular logic programming, for specifying and reasoning with combined behaviours among smart things in a thing-ensemble. We show numerous examples of the use of Sigma for describing a range of behaviours over a diverse range of thing-ensembles, from sensor networks to smart digital frames, demonstrating the versatility of our approach. We contend that our operator approach abstracts away low-level communication and protocol details, and allows systems of context-aware things to be designed and built in a compositional and incremental manner.

Received on 10 November, 2015; accepted on 6 February, 2016; published on 09 March, 2016

Keywords: smart artifacts; context-aware artifacts; operator language; compositionality; Internet-of-Things (IoT)

doi:10.4108/eai.9-3-2016.151113

Introduction

Due to developments in pervasive computing and embedded systems, there has been a growing number of types and varieties of *smart things*, which are capable of computationally based behaviours, communicating with each other over the Internet[1] or over short range networks, and sensing the physical world [4, 30]; they can have context-aware behaviours [41], and are becoming even cloud-enabled [22]. These things can then intelligently respond to the context or situation they sense, via some computation. Smart things might also be everyday objects (e.g., vases, books, picture frames, furniture, clothings, etc)[2] endowed with embedded processors, intelligent sensors, mobile devices with sensors, appliances [26], or simply new types of devices [48].

A number of paradigms have been employed to program such things, including Web service based methods,[3] that assume each device has an embedded Web server that receives and responses to invocations, UPnP service styles,[4] JINI,[5] and peer-to-peer based styles such as JXTA,[6] and various visual editors such as [21, 25, 38]. For some time now, we have seen the application of rule-based programming for behaviours of collections of smart things, such as [24, 25, 49, 50],[7]

*Corresponding author. Email: s.loke@latrobe.edu.au

[1]http://www.theinternetofthings.eu

[2]The e-gadgets model view everyday objects as components, see http://www.extrovert-gadgets.net/

[3]See http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01 and [31]

[4]See http://www.upnp.org/

[5]See http://www.jini.org/wiki/Main_Page

[6]See https://jxta.dev.java.net/

[7]Others are https://www.onx.ms/ and http://appinventor.mit.edu/

though such work generally do not consider reasoning with groups of rules. Basically, context information gathered via sensors are reasoned with and appropriate action taken based on these rules.

The general structure of smart things, sensing, reasoning and acting [33] corresponds to context-aware mobile computing systems as early as 1994 [43], and in fact, many context-aware systems do have that general structure.

However, when a collection of such smart things are put together, and if they can sense and react to each other, it is not easy to reason about the collective behaviour of the ensemble, and it is not easy to determine what will happen as a result of combining such smart things.

Following the design philosophy first introduced in [36] and also used in [35], in this paper, we elaborate on an approach (and formalism), which we call Sigma, to represent compositions (or collections) of smart things that can sense their environment and context and act in response to what they sense. The formalism comprises a language of operators, inspired by modular logic programming [12], for composing systems, the advantage of using operators being to encapsulate the cooperation between systems, leaving it to the implementation to realize that cooperation. The compositions are not necessarily physical compositions of the things (e.g., if the things are vases, a composition here does not necessarily mean gluing the things together), but rather, are compositions of their representations. Such compositions specify how things may cooperate in their sensing and acting; we call a collections of things that can communicate with each other, and whose behaviours are composed, a *thing-ensemble*. Note that our view of cooperation here is the idea of a set of devices working together complementarily for a purpose, and coordinated to achieve an effect via a set of operators with predefined semantics, not autonomous cooperation as in multiagent systems research. It should be noted that a thing-ensemble is, hence, different from an arbitrary set of appliances or sensors, in that we assume that the things in the ensemble can interact with each other, in the way we describe later.

We show how a variety of thing-ensembles can be represented, ranging from the smart living room applications, sensor networks to smart digital frames, and their behaviours specified. We use the term *ensembles* to emphasize that the things are in some kind of coordinated relationship with each other in order to fulfill their own tasks or end-user specified tasks, and things typically play different roles in such a cooperation. Moreover, this paper deals with the formalism to specify such compositions but not the implementation details.

The twofold purpose of our formalism for representing and reasoning with thing-ensembles is as follows:

- *Abstraction*. We present a set of operators that provides well-defined compositions of smart context-aware things. This enables reasoning about what collective behaviours could emerge when an ensemble of such smart things interact or are composed. While this may seem constraining (restricted to the collection of operators we have), we present a small set of operators which can be used to model interactions in a range of scenarios, showing that even a small set of operators can be quite expressive and useful.

- *Incremental and compositional design for the Internet of Things*. We also argue for the need of a design approach that provides a means to build or extend thing-ensembles in an incremental manner, i.e., new things can be added to a collection, and new composed behaviours attainable without considerable developer effort, or considerable change to the existing thing-ensemble. The idea is that smart context-aware behaviours of a collections of smart things (and/or computer devices) can be effectively "grown" over time. While in a different context, a parallel of this idea is McCarthy's notion of *elaboration tolerance* (especially, additive types of elaborations), which is "the ability to accept changes to a person's or a computer program's representation of facts about a subject without having to start all over."[8] In our case, it is the addition of new smart things (e.g., new computational nodes, storage, sensors, appliances, etc) to an existing system (a collection of smart things), without needing to start over or redesigning the whole system. Our examples in a range of scenarios demonstrate this idea where the behaviour of each thing is simply modelled as a module of rules. There are pragmatic advantages of this since if old devices can be combined with newly added devices, then reuse of the old devices in the presence of the new is facilitated, or if newly added devices can override (or suppress) the functionality of older devices in a well-defined manner, then the new behaviour can be better understood.

The contributions of this paper are twofold:

- Theoretically, the Sigma formalism serves as a basis for a unified model of cooperating thing-ensembles that act on an understanding of their situations. We seek to expose and formalize the

[8] http://www-formal.stanford.edu/jmc/elaboration/elaboration.html

underlying structure of a diverse range of thing-ensembles, which might seem rather different on the surface, but in fact, have common structure and behaviour. The fact that these operators can be defined which capture how different systems of context-aware things work together in different contexts suggests the generality of the operators and emphasizes the common features of these systems.

- Practically, the Sigma formalism (perhaps with syntactic sugar if needed) can be used in two ways:
  - as a *specification language* for designing context-aware IoT applications [41, 47, 58]; the formalism encourages a modular approach to the problem, and also allows incremental development where an existing system can be modelled and then, add-ons can be represented and combined with the existing system via suitable operators; a smart thing can be modelled as a rule-base but then implemented in hardware, or it can be a software-programmable device - our approach is agnostic to this, and
  - as a *scripting language* for programming behaviours of cooperating context-aware things, where the combined behaviour is embedded within the operators, thereby providing a high-level of abstraction, without the programmer needn't to deal with underlying protocols; but an interpreter for the operators would need to be implemented first.

In the next section, we present a brief overview of operators we will use in composing behaviours of things. Then, we will describe scenarios of thing-ensembles. Related work is then reviewed, and finally, we conclude with future work.

Sigma Operators for Composing Context-Aware Systems

Context-Aware Systems

We use the operator language and abstract model of context-aware systems from [36], but extend the repertoire of operators here. We use the term *context-aware system* to refer to three categories of systems:

- systems comprising only of sensors,
- systems with sensors and context interpretation components, and
- systems with sensors, context interpretation components, and situation reasoners.

The above components are explained below.

The general model of a context-aware system R is a triple of the form (Σ, Π, Θ). Σ is a finite set of sensors where each sensor σ_i is a function which senses a part of the world (in which the sensor is situated at the time) $W \in \mathbf{W}$ and produces a set of sensor readings $G \in \mathbf{G}$, i.e. $\sigma_i : \mathbf{W} \rightarrow \mathbf{G}$. We let $\mathbf{W}$ and $\mathbf{G}$ remain opaque as our discussions do not require their details, but explicit and precise definitions could be given for them; also, in our case, we assume that the sensors Σ are attached to (or part of) the particular smart thing (e.g., a vase embedded with touch sensors and a microphone). Often, we will write $reading(\sigma_i) = G$ to denote when reading from the sensor σ_i has value G.

In this paper, a context-aware system can map to a thing with sensing, networking and computational capabilities. Or a context-aware system might comprise a thing and associated software and hardware that tracks the thing and provides context-aware behaviour associated with the thing.

The interpreter Π can then be defined as a mapping from sensor readings $\mathbf{G}$ to some context (e.g., a symbolic location such as a room number) $C \in \mathbf{C}$ (which we assume are concepts grounded in some ontology such as SOUPA[9] and CONON [52], or other ontologies as surveyed in [7, 57]), i.e., $\Pi \subseteq (\mathbf{G} \times \mathbf{C})$. So, given $W \in \mathbf{W}$, and suppose $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, and $\sigma_1(W) = G_1$, …, $\sigma_n(W) = G_n$ (for $G_i \in \mathbf{G}$), then Π can be applied to interpret each G_i to obtain a set of contexts $\{C_1, \ldots, C_n\}$, denoted by $\Pi(G_i) = C_i$ (taken here to mean $(G_i, C_i) \in \Pi$). We will also often write the mapping $(G_i, C_i) \in \Pi$ as a rule of the form $(G_i \Rightarrow C_i) \in \Pi$, for readability.

The situation reasoner Θ is a relation mapping sets of contexts or situations to situations, i.e. $\Theta \subseteq (\wp(\mathbf{C} \cup \mathbf{S}) \times \mathbf{S})$, where $\mathbf{S}$ is a set of situations (again, which we assume grounded in some ontology). We will also often write the mapping $(\{C_1, \ldots, C_k, S_1, \ldots, S_l\}, S) \in \Theta$ as $(\{C_1, \ldots, C_k, S_1, \ldots, S_l\} \Rightarrow S) \in \Theta$, for readability.

Examples of aggregating context to infer situations can be found in the literature [2, 16, 56, 57]. Our model makes the distinction between the notions of context and situation following the work such as [17, 32], where context information is used to "characterize the situation of an entity," modelled via Θ. While we do not commit to any ontology here, in practice, we assume that systems do return inferred contexts or situations using concepts from a "well-known" ontology in order to facilitate interoperability.

Moreover, we define the relation denoted by $\vdash$ between systems and pairs of the form (W, S) where $W \in \mathbf{W}$ and S is some situation, such that $R \vdash (W, S)$ if and only if R recognizes S when sensing part of the world W, or alternatively, if one places R in the part of

[9] http://pervasive.semanticweb.org/soupa-2004-06.html

the world W where S is happening, S will be detected by R. R may be in a world W where S is not happening, but as soon as S happens, it detects it.

A situation that is recognized by a system is then either computed from contexts (recognized by some sensors and the interpreter) and other situations (if any) via Θ. This meaning of the relation $\vdash$ can be expressed recursively as follows in a rule written in the form $\frac{premises}{conclusion}$:

$$\frac{\begin{array}{l}(\{C_1,\ldots,C_m,S_1,\ldots,S_k\} \Rightarrow S) \in \Theta,\\ \quad \text{for some } m, \text{where for each } C_i, i \in \{1,\ldots,m\},\\ \quad \Pi(\sigma(W)) = C_i, \text{for some } \sigma \in \Sigma,\\ \quad \text{and for some } k, \text{where for each } S_i, i \in \{1,\ldots,k\},\\ \quad (\Sigma,\Pi,\Theta) \vdash (W,S_i)\end{array}}{(\Sigma,\Pi,\Theta) \vdash (W,S)} \; (one-system)$$

We represent context recognition by a system R, i.e., $R \vdash (W,C)$ for some context C, where $\Pi(\sigma(W)) = C$, for some $\sigma \in \Sigma$.

Sigma Operators on Context-Aware Systems

The Sigma operators on context-aware systems form a small language of Sigma expressions, given by the following EBNF syntax:

$$\begin{array}{lcl} Q & ::= & R \mid Q + Q \mid Q \ominus Q \mid Q \oslash Q \\ E & ::= & Q \mid E \oplus E \mid E \otimes E \end{array}$$

The Sigma expressions defined by Q only are called *Q-expressions* and those defined by E are called *E-expressions*. The idea is that, given a thing-ensemble, i.e., a collection of context-aware things, the Sigma operators can be used to define different compositions of things from this collection. For example, given a thing-ensemble of three system $\{R_1, R_2, R_3\}$, different Sigma expressions can be defined over them, such as $R_1 + R_2 + R_3$ and $R_1 \oplus (R_2 + R3)$. Moreover, for these expressions to be meaningful, there must be some way for the R_is to interact with each other, in the way that we define below. We distinguish between the thing-ensemble and the Sigma expressions defined over (perhaps some subset of the) things in the thing-ensemble, since each Sigma expression describes only *one way (among many) in which the things can work together*.

Informally, the meaning of the operators are as follows, extending our relation $\vdash$ given above. The union of two context-aware systems used in attempting to recognize C (or a situation S), denoted by $R_1 \oplus R_2 \vdash C$, means that context C (or situation S with $R_1 \oplus R_2 \vdash S$) is recognized either using R_1 or R_2, succeeding if either succeeds. Note that this can be nondeterministic but one could employ short-circuit evaluation in practice, that is, try R_1 first and only on failure try with R_2.

Below, C can be replaced by S. The *intersection* of two context-aware systems used in attempting to recognize C, denoted by $R_1 \otimes R_2 \vdash C$, means that context C is recognized using both R_1 and R_2, succeeding if both succeeds.

The *tight-union* of two context-aware systems used in attempting to recognize C, denoted by $R_1 \otimes R_2 \vdash C$, means that context C is recognized using both R_1 cand R_2 working in a tightly coupled combined manner (as we define below).

The *restriction* of two context-aware systems used in attempting to recognize C, denoted by $R_1 \ominus R_2 \vdash C$, means that context C is recognized using rules from R_1 and R_2 with joint conditions (as we define below). The idea is that the rules in the situation reasoner would have joint conditions. For example, if $R_1 = (\Sigma_1, \Pi_1, \Theta_1)$ and $R_2 = (\Sigma_2, \Pi_2, \Theta_2)$, whenever $(\mathbf{S_1} \Rightarrow S) \in \Theta_1$, where $\mathbf{S_1} \subseteq (\mathbf{S} \cup \mathbf{C})$ and $S \in \mathbf{S}$, and also $(\mathbf{S_2} \Rightarrow S) \in \Theta_2$ (in other words, both Θ_1 and Θ_2 have a rule for S), then we join their conditions when applying the rules to infer S, i.e. we must use joint condition rules of the form $(\mathbf{S_1} \cup \mathbf{S_2} \Rightarrow S)$ to infer S. This generalizes to three or more systems, say with R_3 having $(\mathbf{S_3}, S) \in \Theta_3$, we have $(\mathbf{S_1} \cup \mathbf{S_2} \cup \mathbf{S_3} \Rightarrow S)$. Note that this is different from intersection in that with intersection of R_1 and R_2, each independently try to infer S, whereas with restriction, R_1 and R_2 can share sensors and context interpretation, and even situation reasoning rules, but at any step, the rules for a corresponding situation must be joined in their conditions.

The *overriding union* of one context-aware system by another in attempting to recognize C, denoted by $R_1 \oslash R_2 \vdash C$, means that the rules of R_1 takes precedence over those of R_2 whenever the same situation is being considered. For example, if $R_1 = (\Sigma_1, \Pi_1, \Theta_1)$ and $R_2 = (\Sigma_2, \Pi_2, \Theta_2)$, whenever $(\mathbf{S_1} \Rightarrow S) \in \Theta_1$, where $\mathbf{S_1} \subseteq (\mathbf{S} \cup \mathbf{C})$ and $S \in \mathbf{S}$, and also $(\mathbf{S_2} \Rightarrow S) \in \Theta_2$ (in other words, both Θ_1 and Θ_2 have a rule for S), then we use the rule from R_1 only, i.e. $(\mathbf{S_1} \Rightarrow S)$, and ignore any others for the same situation (of the form $(\mathbf{S_2} \Rightarrow S)$) from Θ_2. This operator is inspired by the notion of input suppression from Brook's subsumption architecture [13], as we illustrate below.

The operators $\oplus$ and $\otimes$ corresponds to logical "or" and"and" respectively, and operators in Q-expressions can be rewritten as single systems with contents from the component systems in the way given by the operational semantics of the operators below. In an implementation of these operators, it may not be possible to physically take the actual constituent components and combine them, but in fact, appropriate (e.g., Web service) interfaces can be used by each component to expose their sensors, interpreter and situation reasoner components, and then calls are

made to these interfaces to emulate effectively a single virtual system (the virtual system represented by a Q-expression).

The rules in Figures 1 and 2 provide the operational semantics, defining the modified relation $\vdash$ more precisely. Note that in rules $(one-system)$, (tu), (res) and (ovr), m and k could be zero for a given member of Θ, in which case, there is no context or situation in the antecedent.

$$\frac{\begin{array}{l}(\{C_1,\ldots,C_m,S_1,\ldots,S_k\}\Rightarrow S)\in\Theta,\\ \text{for some } m, \text{where for each } C_i, i\in\{1,\ldots,m\},\\ \Pi(\sigma(W))=C_i, \text{for some } j \text{ and } \sigma\in\Sigma,\\ \text{and for some } k, \text{where for each } S_i, i\in\{1,\ldots,k\},\\ (\Sigma,\Pi,\Theta)\vdash(W,S_i)\end{array}}{(\Sigma,\Pi,\Theta)\vdash(W,S)}\ (one-system)$$

$$\frac{\begin{array}{l}(\{C_1,\ldots,C_m,S_1,\ldots,S_k\}\Rightarrow S)\in\Theta\cup\Theta',\\ \text{for some } m, \text{where for each } C_i, i\in\{1,\ldots,m\},\\ \Pi(\sigma(W))=C_i \text{ or } \Pi'(\sigma(W))=C_i, \text{for some } \sigma\in(\Sigma\cup\Sigma'),\\ \text{and for some } k, \text{where for each } S_i, i\in\{1,\ldots,k\},\\ ((\Sigma,\Pi,\Theta)+(\Sigma',\Pi',\Theta'))\vdash(W,S_i)\end{array}}{((\Sigma,\Pi,\Theta)+(\Sigma',\Pi',\Theta'))\vdash(W,S)}\ (tu)$$

$$\frac{(\bigcup_{i=1}^{n}\Sigma_i,\bigcup_{i=1}^{n}\Pi_i,\bigcup_{i=1}^{n}\Theta_i)\vdash(W,S)}{Q\vdash(W,S)}\ (gtu)$$
where $n>1$ and $Q=(\Sigma_1,\Pi_1,\Theta_1)+\ldots+(\Sigma_n,\Pi_n,\Theta_n)$

$$\frac{\begin{array}{l}(\{C_1,\ldots,C_m,S_1,\ldots,S_k\}\Rightarrow S)\in\Theta\mid\Theta',\\ \text{for some } m, \text{where for each } C_i, i\in\{1,\ldots,m\},\\ \Pi(\sigma(W))=C_i \text{ or } \Pi'(\sigma(W))=C_i, \text{for some } \sigma\in(\Sigma\cup\Sigma'),\\ \text{and for some } k, \text{where for each } S_i, i\in\{1,\ldots,k\},\\ ((\Sigma,\Pi,\Theta)\ominus(\Sigma',\Pi',\Theta'))\vdash(W,S_i)\end{array}}{((\Sigma,\Pi,\Theta)\ominus(\Sigma',\Pi',\Theta'))\vdash(W,S)}\ (res)$$
where $\Theta\mid\Theta'=\{(\mathbf{S_1}\cup\mathbf{S_2}\Rightarrow S)\mid(\mathbf{S_1}\Rightarrow S)\in\Theta\wedge(\mathbf{S_2}\Rightarrow S)\in\Theta'\}$
$\cup\ \{(\mathbf{S_1}\Rightarrow S)\mid(\mathbf{S_1}\Rightarrow S)\in\Theta\wedge((\mathbf{S}\Rightarrow S)\notin\Theta', for\ any\ \mathbf{S})\}$
$\cup\ \{(\mathbf{S_2}\Rightarrow S)\mid(\mathbf{S_2}\Rightarrow S)\in\Theta'\wedge((\mathbf{S}\Rightarrow S)\notin\Theta, for\ any\ \mathbf{S})\}$

Figure 1. Overview of *Sigma* operators and associated rules - part 1.

In the rule for tight-union (tu), we can observe a tighter coupling of the three components of sensors, interpreter and situation reasoner compared to union, in that a tight-union composition behaves as an integrated system at all three levels (white-box integration), whereas the union employs multiple systems but each as a blackbox. While the rule specifies that the union of the corresponding components (union of sensors, union of interpreters and union of situation reasoners) are employed in recognizing a situation

$$\frac{(\bigcup_{i=1}^{n}\Sigma_i,\bigcup_{i=1}^{n}\Pi_i,|_{i=1}^{n}\Theta_i)\vdash(W,S)}{Q\vdash(W,S)}\ (gres)$$
where $n>1$ and $Q=(\Sigma_1,\Pi_1,\Theta_1)\ominus\ldots\ominus(\Sigma_n,\Pi_n,\Theta_n)$

$$\frac{\begin{array}{l}(\{C_1,\ldots,C_m,S_1,\ldots,S_k\}\Rightarrow S)\in\Theta\ \triangleleft\ \Theta',\\ \text{for some } m, \text{where for each } C_i, i\in\{1,\ldots,m\},\\ \Pi(\sigma(W))=C_i \text{ or } \Pi'(\sigma(W))=C_i, \text{for some } \sigma\in(\Sigma\cup\Sigma'),\\ \text{and for some } k, \text{where for each } S_i, i\in\{1,\ldots,k\},\\ ((\Sigma,\Pi,\Theta)\oslash(\Sigma',\Pi',\Theta'))\vdash(W,S_i)\end{array}}{((\Sigma,\Pi,\Theta)\oslash(\Sigma',\Pi',\Theta'))\vdash(W,S)}\ (ovr)$$
where $\Theta\ \triangleleft\ \Theta'=\{(\mathbf{S_1}\Rightarrow S)\mid(\mathbf{S_1}\Rightarrow S)\in\Theta\wedge(\mathbf{S_2}\Rightarrow S)\in\Theta'\}$
$\cup\ \{(\mathbf{S_1}\Rightarrow S)\mid(\mathbf{S_1}\Rightarrow S)\in\Theta\wedge((\mathbf{S}\Rightarrow S)\notin\Theta', for\ any\ \mathbf{S})\}$
$\cup\ \{(\mathbf{S_2}\Rightarrow S)\mid(\mathbf{S_2}\Rightarrow S)\in\Theta'\wedge((\mathbf{S}\Rightarrow S)\notin\Theta, for\ any\ \mathbf{S})\}$

$$\frac{(\bigcup_{i=1}^{n}\Sigma_i,\bigcup_{i=1}^{n}\Pi_i,\triangleleft_{i=1}^{n}\Theta_i)\vdash(W,S)}{Q\vdash(W,S)}\ (govr)$$
where $n>1$ and $Q=(\Sigma_1,\Pi_1,\Theta_1)\oslash\ldots\oslash(\Sigma_n,\Pi_n,\Theta_n)$

$$\frac{E\vdash(W,S)}{(E\oplus E')\vdash(W,S)}\ (union1)$$
$$\frac{E\nvdash(W,S)\ and\ E'\vdash(W,S)}{(E\oplus E')\vdash(W,S)}\ (union2)$$
$$\frac{E\vdash(W,S)\ and\ E'\vdash(W,S)}{(E\otimes E')\vdash(W,S)}\ (intersection)$$

Figure 2. Overview of *Sigma* operators and associated rules - part 2.

(or context), this happens conceptually, and does not necessarily imply the actual integration of software components. Tight-union has represents a type of interaction among two systems. The generalized form of tight-union to n-ary is given by the rule (gtu). Note that (gtu) with two operands is equivalent to (tu). Restriction represents another type of interaction among two systems, similar to tight-union, but where rules to infer the same situation must be joined in their conditions. The generalized form of restriction to n-ary is given by the rule $(gres)$. Note that $(gres)$ with two operands is equivalent to (res).

Overriding union represents another type of interaction among two systems, similar to tight-union, but where rules to infer the same situation have precedence. The generalized form of overriding union to n-ary is given by the rule $(govr)$. Note that $(govr)$ with two operands is equivalent to (ovr). Note that overriding union is left associative and non-commutative. The two rules of union rely on the success of one or the other - operationally, $(union1)$ would first be used for evaluations. If rule $(union1)$ succeeds, then $(union2)$ is not

needed. Note that costs are added up for evaluations. The rules above can be used in a backward-chaining fashion to determine if a particular context or situation currently holds. For example, $(R_1 + R_2) \vdash (W, S)$ determines if situation S holds in world W with respect to the composition $(R_1 + R_2)$, by pattern matching. The rules can also be used in as a query to ask what contexts or situations currently hold (or are happening), if S is a variable instead of a given situation. For example, $(R_1 + R_2) \vdash (W, S?)$ asks what situations may be happening with respect to $(R_1 + R_2)$, resulting in one or more possible instantiations for S? (we denote a variable by an identifier post-fixed by a "?").

A *composite context-aware system* is formed by using the above language of operators to compose context-aware systems.

Action Systems

We define another category of systems called *action systems* which takes situations or context recognized from context-aware system(s) and map those situations or context to actions. Similar to context-aware systems, we can define operators on action systems. An action system M is modelled simply as a relation between sets of context and situations, and actions, i.e. $M \subseteq \wp(\mathbf{S} \cup \mathbf{C} \cup \{\texttt{false}\}) \times \mathbf{A}$, where $\mathbf{A}$ is a set of actions.[10] The idea is that the context and situations are conditions that must be satisfied before the action is taken. For some action system M, $\mathbf{S} \in \wp(\mathbf{S} \cup \mathbf{C})$, and $a \in \mathbf{A}$, we will write mappings of the form $(\mathbf{S}, a) \in M$ as $(\mathbf{S} \Rightarrow a) \in M$, for readability. We will also often use the constant `false` to represent a condition that can never be satisfied. Actions with the `false` condition will never be performed - this will be useful in compositions to inhibit behaviours as we show later.

Sigma Operators on Action Systems

We also have Sigma expressions on collections of action systems. We define four operators on action systems: *a-union* (denoted by "∪"), *a-intersection* (denoted by "∩"), *a-restriction* (denoted by "|"), and *a-overriding* (denoted by "◃"), as follows. Given two action systems M_1 and M_2, $M_1 \cup M_2$ is simplify the set-theoretic union of the corresponding relations, and the intersection $M_1 \cap M_2$ is the set-theoretic intersection of the corresponding relations. The purpose of a-restriction is to cater for cases where different situations or contexts map to the same actions, and the intention is to aggregate (or "and"-ing) the situations or contexts rather than simply "or"-ing them. For example, suppose we have $M_1 = \{(\{s_1, s_2\} \Rightarrow a_1), (\{s_6\} \Rightarrow a_3)\}$ and $M_2 = \{(\{s_3, s_4\} \Rightarrow a_1), (\{s_5\} \Rightarrow a_2)\}$. Then, forming the union gives $\{(\{s_1, s_2\} \Rightarrow a_1), (\{s_3, s_4\} \Rightarrow a_1), (\{s_5\} \Rightarrow a_2)\}$, which means that action a_1 can be triggered either by $\{s_1, s_2\}$ or $\{s_3, s_4\}$. However, suppose we wish to state that a_1 is to be triggered by $\{s_1, s_2, s_3, s_4\}$, or $s_1 \wedge s_2 \wedge s_3 \wedge s_4$. We do this by using *a-restriction*, which is defined as follows:

$$\begin{aligned} M_1 \mid M_2 = \; & \{(\mathbf{S_1} \cup \mathbf{S_2} \Rightarrow a) \mid (\mathbf{S_1} \Rightarrow a) \in M_1 \wedge (\mathbf{S_2} \Rightarrow a) \in M_2\} \\ & \cup \{(\mathbf{S_1} \Rightarrow a) \mid (\mathbf{S_1} \Rightarrow a) \in M_1 \wedge ((\mathbf{S_2} \Rightarrow a) \notin M_2, \mathit{for\ any}\ \mathbf{S_2})\} \\ & \cup \{(\mathbf{S_2} \Rightarrow a) \mid (\mathbf{S_2} \Rightarrow a) \in M_2 \wedge ((\mathbf{S_1} \Rightarrow a) \notin M_1, \mathit{for\ any}\ \mathbf{S_1})\} \end{aligned}$$

Then, when $M_1 = \{(\{s_1, s_2\} \Rightarrow a_1), (\{s_6\} \Rightarrow a_3)\}$ and $M_2 = \{(\{s_3, s_4\} \Rightarrow a_1), (\{s_5\} \Rightarrow a_2)\}$, we have

$$M_1 | M_2 = \{(\{s_1, s_2, s_3, s_4\} \Rightarrow a_1), (\{s_6\} \Rightarrow a_3), (\{s_5\} \Rightarrow a_2)\}$$

The *a-restriction* operator allows a newly added system to possibly inhibit an existing system since it adds new conditions for triggering an action, and is similar to output inhibition from Brook's subsumption architecture [13], as we illustrate below.

The operator *a-overriding* allows the rules of one system to override those of another, but only with regards to actions found in both systems, i.e. suppose M_1 has precedence over M_2, then

$$\begin{aligned} M_1 \triangleleft M_2 = \; & \{(\mathbf{S_1} \Rightarrow a) \mid (\mathbf{S_1} \Rightarrow a) \in M_1 \wedge (\mathbf{S_2} \Rightarrow a) \in M_2\} \\ & \cup \{(\mathbf{S_1} \Rightarrow a) \mid (\mathbf{S_1} \Rightarrow a) \in M_1 \wedge ((\mathbf{S_2} \Rightarrow a) \notin M_2, \mathit{for\ any}\ \mathbf{S_2})\} \\ & \cup \{(\mathbf{S_2} \Rightarrow a) \mid (\mathbf{S_2} \Rightarrow a) \in M_2 \wedge ((\mathbf{S_1} \Rightarrow a) \notin M_1, \mathit{for\ any}\ \mathbf{S_1})\} \end{aligned}$$

Such an operator allows a newly added action system to override the behaviour of a previous system.

A *composite action system* is formed as a composition of action systems using the above operators (e.g., $(M_1 \cup M_2) \cap M_3$).

Context-Aware Action Systems

A *context-aware action system* is then a pair (E, M), comprising a (possibly composite) context-aware system E and an action system M (which also may be composite). But to be meaningful, the situations which trigger actions in M should be infer-able by E, for otherwise, certain actions in M will never be triggered. Hence, for a **valid** context-aware action system, at minimum, we also require that for any $(\mathbf{S} \Rightarrow a) \in M$, for any situation $S \in \mathbf{S}$, we have $E \vdash (W, S)$, for some conceivable W, i.e., as long as there is the right sensor readings (with the judgement of the system designer). We call this the *validation property*.

Expressions formed using the Sigma operators are called Sigma expressions, and corresponds to some composite system.

[10] We use relation rather than function for greater generality, allowing non-determinism but do not discuss specifically how to deal with non-determinism here.

Scenarios of Using Sigma: Thing-Ensembles and Sigma expressions

In this section, we illustrate the use of our formalism to model a range of systems from different domains. In each example, we first identify component context-ware systems and action systems, and then describe how they can be composed to form the desired system.

An Illustrative Example: Triggering the Television

We consider an example now of using our operators to incrementally build a system. The system we intend to build is an automatic turn TV on system, where a user *Seng* sits down on an armchair (in front of a television) and the television is automatically turned on.

We start with a basic version of the system which has sensors in the armchair to detect Seng's weight and then infer the situation that Seng is on the chair, afterwhich this is mapped to the action of turning on the television.

We can describe this system, calling it $P = (R_p, M)$, where R_p is the triple $(\{weight\_sensor\}, \Pi_p, \Theta_p)$, where

$$('reading(weight\_sensor) = 65' \Rightarrow\ seng\_on\_chair) \in \Pi_p$$

that is, if the reading on the chair's weight sensor is 65, we interpret this to mean Seng is sitting on the chair, and

$$\Theta_p = \{(\{seng\_on\_chair\} \Rightarrow seng\_ready\_for\_tv)\}$$

which states that Seng sitting on that chair implies he is ready to watch television. And M only has one rule which maps Seng ready for television to the action of turning on the television, i.e.

$$M = \{(\{seng\_ready\_for\_tv\} \Rightarrow turn\_on\_tv)\}$$

Figure 3 depicts this system.

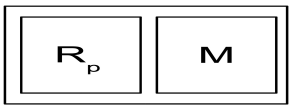

Figure 3. A context-aware action system for the basic version.

Now, there are various shortcomings to such a basic system, namely, when someone else sits on the chair with the same weight as Seng. To further confirm Seng's presence, we can attach an RFID system to the chair that can identify Seng (assume to have an ID of "0101", for example sake) as being nearby. We define this system as $R_q = (\{RFID\_sensor\}, \Pi_q, \Theta_q\})$, where

$$('reading(RFID\_sensor) = 0101' \Rightarrow\ seng\_detected) \in \Pi_q$$

and

$$\Theta_q = \{(\{seng\_detected\} \Rightarrow seng\_ready\_for\_tv)\}$$

which states that Seng detected by this RFID reader implies he is ready to watch television.

Then, we can form the new composite system $R_p \ominus R_q$, using rule (*gres*), so that:

$$R_p \ominus R_q = (\{weight\_sensor, RFID\_sensor\}, \Pi_p \cup \Pi_q, \Theta_p \mid \Theta_q)$$

where

$$\Theta_p|\Theta_q = \{(\{seng\_on\_chair, seng\_detected\} \Rightarrow seng\_ready\_for\_tv)\}$$

Figure 4 depicts the extended context-aware action system $(R_p \ominus R_q, M)$. If we instead wanted to model suppression, i.e., to have the rules of R_q precede over R_p, we can use $R_q \oslash R_p$ which uses only the rule in Θ_q and ignores that in Θ_p.

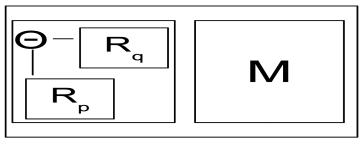

Figure 4. A context-aware action system for the extended version.

Now suppose, a friend of mine invented a head and eye (i.e., gaze) tracking system which can detect if a person is actually looking at the television and determines how long the person does so continuously, and the idea is that a person expresses the intention to watch television by staring at the television continuously for at least 4 seconds.

We provide an abstract description of the system as the following context-aware system $R_r = (\{gaze\_sensor, tv\_sensor\}, \Pi_r, \Theta_r)$, where

$$\{('reading(gaze\_sensor) = 4' \Rightarrow\ person\_gazing\_tv),\ ('reading(tv\_sensor) = 0' \Rightarrow\ tv\_off)\} \in \Pi_r$$

and

$$\Theta_r = \{(\{person\_gazing\_tv, tv\_off\} \Rightarrow person\_want\_tv\_on)\}$$

We also have the action system on the television that uses this trigger M' given by:

$$M' = \{(\{person\_want\_tv\_on\} \Rightarrow turn\_on\_tv)\}$$

Now, considering that it would be useful to combine M' with M in order to better regulate how the television is switched on, we use the operator *a-restriction*, forming

$$M \mid M' = \{(\{seng\_ready\_for\_tv, person\_want\_tv\_on\} \Rightarrow turn\_on\_tv)\}$$

Figure 5 depicts the extended (valid) context-aware action system

$$((R_p \ominus R_q) \oplus R_r, M \mid M')$$

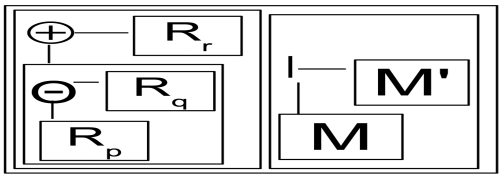

Figure 5. A context-aware action system for the extended (valid) version.

For testing purposes, perhaps we may want to disable M. We can then use *a-overriding* in the composition $M' \triangleleft M$, which only takes the rule in M' into account. Or if we wanted to "or" the triggers for the turning on the television, we would use *a-union*

$$((R_p \ominus R_q) \oplus R_r, M \cup M')$$

A key idea of our operator-based approach is not only that different semantics of composite systems can be easily stated using different expressions, thereby clarifying their differences, but also, that our approach enables incremental extensions to an existing system while preserving the existing systems. We can define a composite system without altering the definitions of its constituent systems. The advantage is, of course, that different compositions can be formed from the same constituent systems.

We an use the *overriding union* on context-aware systems to have a new system suppress the rules of another, or use *a-overriding* or *a-restriction* to effectively inhibit (depending on sensor readings) the action triggers of a system.

Our approach also separates the context-aware system from the action system, in order to provide greater modularity, requiring only that the validation property be satisfied when combining a context-aware system with an action system.

Modelling Sensor Networks

We can use our formalism to model sensor networks, with sensors only having context interpretation, i.e. of the form $(\Sigma, \Pi, \emptyset)$.

Consider a sensor network comprising a collection of nodes (each with a camera and temperature sensor) distributed throughout the fringe of a neighbourhood, the purpose of which is to detect approaching bush fires. We also assume each node has a position.

A node N_i, at position p_i, denoted by $N_i.p_i$ is represented by a context-aware system of the form

$$(\{camera, temp\_sensor\}, \Pi_i, \emptyset)\})$$

where

$$\{('reading(camera) = flame' \Rightarrow fire\_seen),$$
$$('reading(temp\_sensor) \geq 70' \Rightarrow hot)\} \subseteq \Pi_i$$

Each node would have its own *camera* and *temp_sensor* though notationally, we don't distinguish among them (if need be, we could use $camera_i$ and $temp\_sensor_i$). Assuming a sensor network $\mathbf{N}$ of n nodes distributed at different locations, we can represent this by the composition $\oplus_{i=1}^{n} N_i(p_i)$ (or we write this as $\oplus\mathbf{N}$, where $\mathbf{N} = \{N_i \mid i \in \{1, \ldots, n\}\}$). And we could pose queries such as $\mathbf{N} \vdash fire\_seen$? which will return true if fire is seen at any one location. A query such as $(N_2 \oplus N_3 \oplus N_6) \vdash fire\_seen$? asks if a fire is seen at nodes 2, 3 or 6. Effectively, we can use our formalism to pose queries to selected sensors or to different parts of the sensor network or modify the semantics slightly to return the node where fire is seen.

However, over the same physical sensor network, one could define different compositions, relating to different possible "behaviours" being sought. For example, what would the composition $\otimes\mathbf{N}$ mean? Physically, the sensor network is not changed, but $\otimes\mathbf{N} \vdash fire\_seen$ is true when fire is seen in all locations.

Extending nodes and the sensor network.

Now, let us consider adding m sensor nodes to the sensor network for nodes N_{n+1} to N_{n+m}. The above compositions then extend to the $n + m$ nodes easily. Let $\mathbf{M} = \{N_i \mid i \in \{m+1, \ldots, n+m\}\}$. Then, determining if

$$((\otimes\mathbf{N}) \otimes (\oplus\mathbf{M})) \vdash fire\_seen$$

is true asks if fire is seen in all of the n nodes and in any of the new m nodes.

We can add a fire inference engine to each node F_i, denoted by $(\emptyset, \emptyset, \Theta_i)$, where Θ_i has only one rule that aggregates the context to infer the situation that a fire occurs:

$$\Theta_i = \{(\{fire\_seen, hot\} \Rightarrow fire)\}$$

Such a component F_i might correspond to a physical component (later) attached to each of the sensors.

Then, let $NF_i = N_i + F_i$ represent an extended node, $NF_i \vdash fire$ is true when for node i, $fire$ is inferred. A composition such as $\oplus\mathbf{NF} \vdash fire$ denotes if this is true for any node NF_i in the set $\mathbf{NF}$.

Now, we want to add smoke sensor nodes with an inference component, denoted by N_i', where

$$N_i' = (\{smoke\_detector\}, \Pi_i', \Theta_i')\})$$

where

$$\{('reading(smoke\_detector) = yes' \Rightarrow smoke\_present),$$
$$('reading(smoke\_detector) = no' \Rightarrow smoke\_absent)\} \subseteq \Pi_i'$$

and

$$\Theta_i' = \{(\{smoke\_present\} \Rightarrow fire)\}$$

And a network of smoke sensor nodes comprising a set of k such nodes, denoted by $\mathbf{S}$, can be distributed in the same areas as the earlier $n + m$ nodes.

A query on the composition such as $(\oplus\mathbf{S}) \otimes (\oplus\mathbf{NF})) \vdash fire$ is true if both the smoke sensor network and the previous sensor network confirms there is a fire. But the composition does not take into account the locations of nodes explicitly.

Composing a NF_i node with a smoke detector node N_j' can be represented, such as $N_{ij} = NF_i \ominus N_j'$ which denotes a node $N_{ij} = (\Sigma_{ij}, \Pi_{ij}, \Theta_{ij})$, where:

$$\Sigma_{ij} = \{camera, temp\_sensor, smoke\_detector\}$$

and

$$\{('reading(smoke\_detector) = yes' \Rightarrow smoke\_present),$$
$$('reading(smoke\_detector) = no' \Rightarrow smoke\_absent),$$
$$('reading(camera) = flame' \Rightarrow fire\_seen),$$
$$('reading(temp\_sensor) \geq 70' \Rightarrow hot)\} \subseteq \Pi_{ij}$$

and

$$\Theta_{ij} = \{(\{fire\_seen, hot, smoke\_present\} \Rightarrow fire)\}$$

Note that N_{ij} might comprise two physically separated nodes NF_i and $N'j$ but abstractly, it can be represented as (conceptually) one node.

Context Attributes for Context-Aware Systems. If we associate a set of context attributes (e.g., location, power available, etc) to each node, denoted using the notation $< node > . < property >$, and form predicates over these properties, we an define sets of nodes.

For example, the following is the set $\mathbf{P}$ of extended nodes within distance r of *here* with high power levels:

$$\mathbf{P} = \{NF_i \mid NF_i.power = high \wedge dist(NF_i.location, here) \leq r \wedge i \in \{1, \ldots, n+m\}\}$$

We can then define compositions on these notes such as $\oplus\mathbf{P}$. We can also define a set of high powered smoke sensor nodes in the same area:

$$\mathbf{P}' = \{N_i' \mid N_i'.power = high \wedge dist(N_i'.location, here) \leq r \wedge i \in \{1, \ldots, k\}\}$$

A query on the composition such as $(\oplus\mathbf{P}') \otimes (\oplus\mathbf{P})) \vdash fire$ is true if both P and P' confirms there is a fire in the same area (but only as defined by the distance r from *here*). Finer grained areas can, of course, be defined such as a band given by $r' \leq dist(< node > .location, here) \leq r$, for some distance $r' \leq r$.

Summary. What we have shown here are expressions to represent extensions of a sensor network in different ways as well as to represent abstract sensor nodes (which might be physically composed from two or more nodes). They serve as a notation to formally and precisely represent different sensor networks (or different parts of a sensor network) but yet can provide a basis for operational meaning of queries.

Tiling Cooperative Digital Picture Frames

We consider modelling a collection of digital picture frames on a wall; we assume that each digital picture frame has a unique identifier and can sense if another digital picture frame is near it, in four directions: above it, to the left of it, to the right of it and below it. Then, according to what each picture frame thinks of its relative position, each picture frame then show a particular picture. We assume that it is possible to change the position of the frames - that they are fastened to the wall by the owner and they can be repositioned.

Each digital picture frame F_i (assuming $1 \leq i \leq 9$) is modelled as a context-aware action system with a frame sensor (for sensing another frame in four directions which returns a subset of $\{above, below, left, right\}$ depending on what is sensed) of the form $F_i = (R_i, M_i)$, where $R_i = (\Sigma_i, \Pi_i, \Theta_i)$ with

$$\Sigma_i = \{frame\_sensor_i\}$$

and

$$\{('above \in reading(frame\_sensor_i)' \Rightarrow frame\_above_i),$$
$$('below \in reading(frame\_sensor_i)' \Rightarrow frame\_below_i),$$
$$('left \in reading(frame\_sensor_i)' \Rightarrow frame\_left_i),$$
$$('right \in reading(frame\_sensor_i)' \Rightarrow frame\_right_i),$$
$$('above \notin reading(frame\_sensor_i)' \Rightarrow \neg frame\_above_i),$$
$$('below \notin reading(frame\_sensor_i)' \Rightarrow \neg frame\_below_i),$$
$$('left \notin reading(frame\_sensor_i)' \Rightarrow \neg frame\_left_i),$$
$$('right \notin reading(frame\_sensor_i)' \Rightarrow \neg frame\_right_i)\} \in \Pi_i$$

and

$$\Theta_i = \{\ (\{frame\_left_i, frame\_right_i\} \Rightarrow hor\_linear_i),$$
$$(\{\neg frame\_below_i\} \Rightarrow bottom\_end_i),$$
$$(\{\neg frame\_above_i\} \Rightarrow top\_end_i),$$
$$(\{\neg frame\_right_i\} \Rightarrow right\_end_i),$$
$$(\{\neg frame\_left_i\} \Rightarrow left\_end_i),$$
$$(\{frame\_above_i, frame\_below_i\} \Rightarrow ver\_linear_i)\ \}$$

and M_i is of the form:

$$\{(\{hor\_linear_i\} \Rightarrow show\_hline_i), (\{ver\_linear_i\} \Rightarrow show\_vline_i),$$
$$(\{bottom\_end_i\} \Rightarrow show\_bottom\_cap_i), (\{top\_end_i\} \Rightarrow show\_top\_cap_i),$$
$$(\{right\_end_i\} \Rightarrow show\_right\_cap_i), (\{left\_end_i\} \Rightarrow show\_left\_cap_i)\}$$

Note that a frame might show multiple caps at the same time depending on what frames it (not) detects, e.g., a frame that is right ended and left ended at the same time might show both a right cap and a left cap at the same time. Each frame works independently and what emerges is the overall pattern. The frames do not need to communicate with each other, but merely to sense each other's proximity.

Figure 6 gives the idea of how the rules would result in the different patterns being displayed.

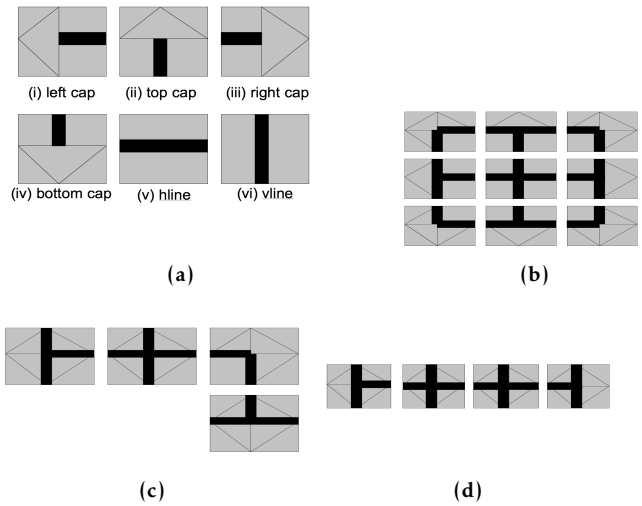

Figure 6. Patterns from rules. (a) shows each of the six display objects.
(b) shows a 3x3 configuration of nine frames and the pattern displayed according to the above rules.
(c) shows the pattern for an L-shaped configuration of four frames according to the rules above.
(d) shows the pattern for a horizontal configuration of four frames.

Next we add a particular frame that can gather information from all other frames (assuming four frames) and so detect the overall arrangement of all the frames (including itself), including horizontally linear, and vertically linear. The frame is denoted by $F' = (R', M')$, where $R' = (\emptyset, \emptyset, \Theta')$, where Θ' contains 12 rules as follows

$$(\{hor\_linear_i, right\_end_j, left\_end_k, hor\_linear_l\} \Rightarrow config\_horizontal) \in \Theta'$$
for $i, j, k, l \in \{1, 2, 3, 4\}$, i, j, k, l all different

since any of the four frames can be in the situation hor_linear, $right\_end$ or $left\_end$ at a time, and at any time, when the frames are lined up horizontal, two of them will be hor_linear and one $right\_end$ and another $left\_end$, and for vertical configuration, another 12 rules as follows

$$(\{ver\_linear_i, bottom\_end_j, top\_end_k, ver\_linear_l\} \Rightarrow config\_vertical) \in \Theta'$$
for $i, j, k, l \in \{1, 2, 3, 4\}$, i, j, k, l all different

And M' then voices out which configuration the system detects as follows:

$$\{(\{config\_vertical\} \Rightarrow say\_vertical),$$
$$(\{config\_horizontal\} \Rightarrow say\_horizontal)\}$$

The combined valid context-aware action system comprising components from F_1, F_2, F_3, F_4 and F' is of the form

$$(R_1 + R_2 + R_3 + R_4 + R', M_1 \cup M_2 \cup M3 \cup M_4 \cup M')$$

Each M_i does its action as before but M' also utilizes "inputs" from the R_is and R' to perform actions. Here, the idea is that F' can be added to the F_is without the F_is needing to change.

For aesthetic reasons, one might want to add new digital frames to the system, that will alter (and subtract from) the behaviour of an existing composite system. For example, suppose we add new "inhibitor" frames that will prevent certain shapes from ever being displayed. Consider a frame $F^i_{inh} = ((\emptyset, \emptyset, \emptyset), M^i_{inh-hline})$ with no context-aware system but has an action system $M^i_{inh-hline}$ which is an inhibitor for frame F_i, preventing the frame from ever showing a horizontal line, of the form:

$$\{(\{\texttt{false}\} \Rightarrow show\_hline_i)\}$$

So, the system $(R_i, M_i \mid M^i_{inh-hline})$ has a rule of the form

$$\{(\{hor\_linear_i, \texttt{false}\} \Rightarrow show\_hline_i)\}$$

which will never be satisfied, and so, the $show\_hline_i$ action will never be performed regardless of the what sensor readings are received.

Incidentally, new tailored operators on frames can be defined based on the Sigma operators. For example, we can define an operator $\odot$ on frames with the semantics:

$$F_1 \odot F_2 = (R_1 \oplus R_2, M_1 \mid M_2)$$

where $F_1 = (R_1, M_1)$ and $F_2 = (R_2, M_2)$.

Then, we have:

$$F_i \odot F^i_{inh} = (R_i \oplus (\emptyset, \emptyset, \emptyset), M_i \mid M^i_{inh-hline})$$

forming the (virtual) frame (or pair of frames) which has restricted display compared to F_i alone.

Smarter Cars and Urban Awareness

There has been interesting developments in using sensors to monitor city and urban environments. The MIT SENSEable Real Time Rome project[11] used cellphones and GPS devices to discover movements patterns of people and transportation systems in Rome, enabling usage of streets and neighbourhoods to be tracked and visualized. The more recent SENSEable CURRENTcity[12] project is working on visualizing the real-time dynamic behaviours of cities including traffic jams and gatherings. The Cityware initiative[13] has a project to monitor the paths of tourists using GPS tracking.

There is also other work towards monitoring, in real-time, traffic behaviour to provide, in advance, information about traffic jams on roads. The notion of crowd sensing and understanding events in the city provides insight into life in the city for its inhabitants and tourists. There is also work towards supporting place-based awareness leading towards what we call "legible places", places that can be "read" and "queried" like a book.

Modern cars will be equipped increasingly with sensors from rain sensors, pedestrian sensing, adaptive cruise control to parking sensors. Pedestrians can carry sensors which cars can pick up to ensure that cars don't hit them. Cars can sense and detect approaching emergency vehicles and react accordingly. Accurate GPS (up to the metre) can help cars in manouvres such as lane change and overtaking. Cars could become aware of nearby parking slots and busy intersections. There is a need for context-aware action systems to work together, the car (as a whole if viewed as a context-aware action system, with its subsystems), and the sensors in the environment of the car.

We illustrate a scenario of car using sensors in the urban environment to become better aware of where vehicular and people crowds are. We imagine a future urban environment which has sensors to detect crowds of people and heavy traffic streets. And the car is able to detect these sensors, connect to them and acquire information about crowds (of people or vehicles) within a preset radius from the current position of the car. Such information might then be displayed on the car's computer on a map indicating nearby crowds. Let this capability of a context-aware car be represented as an action system with a rule that gets the crowd at a position pos and displays that, represented by $M = \{(\{crowd\_at(pos)\}, display\_crowd(pos))\}$. A crowd sensor embedded in the urban environment is represented by context-aware systems of the form

$$R = (\{crowd\_sensor\}, \{('reading(crowd\_sensor) = pos' \Rightarrow crowd\_at(pos))\}, \emptyset)$$

And the (R, M) represents the overall (distributed) system, comprising a component on the car and a component in the stationary urban environment. When the car detects another crowd sensing system in the infrastructure in the vicinity, say R' in the same form as R, it can employ this system, whether to confirm the crowd position, i.e. we have $(R \otimes R', M)$, or to provide new crowd positions $(R \oplus R', M)$. The car itself might have sensors, represented by the context-aware system R'' which can be combined with infrastructure sensors: $(R \oplus R' \oplus R'', M)$.

Meta-Programming with Sigma Expressions. It is possible to embed Sigma expressions into a programming language, in order to form new compositions at run-time, especially when the set of available context-aware systems might change over time, e.g., upon discovery of new systems.

With the smart car example earlier, consider a simple event-driven program that can discover the presence of context-aware systems in the vicinity, and recomputes a composition based on discovered context-aware systems. Let R'' be the car's own context-aware system, and M, the car's action system. We also introduce an operator that takes a context-aware action system E, and executes it (i.e., gets the sensor readings, performs reasoning, and determines if any action should be taken, and executes the action), denoted by **execute**$[E]$. Note that there are two ways that Sigma expressions can be executed, in the forward-chaining (data-driven) style as in **execute**$[E]$, or using a backward-chaining query style where, given a situation, determine if the situation is occurring of the form $E \vdash S?$.

loop
  initiate discovery of context-aware systems
  $\mathbf{R_d}$:= discovered context-aware systems
  execute$[(\oplus \mathbf{R_d}) \oplus R'', M)]$

[11] http://senseable.mit.edu/realtimerome/
[12] http://www.currentcity.org/
[13] http://www.cityware.org.uk; see Digital Footprints project.

end

Note that the above algorithm works with the just-discovered context-aware systems in the car's vicinity; as the car moves, a different set $\mathbf{R}_d$ of such systems will be discovered.

Modular Robotics

Modular robotics involves building a larger robot from the cooperation of a set of individual robots. The Modular Robotics company[14] have building blocks which are classified into *sense blocks*, *think/utility blocks* and *action blocks*, where sense blocks can sense light, temperature, and distance from other objects, think/utility blocks can map sensor inputs to actions according to specific relationships (e.g., inverse, maximum, minimum, etc), and action blocks performs actions based on sensor inputs. For example, with an *Inverse* thin/utility block between the Light Sensor and the Drive sensor blocks, the robot drives slower when the light gets brighter. Infrared ports are used by robots to connect to neighbouring robots and sense distance from to nearby objects.

Inspired by this approach, we represent a simple system of modular robots, comprising context-aware systems and action systems. Let sense blocks be denoted by context-aware systems of the form $R_s = (\Sigma, \emptyset, \emptyset)$. Let think blocks be denoted by context-aware systems of the form $R_t = (\emptyset, \Pi, \Theta)$. Let action blocks be denoted by action systems M.

Given the above blocks, we can form a configuration of robots, represented by the composition $(R_s + R_t, M)$. x sensor blocks, y think blocks and z action blocks can be represented by compositions of the form

$$((R_s^1 + \ldots + R_s^x) + (R_t^1 + \ldots + R_t^y), M_1 \cup \ldots \cup M_z)$$

But if each block has up to six interfaces, with $z = 1$ and other blocks attached to it, we have $x + y \leq 6$.

The work in [44] mentions the notion of roles which map to sets of behaviours. A robot/block decides what role to take up by sensing its context. Depending on its role assumed, the robot then behaves in a certain predefined way. Such roles can be encoded as a set of rules in an action system.

Related Work

The Internet of Things has caught the attention of the world, and involves the development of suitable hardware, software and paradigms where everyday objects can be linked to the Internet, can be sensed so that data about them can be obtained, and where the things can be composed in useful ways [41, 47], e.g. sensors used in building smart cities [14, 58]. However, recent IoT visions aim to go further, including linking smart things with social networks and embedding their behaviours within the context of social interactions [40], and adding a layer of data processing so that intelligent perception and behaviours can be harnessed from the Internet of Things [55]. This paper focuses on a different, yet complementary, vision for the IoT, where IoT systems or thing-ensembles can be built in an incremental and compositional manner, e.g.,

- sensing capability is increased compared to the previous system with the addition of new sensors, in that what can be sensed previously can still be sensed, and also new aspects can be sensed due to the new sensors;
- action capability is increased with the addition of new actuators so that, what can be done previously is either enhanced or that more actions can be taken, or better actions can now be taken, apart from preservation of some previous actions; and
- "intelligence" or reasoning capability can be increased so that what was previously possible to reason about is still retained but new rules added allow further inferences to be made (even from the same sensor system).

Hence, while IoT work is expanding rapidly, our work focuses on the above aspects for future IoT systems. We focus on reasoning with the components of a system rather than with the data from IoT systems as in other work such as [37, 51].

There are numerous middleware, frameworks and toolkits for building context-aware systems in ubiquitous computing environments, as surveyed in [6, 19, 27, 33]. However, our formalism aims at a programming language based approach, combined with an operator-based formalism; the operators theoretically define how the constituents of two context-aware systems (or action systems) should work together, but does not prescribe how such integration can be carried out; the operators could be implemented in different ways: Web services could be used if the systems are distributed or proprietary protocols might be employed in the case of digital appliances in the home. We aim towards a high level specification language. In [39] is described a framework where integration of heterogeneous legacy systems and reusing components are helpful for incrementally constructing global smart spaces.

Among work on software engineering paradigms and models for building context-aware systems, there has been investigation of suitable abstractions. For example, the sentient object model [8] encapsulates the key features of a context-aware system comprising

[14] See also http://www.modrobotics.com

sensor capture and context-reasoning components. A context-aware system in our model can be mapped to a sentient object in their model, and they do not provide composition operators as we do here. A survey of programming languages supporting the notion of context-oriented programming, as extensions to object-oriented programming, is given in [3]; such languages provide mechanisms to represent context as an explicit concept in the language, and to model context-aware execution of programs (or program fragments) within a software system, especially when they cut across objects. In several of these context-oriented programming languages [15], context is represented in the programs as layers, and during program execution, such layers can be activated or deactivated. Context propagators [5] has been proposed to capture the idea of adaptations having dependencies on each other, and such dependencies are themselves dependent on context. The notions of isotope and context block that represents conditions for execution of isotope elements are proposed in [42]. In [29], a context-aware application is programmed using an abstraction called activity - policies are used to specify context-aware behaviour and resources. The work in [20] uses the notion of context graphs to define the context-aware behaviour of applications, including the transitions among states of the application, and what behaviours are triggered in different situations. Our approach does not represent states of the participating systems but only how they should work together. Model-driven approaches have been investigated for with CAMEL (Context-Awareness Modelling Language) [46], where context-sensing and adaptation to context can be easily represented. In summary, different abstractions have been proposed to represent context-aware behaviours, many embedded within an existing programming language and often also used in designing the applications themselves. However, they have not investigated a compositional operator approach as we do here.

Due to the popularity of context-awareness in pervasive/ubiquitous computing, there has arisen formal systems to model context-aware behaviour. Notable are those based on process calculi approaches, such as the Context-Aware Calculus [59] which has similarities with the ambient calculus (which models processes with spatial notions), the calculus of context-aware ambients which builds on the ambient calculus [45], CONAWA which also builds on ambient calculus ideas with a richer modelling of context with several ambient-like trees [28], a calculi about contextual reactive systems [10, 11], and a bigraphical model of context-aware systems [9]. There has also been work on using the ambient calculus as a basis for a programming paradigm [53]. However, our approach is not based on process calculi and we adopt a software engineering style approach of combining components structurally, rather than combining process descriptions of run-time behaviours as in the context-aware calculi approaches. Then, in our approach, the behaviour of composed components is then determined by following the operational semantics rules and depends on the actual content within the systems. Also, our approach is at a different level of abstraction - details of networking and communication are abstracted away within the semantics of each operator; we only define conceptually how the system should cooperate (a mode of cooperation represented by the meaning of an operator), and leave the protocol details to the implementation. Our approach does not replace but could complement such process calculi models of context-aware behaviour; we use our formalism for structural compositions, and then context-aware calculi could then be used to model what happens at run-time within a composition. Given descriptions of constituent systems in a thing-ensemble, investigating translation of a (a Sigma expression) into an initial context-aware calculi expression and then reasoning about run-time behaviour in the calculi could perhaps be an avenue for future work.

Ideas from component based software engineering (e.g., [23]), service composition (e.g., [18]) and modular logic programming (e.g., [12]), share similarities with our algebraic approach, and in fact, we draw inspiration from operators for combining logic programs in our work; however, the entities we compose are not traditional software components, services, programs but triples representing context-aware systems and relations representing action systems.

The actor model [1] formalizes concurrent computations with the key notion of the actor, and multiagent systems research (e.g., [54]) have formalized representations of software entities that are autonomous, proactive and respond to environmentally sensed inputs. However, the resemblance to our notion of context-aware systems is superficial, given their emphasis on proactive behaviours, and their machinery for that purpose, and our operator language is unique in the way that the constituent contents of systems are composed, rather than multiagent cooperation ideas. While one can consider modelling the context-aware action systems in our examples as agents, and then model their cooperation protocols, and allow cooperation to emerge at run-time (and it would be interesting future work to do so), in our approach, the cooperation among context-aware systems is essentially scripted out (using the Sigma operators), and we adopt a global picture view (from the programmer's perspective) rather than a decentralized view, with the operators abstracting away protocol details.

The work by Beal[15] developed Proto, a functional language where rules can be programmed that describe the behaviour of a collection of nodes spread out across space. Our work is different in that it focuses on a small set of operators for combining and reasoning with thing-ensembles (which may or may not be distributed spatially).

Conclusion and Future Work

We have taken a first step towards providing a language that can serve two purposes, given a set of context-aware things that can communicate with each other, i.e. a thing-ensemble, we can: (i) specify compositions of context-aware things at design-time, and (ii) script ways in which context-aware things should work together at run-time. We built on the design philosophy and operator language first introduced in [36], but here, giving a thorough treatment with new operators and a range of examples to illustrate the versatility of our approach.

Based on the theoretical basis in this paper, future work includes providing design and programming tools based on Sigma, in order to demonstrate the feasibility of our approach. While demonstrating the applicability of Sigma in a wide range of applications, we have not yet dealt with examples of greater complexity and of a much larger scale. For example, an entire smart house functionality could be modelled with our approach.

We have not discussed implementation details in depth in this paper. There are two paradigms for implementation when Sigma expressions are used for scripting behaviours of things:

- *orchestration*: we can use a central coordinator which will interpret Sigma expressions and when contents of a particular system/thing needs to be used, the coordinator can contact the system (e.g., via Web services if the system provide a Web service interface); essentially, most of the communication is between coordinator and systems, and the coordinator consults the appropriate systems as determined by the operators in the expression.

  As an example, we sketch a strategy to implement a system comprising situation/context-aware recognition and actions via an example given earlier in Figure 5, corresponding to the expression:

  $$((R_p \ominus R_q) \oplus R_r, M \mid M')$$

  To implement a system described by the above expression, the appropriate sensors as defined in R_p, R_q and R_r, are needed, namely, the weight sensor on the chair, the RFID sensor to detect $Seng$ and the gaze sensor worn by $Seng$. Computation then proceeds in a (sensor) data-driven fashion by converting the operational rules given earlier into logic programming rules [34]. The readings from the sensors are fed into the rules defining the situation-recognition behaviours in R_p, R_q and R_r to determine if any of the action-triggering situations in $M \mid M'$ are occurring and if occurring, appropriate actions are taken. Backward chaining as illustrated in [34] can be employed, i.e. to determine if any of the situations triggering an action as defined in $M \mid M'$ is occurring, we first do backward chain reasoning on the rules, which then eventually results in interrogating the sensors. The sensors are then queried for their readings and if a situation-triggering action is occurring, the appropriate actions as defined by $M \mid M'$ are taken, and if not, no action is taken. When the sensor readings change beyond a preset threshold, a similar chain of reasoning is triggered to see if any action is now to be taken. Future work will consider efficient implementations of this idea.

- *choreography*: we can use a (mostly) peer-to-peer interaction model where one of the peers/things/systems (or a coordinator, albeit with a diminished role compared to orchestration) receives the Sigma expression from the user, and then multicasts the expression to all the things/systems in the thing-ensemble; each thing involved in the composition (mentioned in the expression) then communicates with other things, appropriate to the given expressions.

For example, given an expression involving three things, modelled as context-aware systems R_1, R_2 and R_3, and a query $(R_1 \oplus R_2) \otimes R_3 \vdash S$ to determine if a situation S is occurring, then in the orchestration approach, the coordinator interprets this expression, and consults the component systems whenever their sensors or rules needs to be used (according to the operational semantics given earlier), e.g., what we represent as finding a rule whose head matches a situation such as $(\{S_1, \ldots, S_k, C_1, \ldots, C_m\} \Rightarrow S) \in \Theta_i$ results in a service call to system R_i. In the choreography approach, each R_i receives (say, from a coordinator) a copy of the expression and the query about S, and works out which other system(s) to cooperate with and how it should interact with the system(s), e.g., R_1, on receiving the expression sees that it is in $\oplus$ with R_2 and so works with R_2 to get a result, and in a mutual decision determines that it should forward its result to the next level, which is to R_3. R_3 works out its own result, and waits for a result from the composition (or one of) R_1 and R_2, and on receiving

[15] http://proto.bbn.com/commons/?q=user

the result, does an ⊗ with the result and forwards the combined result back to the coordinator. Future work will investigate both paradigms.

While we have detailed a range of scenarios, practical deployments of our approach is still needed, as well as, extensive case studies. Future work will consider case studies in smart city applications [58].[16]

References

[1] Gul Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1998.

[2] C. B. Anagnostopoulos, Y. Ntarladimas, and S. Hadjiefthymiades. Situational Computing: An Innovative Architecture with Imprecise Reasoning. *Journal of Systems and Software*, 80(12):1993–2014, 2007.

[3] Malte Appeltauer, Robert Hirschfeld, Michael Haupt, Jens Lincke, and Michael Perscheid. A comparison of context-oriented programming languages. In *COP '09: International Workshop on Context-Oriented Programming*, pages 1–6, New York, NY, USA, 2009. ACM.

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[5] Engineer Bainomugisha, Wolfgang De Meuter, and Theo D'Hondt. Towards context-aware propagators: language constructs for context-aware adaptation dependencies. In *COP '09: International Workshop on Context-Oriented Programming*, pages 1–4, New York, NY, USA, 2009. ACM.

[6] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[7] N. Baumgartner and W. Retschitzegger. A Survey of Upper Ontologies for Situation Awareness. In *Proceedings of Knowledge Sharing and Collaborative Engineering*. ACTA Press, 2006.

[8] Gregory Biegel and Vinny Cahill. A framework for developing mobile, context-aware applications. In *PerCom*, pages 361–365, 2004.

[9] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical models of context-aware systems. In *Foundations of Software Science and Computation Structures (FOSSACS)*, *LNCS 3921*. Springer-Verlag, 2006. http://www.itu.dk/people/birkedal/papers/bigmcs-conf.pdf.

[10] Pietro Braione. *On Calculi for Context-Aware Systems*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, P.zza Leonardo da Vinci 32, 20133 Milano, Italy, April 2004.

[11] Pietro Braione and Gian Pietro Picco. On calculi for context-aware coordination. In Rocco De Nicola, Gianluigi Ferrari, and Greg Meredith, editors, *Proceedings of the 7th International Conference on Coordination Models and Languages (COORDINATION 2004)*, volume 2949 of *Lecture Notes in Computer Science*, pages 38–54. Springer, 2004.

[12] Antonio Brogi, Paolo Mancarella, Dino Pedreschi, and Franco Turini. Modular logic programming. *ACM Trans. Program. Lang. Syst.*, 16(4):1361–1398, 1994.

[13] Rodney A. Brooks. A robust layered control system for a mobile robot. In *Artificial intelligence at MIT: expanding frontiers*, pages 2–27, Cambridge, MA, USA, 1990. MIT Press.

[14] Shanzhi Chen, Hui Xu, Dake Liu, Bo Hu, and Hucheng Wang. A vision of iot: Applications, challenges, and opportunities with china perspective. *Internet of Things Journal, IEEE*, 1(4):349–359, Aug 2014.

[15] Dave Clarke and Ilya Sergey. A semantics for context-oriented programming with layers. In *COP '09: International Workshop on Context-Oriented Programming*, pages 1–6, New York, NY, USA, 2009. ACM.

[16] P.D. Costa, G. Guizzardi, J.P.A. Almeida, L.F. Pires, and M. van Sinderen. Situations in Conceptual Modeling of Context. In *EDOCW '06: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, Washington, DC, USA, 2006. IEEE Computer Society. Available at http://www.loa-cnr.it/Guizzardi/DockhornCosta-et-al-VORTE06_final.pdf.

[17] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[18] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1):1–30, 2005.

[19] Christoph Endres, Andreas Butz, and Asa MacWilliams. A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile Information Systems*, 1(1):41–80, 2005.

[20] Serena Fritsch, Aline Senart, and Siobhan Clarke. Addressing dynamic contextual adaptation with a domain-specific language. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 2, Washington, DC, USA, 2007. IEEE Computer Society.

[21] C. Goumopoulos and A. Kameas. Ambient ecologies in smart homes. *Comput. J.*, 52(8):922–937, 2009.

[22] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660, 2013.

[23] George T. Heineman and William T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[24] Justin Huang and Maya Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, pages 215–225, New York, NY, USA, 2015. ACM.

[25] Achilles Kameas, Irene Mavrommati, and Panos Markopoulos. Computing in tangible: using artifacts as components of ambient intelligence environments. In *In Ambient Intelligence: The evolution of Technology, Communication and Cognition (G.Riva, F.Vatalaro, F.Davide and M.Alcaniz,eds)*, pages 121–142. IOS Press,

[16] http://smartcities.ieee.org/

2004.

[26] Sung Woo Kim, Sang Hyun Park, JungBong Lee, Young Kyu Jin, Hyun-Mi Park, Amy Chung, SeungEok Choi, and Woo Sik Choi. Sensible appliances: applying context-awareness to appliance design. *Personal Ubiquitous Computing*, 8(3-4):184–191, 2004.

[27] Kristian Ellebaek Kjaer. A survey of context-aware middleware. In *SE'07: Proceedings of the 25th conference on IASTED International Multi-Conference*, pages 148–155, Anaheim, CA, USA, 2007. ACTA Press.

[28] Mikkel Baun Kj¿rgaard and Jonathan Bunde-Pedersen. A formal model for context-awareness. Technical report, BRICS: Basic Research in Computer Science, 2006.

[29] Devdatta Kulkarni. Programming framework for sensor-data driven context-aware applications. *SIGBED Rev.*, 5(1):1–2, 2008.

[30] Mike Kuniavsky. *Smart Things: Ubiquitous Computing User Experience Design*. Morgan Kaufmann, 2010.

[31] Seng W. Loke. Service-oriented device ecology workflows. In *ICSOC*, pages 559–574, 2003.

[32] Seng W. Loke. Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective. *Knowl. Eng. Rev.*, 19(3):213–233, 2004.

[33] Seng W. Loke. *Context-Aware Pervasive Systems*. Auerbach Publications, Boston, MA, USA, 2006.

[34] Seng W. Loke. *Supporting Real Time Decision-Making: The Role of Context in Decision Support on the Move*, chapter Towards Real-Time Context Awareness for Mobile Users: A Declarative Meta-Programming Approach, pages 89–112. Springer US, Boston, MA, 2011.

[35] Seng W. Loke. Supporting ubiquitous sensor-cloudlets and context-cloudlets: Programming compositions of context-aware systems for mobile users. *Future Generation Computer Systems*, 28(4):619 – 632, 2012.

[36] Seng Wai Loke. Incremental awareness and compositionality: A design philosophy for context-aware pervasive systems. *Pervasive and Mobile Computing*, 6(2):239–253, 2010.

[37] A.I. Maarala, Xiang Su, and J. Riekki. Semantic data provisioning and reasoning for the internet of things. In *Internet of Things (IOT), 2014 International Conference on the*, pages 67–72, Oct 2014.

[38] Nicolai Marquardt, Tom Gross, M. Sheelagh T. Carpendale, and Saul Greenberg. Revealing the invisible: visualizing the location and event flow of distributed physical devices. In Marcelo Coelho, Jamie Zigelbaum, Hiroshi Ishii, Robert J. K. Jacob, Pattie Maes, Thomas Pederson, Orit Shaer, and Ron Wakkary, editors, *Tangible and Embedded Interaction*, pages 41–48. ACM, 2010.

[39] René Meier, Anthony Harrington, Kai Beckmann, and Vinny Cahill. A framework for incremental construction of real global smart space applications. *Pervasive and Mobile Computing*, 5(4):350–368, 2009.

[40] A.M. Ortiz, D. Hussein, Soochang Park, S.N. Han, and N. Crespi. The cluster between internet of things and social networks: Review and research challenges. *Internet of Things Journal, IEEE*, 1(3):206–215, June 2014.

[41] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):414–454, First 2014.

[42] Qi Saiyu, Xi Min, and Qi Yong. Isotope programming model for context aware application. *International Journal of Software Engineering and Its Applications*, 1(1), 2007.

[43] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

[44] Ulrik Pagh Schultz, Mirko Bordignon, David Johan Christensen, and Kasper Stoy. Spatial Computing with Labels. In *Proceedings of the SASO'08 Spatial Computing Workshop (SCW'08)*, Venice, Italy, October 20 2008.

[45] F. Siewe, A. Cau, and H. Zedan. Cca: A calculus of context-aware ambients. In *Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on*, pages 972 –977, 26-29 2009.

[46] Andrea Sindico and Vincenzo Grassi. Model driven development of context aware software systems. In *COP '09: International Workshop on Context-Oriented Programming*, pages 1–5, New York, NY, USA, 2009. ACM.

[47] J.A. Stankovic. Research directions for the internet of things. *Internet of Things Journal, IEEE*, 1(1):3–9, Feb 2014.

[48] Norbert A. Streitz, Carsten Rocker, Thorsten Prante, Daniel van Alphen, Richard Stenzel, and Carsten Magerkurth. Designing smart artifacts for smart environments. *Computer*, 38(3):41–49, 2005.

[49] Martin Strohbach, Hans-Werner Gellersen, Gerd Kortuem, and Christian Kray. Cooperative artefacts: Assessing real world situations with embedded technology. In *Ubicomp*, pages 250–267, 2004.

[50] Tsutomu Terada and Masahiko Tsukamoto. Smart object systems by event-driven rules. In *Proc. of the 1st International Workshop on Smart Object Systems*, pages 100–109, 2005.

[51] Sal vatore Gaglio, Giuseppe Lo Re, Gloria Martorella, and Daniele Peri. High-level programming and symbolic reasoning on iot resource constrained devices. *EAI Endorsed Transactions on Cognitive Communications*, 15(2), 5 2015.

[52] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0:18, 2004.

[53] T. Weis, C. Becker, and A. Brandle. Towards a programming paradigm for pervasive applications based on the ambient calculus. In *Proc. of the International Workshop on Combining Theory and Systems Building in Pervasive Computing, in conjunction with PERVASIVE 2006*, 2006.

[54] Michael Woolridge and Michael J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[55] Qihui Wu, Guoru Ding, Yuhua Xu, Shuo Feng, Zhiyong Du, Jinlong Wang, and Keping Long. Cognitive internet of things: A new paradigm beyond connection. *IEEE Internet of Things Journal*, 1(2):129–143, 2014.

[56] S.S. Yau and J. Liu. Hierarchical Situation Modeling and Reasoning for Pervasive Computing. In *SEUS-WCCIA '06: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pages 5–10, Washington, DC, USA, 2006. IEEE Computer Society.

[57] Juan Ye, Lorcan Coyle, Simon Dobson, and Paddy Nixon. Ontology-based models in pervasive computing systems. *Knowledge Engineering Review*, 22(4):315–347, 2007.

[58] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *Internet of Things Journal, IEEE*, 1(1):22–32, Feb 2014.

[59] Pascal Zimmer. A calculus for context-awareness. Technical report, BRICS: Basic Research in Computer Science, 2005.