Incremental Awareness and Compositionality: a Design Philosophy for Context-Aware Pervasive Systems

Seng W. Loke
Department of Computer Science and Computer Engineering
La Trobe University
s.loke@latrobe.edu.au

October 8, 2008

Abstract

Context-aware pervasive systems are an important emerging category of software, increasingly pervading into daily life, play and work. These systems are characterized by capabilities for sensing the physical world and taking action, autonomously or in cooperation with users. This paper proposes an incremental approach to building context-aware pervasive systems, with a particular emphasis on systematically extending over time the contexts and situations a system can be aware of, and creating a formalism in which these systems can be composed. We present a formalism of operators for building context-aware pervasive systems incrementally and in a compositional manner (by combining multiple systems and subsystems), facilitating reuse in a formal way. The formalism can serve as (i) a basis for a scripting language for programming composite systems, and (ii) a language for specifying these systems (whether existing or to be built) and then reasoning with specifications of these systems.

1 Introduction

There has been a growing excitement about context-aware pervasive systems, that is, systems capable of knowing and understanding the physical and virtual context of users

and objects and respond intelligently to such knowledge. From context-aware services to artifacts, there has been diverse efforts to build context-awareness into systems [16, 15]. Such systems are termed "pervasive" in the way that they can be ubiquitous and pervade in daily living environments.

This paper proposes an incremental approach to building context-aware pervasive systems, with a particular emphasis on extending over time what a system can be aware of and creating a formalism in which these systems can be composed (perhaps old with old or new with old, or new with new). The idea is that a system initially built might only be capable of recognizing particular contexts or particular situations[1] of entities, but later, can be extended (by developers or even more technically savvy users) to recognize more types of context and more situations. We do not deal with learning in this paper, i.e., applying machine learning techniques to recognize more context and situations, but our approach does not necessarily exclude applying machine learning later. A key underlying assumption in our approach is that knowledge about situations can be modularized or discretized, as in [17, 25].

More specifically, the contributions of this paper are twofold:

1. We propose an abstract model of a situation recognition system, influenced by our discretized view of knowledge about situations. The model makes precise what it means for a system to be more powerful (in recognizing situations) than another system.

2. We then consider a language of operators for combining a mathematical model of context-aware systems, and show how composite systems formed using these operators results in greater recognition ability than individual systems. From a software engineering perspective, this language can serve two purposes, programming and specification:

   - *a basis for scripting or programming composite context-aware pervasive systems*: given an existing set of context-aware systems, the operators can be used to compose any subset of them in a systematic way with well-defined semantics, and
   - *a language for specifying these systems (whether existing or to be built) and then composing specifications of these systems*: at design time, a set of context-aware systems can be represented in abstract form (in a way we will demonstrate later). We can then reason with different possible compositions of these systems to explore the capabilities of these composites

[1] We take situation as being at a high level of abstraction than context as also done in other work [1, 25, 23, 8].

(or of their specifications). Moreover, the same system can be reused in different composites, and hence, the formalism can be used to reason with and represent modular designs.

Our approach has the following benefits:

- *tailored formalism:* we view this language as a first step towards a specialized formalism tailored for building extensible and composable context-aware pervasive systems, in contrast to general specification formalisms for distributed systems or ad hoc approaches.
- *extensible:* our approach supports the modular construction of context-aware pervasive systems, not only by encouraging such systems to be built separately and then composing them using operators (whether at run-time or design-time), but also, the set of operators to be changed (more operators can be added over time).
- *high level of abstraction:* our approach models systems either at the blackbox level, or a whitebox level; but does not require details of the underlying implementation of the system.

By "context", in this paper, we use Dey's definition [10], which is "any information that can be used to characterize the situation of an entity." Although we have in mind the typical sensors (from temperature to motion sensors), we also use a broad definition of a sensor, which is any device (software and hardware) that can be used to acquire context. By situation, we mean a state of affairs or from [9], "a structured part of reality that it (the agent) somehow manages to 'pick out'." And this done by directly perceiving the situation and some reasoning without requiring the agent to give a complete or exact description.

We use an analogy with expert systems, where particular knowledge is added to a generic reasoning engine in order to create a particular expert system, and such knowledge can be updated or extended, thereby updating or extending the expert system. Often, such knowledge might be structured into modules. The typical form of knowledge represented as rules facilitates this process, in that knowledge is in a sense discretized into collections of units of knowledge, where a unit of knowledge is a rule or a bunch of related rules. We suggest that what a system can be aware of can be similarly discretized, and call this *awareness discretization*; for example, a system capable of recognizing situations A, B and C can be extended to recognize situations (i.e., physical situations of the world as well as computational and networking states) A, B, C and

D, or previously a system can only recognize three types of context (e.g., location, time and nearby objects) but is extended to recognize temperature as well (another type of context information). We have in mind a knowledge-based approach in this paper using a representation of situations as units of knowledge, e.g. *situation programs* in [15], but note that our approach is not restricted to only systems using situation programs.

The rest of this paper is organized as follows. Section 2 presents an abstract model of incremental context-awareness, introducing a notion of monotonic extension of context-aware pervasive systems. Section 3 dicusses a language of operators for composing context-aware pervasive systems. Section 4 describes an examples of composition in a smart home context. Section 5 outlines a scheme to compose systems running on different machines. Section 6 notes related work and Section 7 concludes with directions for future work.

2 An Abstract Model of Incremental Context-Awareness

Following [16], we consider a typical context-aware pervasive system as having three subsystems in order to recognize situations: a subsystem comprising sensors, a subsystem mapping sensor readings to context or models of situations, and a subsystem taking actions. If we are extending a context-aware system, we want to be able to say, in some precise way, the sense in which one system extends another, in terms of the ability to recognize situations, and the following subsections aim towards this.

2.1 Recognition Power

Suppose that, with respect to a system R, we use the operator $R(K) \vdash S$ to mean, the system with capability K is able to recognize situation S. By "capability" we mean knowledge as well as perhaps sensors and reasoning components to make recognition of situations possible.

We can define the *recognition power* $\mathcal{R}_K^R$ of the system with capability K as *the set of all situations that the system can correctly recognize*, and write this as follows:

$$\mathcal{R}_K^R = \{S \mid R(K) \vdash S\}$$

While the recognition power of a system can be regarded as a property of the system, we can also use a set of situations as a specification or requirement for a system: we want to build a system that has to correctly recognize a given set of situations, i.e. the recognition power of the system we build is either equal to or larger than the given set.

2.2 Breadth-Monotonic Extension

Now, if we add capability to the system, representing this additional capability by δ, and suppose that this additional capability allows the system to recognize *monotonically* more situations, i.e., $\mathcal{R}_K^R \subseteq \mathcal{R}_{K\cup\delta}^R$. Then, we say that we have *breadth-monotonically* increased the capability of the system, and write this as

$$R(K) \leq_{bm} R(K \cup \delta)$$

or more generally, we simply write $R \leq_{bm} R'$, given two systems R and R' where their capabilities are understood. Monotonicity is a useful property we would like to have, which makes the system amenable to an incremental approach of construction.

2.3 Depth-Monotonic Extension

There are other means by which an extension can be monotonic, i.e., when there is an uncertainty (e.g., a probability or confidence measure) associated with the recognition of a situation. We assume that the system is able to provide a measure of how likely a situation is occurring, albeit this measure is merely the system's own estimate.

For example, on a certainty scale of 0 (totally uncertain) to 100 (totally certain), and we write $R(K) \vdash S[u]$ to mean the system with capability K recognizes situation S with a measure of u (we assume that least $u > 0$ for otherwise, we cannot say that the system recognizes the situation). Then, another way in which we can extend the capabilities of the system with respect to the situation S is to add δ_S to K such that situation S is recognized with higher certainty $v(\geq u)$, written as $R(K \cup \delta_S) \vdash S[v]$. More generally, we can add capability δ to R so that R recognizes the situations, at least as estimated by the system, with no less certainty than previously or even with greater certainty: δ is such that for each S recognized by R with capability K, i.e. suppose that $R(K) \vdash S[u]$ for some certainty measure u, we then have $R(K \cup \delta) \vdash S[v]$ for some $v \geq u$. We say that we have *depth-monotonically* increased the capability of the system, and write this as

$$R(K) \leq_{dm} R(K \cup \delta)$$

or more generally, we simply write $R \leq_{dm} R'$, given two systems R and R' where their capabilities are understood.

Note that both the above monotonicities permit equality, i.e., it is possible that $R_K = R_{K\cup\delta}$ for some δ. But say, assuming that we are comparing unequal systems, then an extension may be depth-monotonic but not breadth-monotonic, in that the ex-

tension does not allow any new situation to be recognized, but at least some situation(s) is(are) recognized with greater certainty. We have the converse when the extension allows more situations to be recognized but existing situations are not recognized with greater certainty.

2.4 Effective Recognition Power

We can define the *effective recognition power* $\mathcal{ER}_K$ of the system with capability K as the set of all situations that the system can recognize as follows together with their certainty measure; for a system R:

$$\mathcal{ER}_K^R = \{(S, u) \mid R(K) \vdash S[u]\}$$

and then, $R(K) \leq_{dm} R(K \cup \delta)$ means for each $(S, u) \in \mathcal{ER}_K^R$, we will have $(S, v) \in \mathcal{ER}_{K \cup \delta}^R$ such that $u \leq v$.

2.5 Discussion

The idea of incremental awareness is, hence, to effectively "grow" the system over time in such a way as to preserve either or both of these monotonicity properties, the system at a later time being a depth and/or breath monotonic extension of the previous version.

We make three further observations:

- These notions of monotonicity can also be used to compare two or more different context-aware systems, in terms of their recognition power, or to clearly define the intersection of two systems. For example, given two systems R and R', with capabilities K and K' respectively, the difference between the two (say what R can recognize but R' cannot) is as follows:

  $$\mathcal{R}_K^R \setminus \mathcal{R}_{K'}^{R'}$$

  and similarly, one can compute what R' can recognize and R cannot. Their intersection provides the situations both can recognize.

  We note, however, that such a comparison is only meaningful when the situations a system can recognize is enumerated based on an explicit or implicit set of situations (expressed in some ontology, say), i.e. if R recognizes a situation "Tom is in a meeting," and R' recognizes also the same situation, then both

systems should refer to the same situation in an ontology (or at least the two situations must be represented as synonymous in the ontology).

- Note that we consider recognizing the occurrence of a situation as being at the same level as the non-occurrence of a situation (the situation of the non-occurrence of a situation is itself a situation). A system may be able to recognize the occurrence of a situation but may fail to recognize the non-occurrence of a situation. The fact that a system did not recognize the occurrence of a situation does not necessarily mean that the system is certain that the situation is not occurring. Note that, however, in a specific application, we might choose to use the Closed World Assumption (as done for interpreting negation in many logic-based languages), i.e., any situation that is not detected by the system is assumed to be not occurring (at least from the system's perspective).

- For comparison purposes, when one is developing towards an ideal (or desired) system, we can define an imaginary system, called the *oracle*, which is the depth and breadth monotonic extension of every other system, i.e. the oracle always recognizes situations correctly, whether they occur or do not occur. Then, the closer the (effective) recognition power of a system is to the oracle, the "better" the system becomes. Practically, given a set of situations being considered as the requirements on what a system should recognize, one can define an *oracle with respect to that set*, i.e. the oracle for that set will always recognize all the situations in that set correctly. The oracle is merely a formalization of the intuitive notion of the required ideal system.

- Note that we have presented non-strict versions of the relations $\geq_{bm}$ and $\geq_{dm}$, but we can similarly define the strict versions of the relations (without equality), i.e. $>_{bm}$ and $>_{dm}$.

3 Composing Systems

We explore operators which combine two or more systems, and formalize the relationship between the combined system and its component systems. We mentioned that the situations must come from (or be mapped to) a common ontology; so that it makes sense to say that two systems are recognizing the same or different situations. This is assumed in the discussions which follow. Section 3.1 first presents simple composition operators, section 3.2 then discusses more sophisticated compositions, presenting

a language of operators in order to illustrate what is possible, and Section 3.3 discusses completeness and expressiveness of sets of operators.

3.1 Simple Composition

Ideally, we would want the combined system to be a depth and/or breadth monotonic extension of its component systems. We discuss two operators here analogous to set union and set intersection.

Union. We can define the $union$ combination (denoted by $\oplus$) of two systems R and R', denoted by $R \oplus R'$, each with capabilities K and K', respectively, as follows (the combined capability denoted by $K \oplus K'$):

$$\mathcal{R}^{R \oplus R'}_{K \oplus K'} = \mathcal{R}^{R}_{K} \cup \mathcal{R}^{R'}_{K'}$$

Hence, $R \oplus R'$ exactly recognizes all the situations that R and R' recognizes.

And considering uncertainty, an initial definition would be

$$\mathcal{ER}^{R \oplus R'}_{K \oplus K'} = \mathcal{ER}^{R}_{K} \cup \mathcal{ER}^{R'}_{K'}$$

This definition of $\mathcal{ER}^{R \oplus R'}_{K \oplus K'}$ is sensible if there are no situations recognized by both R and R'. Now, if some situation S is recognized by both systems, then both (S, u) and (S, u') would exists for measures u and u' estimated by R and R', respectively. This would be the correct definition and, operationally, either u or u' would be applicable - we nondeterministically choose one of them. In an operational definition of union we give later, we nondeterministically choose either u or u'. This is because, practically, if a situation is recognized by one of the systems (say, with certainty u), using short-circuit evaluation as an optimization for performance, we may not proceed to determine if the situation is recognized by the other system.

But if we do *not* apply short-circuit evaluation, i.e., we still proceed to determine if the situation is recognized by the other system, and possibly obtain another certainty measure - we can then compute the greater certainty measure of the two (assuming the measures are comparable, or if not initally comparable, perhaps mapped to a common scale for comparison). Taking the greater of the two measures, denoted by $max(u, u')$, is conservative in that we would expect a situation recognized by two systems to have greater certainty than if recognized by only one, so that the measure might actually be greater than $max(u, u')$. Nevertheless, quantifying this greater certainty is not obvious

given two different systems and two different ways by which certainty values might have been computed - such certainty values would have to be computed via a module added when the systems are "glued" together.

Taking this conservative stance, the effective recognition power of this union of two systems R and R' can then be given by the union of the set of situations recognized by both systems (with certainty measure being the maximum of the two), the situations recognized by only R and the situations recognized by only R', i.e.:

$$\begin{aligned}\mathcal{ER}^{R\oplus R'}_{K\oplus K'} = \quad & \{(S, max(u, u')) \mid R(K) \vdash S[u] \text{ and } R'(K') \vdash S[u']\} \\ & \cup \{(S, u) \mid R(K) \vdash S[u] \text{ and } R'(K') \nvdash S\} \\ & \cup \{(S, u') \mid R'(K') \vdash S[u'] \text{ and } R(K) \nvdash S\}\end{aligned}$$

We review this version of union in Section 3.2.3.

Intersection. Similarly, we can define the recognition power of intersection, denoted by $\otimes$, of R and R' as follows (if we leave out certainty measures):

$$\mathcal{R}^{R\otimes R'}_{K\otimes K'} = \mathcal{R}^{R}_{K} \cap \mathcal{R}^{R'}_{K'}$$

The composite system $R \otimes R'$ recognizes only situations recognized by both its constituent systems. This means that $R \otimes R'$ is not a breadth monotonic extension of either of its constituents but recognition by both can serve a confirming role.

Taking into account uncertainty, and as before, taking a conservative stance, we define the intersection of two systems (with certainty measures) as follows:

$$\mathcal{ER}^{R\otimes R'}_{K\otimes K'} = \{(S, max(u, u')) \mid R(K) \vdash S[u] \text{ and } R'(K') \vdash S[u']\}$$

Expressions of Compositions. With the two operators above, we can define a set of composition expressions in EBNF form:

$$E \ ::= R \mid E \oplus E \mid E \otimes E$$

where R here stands for an identifier of some system. Such expressions can either be used as specifications of systems, or the operators represent different types of "glue" for systems.

Discussion. A question is how can we extend a system, or combine two (or more) systems, such that the extension, or composition, is a depth or breadth monotonic extension of the original system(s)? The answer is clearly in the affirmative if we apply union. However, in general, this question can only be answered with respect to the specific technologies we use to build the system.

Systems can be "glued" together synergistically. It might be possible to combine two systems such that the situations the composite system recognize is more than the union and the certainties are at least equal to its constituent systems or even greater. Substantial engineering may be required but one way to achieve this synergy is when situations are defined which are composite or comprising other situations (as done in [23]). So, given that to recognize a situation S1, two other situations S2 and S3 need to be recognized according to a knowledge base relating S1, S2 and S3, then, if R can recognize S2 and R' can recognize S3, then $R \oplus R'$ can recognize S2 and S3 but if also extended with access to the knowledge base, also recognize S1. Such a composition does not require engineering of the internals of R and R' but merely to take their outputs and relate them via the knowledge base. The certainty measure with which R1 is recognized would have to depend on the measures from R2 and R3, and the procedure to compute the certainty measure for R1 would have to consider the details of the mechanisms used to compute certainty measures in R and R'.

3.2 Deeper Composition

So far, our model of systems and their capabilities have been opaque (i.e., we assume a blackbox) and based on their recognition power. Keeping the capabilities and the system a blackbox means that the above definitions and terminology applies to systems independently of their internal implementation. To model more varieties of compositions, at the cost of generality, we consider now a more detailed yet abstract architecture of context-aware systems.

3.2.1 A WhiteBox Model of a Context-Aware System

Based on the location stack model in [13], we generalize location to context in our model, and define a context-aware system R as a tuple (or a triple) comprising sensors, an interpreter (which translates sensor readings to context information), and a situation-reasoner (which takes the context information and infers possible situations, or "aggregates" situations to infer more situations), and model each of these compo-

nents as a relation. More formally:

$$R = (\Sigma, \Pi, \Theta)$$

In the above formulation, 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. $\sigma_i : \mathbf{W} \rightarrow \mathbf{G}$. In the examples later, G is represented in the form of equalities or inequalities (i.e., ranges) on sensor readings, and we use the function symbol $reading(\sigma, T)$ to denote the reading of a sensor σ at some time T (or simply, $reading(\sigma)$ if time is implicit). 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; for example, W can be defined as a 3-dimensional volume of spherical space (and $\mathbf{W}$ as a set of such spaces) at some location of a prescribed size where the sensor is contained, and G can be a sensor stream comprising a set of timestamped data samples as in [14], and $\mathbf{G}$ a set of such streams of readings.

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 in work surveyed in [3, 2, 24] - again, we do not need the details of this), 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 $(G_i, C_i) \in \Pi$).

The separation of sensors and interpreters is for the purpose of generality; the same sensor readings can be mapped to different contexts (i.e., interpreted differently) by different systems. This would allow us to model a composition, where one system may utilize a sensor of the another system but interpret the readings of this sensor differently, i.e. mapping the readings to another type of context. For example, a set of GPS coordinates may be reverse geocoded to "Bob's room" in one system and to "PS1 224" in another.

The situation reasoner Θ is a pair of relations (Θ_c, Θ_s) with Θ_c, mapping sets of contexts to situations, and Θ_s, mapping sets of situations to other situations, where

$$\Theta_c \subseteq (\wp(\mathbf{C}) \times \mathbf{S})$$

where $\mathbf{S}$ is a set of situations (again, which we assume grounded in some ontology), and

$$\Theta_s \subseteq (\wp(\mathbf{S}) \times \mathbf{S})$$

Examples of aggregating context to infer situations can be found in the literature [1, 25, 23, 8, 12], as well as aggregating situations to infer situations [23, 25]. Ontologies for what may constitute contexts exists such as SOUPA[2], and CONON [22].

Our model makes the distinction between the notions of context and situation following the work such as [22, 23, 20, 15, 10], where as mentioned earlier, in Dey's definition [10], context information is used to "characterize the situation of an entity," modelled via Θ_c. Hence, we see situation as the state of affairs that aggregated context information aim to inform about. However, to maintain generality, we include Θ_s since relationships among situations also need to be captured (e.g., as noted in [25, 9][3]).

We consider how examples from different sources, such as [22, 23, 20], can be represented in our approach as follows.

Adapted from [22]; we can think of the following pairs as members of Θ_c:

```
({u located in bathroom, waterheater located in bathroom,
      bathroom-door status closed, waterheater status on},
  u's situation is showering)

({u located in kitchen, electric-oven located in Kitchen,
      electric-oven status on},
  u's situation is cooking)

({u located in living room, tv-set located in living room,
      tv-set status on},
  u's situation is watching tv)
```

From [23], we have

```
({u in conference room, room light on},
  ready for meeting)
```

And from [20], we have

```
({more than 3 people in room 2401, powerpoint running in room},
  presentation)
```

and

```
({more than 1 people in room 2401, mpegplayer running},
 movie presentation)
```

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

[3] We note in http://www.stanford.edu/~kdevlin/HHL_SituationTheory.pdf that one example of a relationship between situations is "involves", where if one (or more) situation *involves* another, then the occurrence of the situation(s) may imply the occurrence of another situation, which is what Θ_s can be used to represent.

We assume a "blackbox" in realizing Θ_c and Θ_s, even if there may be some sophisticated mechanism mapping context to situations or inferring a situation from other situations and context.[4] Hence, our approach can be used to model an existing context-aware system, i.e. a triple represents the observable properties of the existing system, or to model systems to be created, i.e. a triple represents the observable properties of a system to be built.

3.2.2 Operators

Here, we define a context-aware system as effectively taking some part of the world and mapping that to a set of situations. Based on this "whitebox" notion of a context-aware system, we can define operators involving deeper interactions between systems, where the output of the component of one system can become the input to a component of another system and so on. In order to define these operators, we will employ the style of Plotkin's structural operational semantics for programming languages [19]; we will use rules of the form

$$\frac{premises}{conclusion}$$

which means the $conclusion$ holds whenever the $premises$ hold. In the discussions which follow, we shall omit certainty measures for simplicity, until Section 3.2.4.

We first 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. A situation that is recognized by a system is then either computed from contexts (recognized by some sensors and the interpreter) via Θ_c or aggregated via Θ_s from other recognized situations (it could be from both context and situations as appropriately modelled). This meaning of the

[4] Note that inferring situations via a mix of situations and context can be represented by appropriate definitions for Θ.

relation $\vdash$ can be expressed recursively as follows:

$$\frac{\begin{array}{l}\text{Where} \quad \Theta = (\Theta_c, \Theta_s), \text{either} \\ (i) \quad [(\{C_1, \ldots, C_m\}, S) \in \Theta_c \\ \text{for some } m, \text{where for each } C_i,\ i \in \{1, \ldots, m\}, \\ \Pi(\sigma_j(W)) = C_i, \text{for some } j \text{ and } \sigma_j \in \Sigma], \\ \text{or} \\ (ii) \quad [(\{S_1, \ldots, S_k\}, S) \in \Theta_s \text{for some } k, \\ \text{where for each } S_i,\ i \in \{1, \ldots, k\},\ (\Sigma, \Pi, \Theta) \vdash (W, S_i)]\end{array}}{(\Sigma, \Pi, \Theta) \vdash (W, S)} \ (one - system)$$

Similarly, we would have context recognition, i.e., $R \vdash (W, C)$ for some context C, where $\Pi(\sigma(W)) = C$, for some $\sigma \in \Sigma$. And one could generalize the definition of recognition power to include not just situations but also contexts.

With this whitebox model of a system, we can define capabilities now in terms of the three components of a system and be more specific about what extending the capabilities of a system could mean. Given a system $R = (\Sigma, \Pi, \Theta)$, extending the capabilities of a system can mean either adding sensors to, extending the interpreters or the situation recognizers. Note that based on the model of these components as relations, extending the system could be adding sensors to Σ, with the need to extend the relation Π to interpret the larger range of sensor readings, and extending Θ to now consider the newly available context information.

Consider the operators defined on two systems $R = (\Sigma, \Pi, \Theta)$ and $R' = (\Sigma', \Pi', \Theta')$.

Union. Then, the union can be given by the rule $(r1)$:

$$\frac{R \vdash (W, S) \ \ or \ \ R' \vdash (W, S)}{(R \oplus R') \vdash (W, S)} \ (r1)$$

We discuss how union without short-circuit evaluation can be defined later in Section 3.2.3.

Intersection. Intersection is defined by the rule $(r2)$:

$$\frac{R \vdash (W, S) \ \ and \ \ R' \vdash (W, S)}{(R \otimes R') \vdash (W, S)} \ (r2)$$

Tight-union. To model deeper "cooperation" among systems, we define an operator, which we call *tight-union*, denoted by $+$, recursively, which combines the use of sensors, interpreters and situation recognizers from two systems in a tightly coupled way:

$$\begin{array}{l}
\text{Where } \Theta = (\Theta_c, \Theta_s), \text{ and } \Theta' = (\Theta'_c, \Theta'_s), \text{ either} \\
(i) \quad [(\{C_1, \ldots, C_m\}, S) \in \Theta_c \quad or \quad (\{C_1, \ldots, C_m\}, S) \in \Theta'_c \\
\text{for some } m, \text{ where for each } C_i, i \in \{1, \ldots, m\}, \\
\Pi(\sigma(W)) = C_i, \text{ or } \Pi'(\sigma(W)) = C_i, \text{ for some } \sigma \in (\Sigma \cup \Sigma')], \\
\text{or} \\
(ii) \quad [(\{S_1, \ldots, S_k\}, S) \in \Theta_s \quad or \quad (\{S_1, \ldots, S_k\}, S) \in \Theta'_s \text{ for some } k, \\
\text{where for each } S_i, i \in \{1, \ldots, k\}, ((\Sigma, \Pi, \Theta) + (\Sigma', \Pi', \Theta')) \vdash (W, S_i)] \\
\hline
((\Sigma, \Pi, \Theta) + (\Sigma', \Pi', \Theta')) \vdash (W, S)
\end{array} \quad (r3)$$

From the above rule, defined recursively, we observe that at each step (of the recursion) to provide context information, sensors and interpreters from either system can be employed, and to infer situations, situation-recognizers from either system can be employed.

The rule $(r3)$ specifies a deep cooperation of the two systems, but declaratively, we can also define tight-union equivalently as follows:

$$(\Sigma, \Pi, \Theta) + (\Sigma', \Pi', \Theta') = (\Sigma \cup \Sigma', \Pi \cup \Pi', (\Theta_c \cup \Theta'_c, \Theta_s \cup \Theta'_s))$$

where $\Theta = (\Theta_c, \Theta_s)$ and $\Theta' = (\Theta'_c, \Theta'_s)$. From this definition, we also note that tight-union is commutative and associative.

We can see that tight-union is hence more powerful than union since it involves a tighter coupling of use of components - for example, context from the interpreters can be fed into the situation recognizers of either system, and situations recognized in one system can be fed into the situation-recognizers of the other system, and not merely one of them. Hence, it is clear that:

$$(R \oplus R') \leq_{bm} (R + R')$$

but tighter integration may be more difficult to implement and involves more communication between the systems. For example, we can have $R = R' + R'' + R'''$, where $R = (\Sigma, \Pi, \Theta)$, $R' = (\Sigma, \emptyset, \emptyset)$, $R'' = (\emptyset, \Pi, \emptyset)$, and $R''' = (\emptyset, \emptyset, \Theta)$, i.e. we can

further effectively decouple a system into its components via tight-union. Similarly, we can model a system R which acquires context from several sources and then reason with such information using the expression: $R' + R'' + R'''$, $R' = (\Sigma', \Pi', \emptyset)$, $R'' = (\Sigma'', \Pi'', \emptyset)$, and $R''' = (\emptyset, \emptyset, \Theta)$, i.e. R''' reasons with the contexts acquired via R' and R''.

Extended Expressions of Compositions. With these three operators, we can define a set of composition expressions in EBNF:

$$\begin{aligned} Q &::= R \mid Q + Q \\ E &::= Q \mid E \oplus E \mid E \otimes E \end{aligned}$$

where R represents a system and is of the form (Σ, Π, Θ). Then, $E \vdash (W, S)$ can be computed by using this rule for tight-union generalizing from $(r3)$:[5]

$$\frac{(\bigcup_{i=1}^{n} \Sigma_i, \bigcup_{i=1}^{n} \Pi_i, (\bigcup_{i=1}^{n} \Theta_{ic}, \bigcup_{i=1}^{n} \Theta_{is})) \vdash (W, S)}{Q \vdash (W, S)} \ (tu)$$

where $n > 1$, $Q = (\Sigma_1, \Pi_1, (\Theta_{1c}, \Theta_{1s})) + \ldots + (\Sigma_n, \Pi_n, (\Theta_{nc}, \Theta_{ns}))$, and using generalized forms of the above rules for union (with R replaced by E in the rule $r1$) and intersection (also with R replaced by E in the rule $r2$), as follows:

$$\frac{E \ \vdash \ (W, S) \ \ or \ \ E' \ \vdash \ (W, S)}{(E \oplus E') \ \vdash \ (W, S)} \ (union)$$

and

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

[5] An equivalent form of this rule, for any positive integer n, is simply:

Where $\Theta_1 = (\Theta_{1c}, \Theta_{1s})$, ..., $\Theta_n = (\Theta_{nc}, \Theta_{ns})$, either

(i) $[(\{C_1, \ldots, C_m\}, S) \in \Theta_{1c} \quad or \ldots or \quad (\{C_1, \ldots, C_m\}, S) \in \Theta_{nc}$
for some m, where for each C_i, $i \in \{1, \ldots, m\}$,
$\Pi_1(\sigma(W)) = C_i$ or ... or $\Pi_n(\sigma(W)) = C_i$, for some $\sigma \in (\Sigma_1 \cup \ldots \cup \Sigma_n)]$,
or

(ii) $[(\{S_1, \ldots, S_k\}, S) \in \Theta_{1s} \quad or \ldots or \quad (\{S_1, \ldots, S_k\}, S) \in \Theta_{ns}$ for some k,
where for each S_i, $i \in \{1, \ldots, k\}$, $((\Sigma_1, \Pi_1, \Theta_1) + \ldots + (\Sigma_n, \Pi_n, \Theta_n)) \vdash (W, S_i)]$
$\overline{((\Sigma_1, \Pi_1, \Theta_1) + \ldots + (\Sigma_n, \Pi_n, \Theta_n)) \vdash (W, S)}$

Then, we can compute a proof tree for $(R + R') \oplus (R'' \otimes R''') \vdash (W, S)$:

$$\dfrac{\dfrac{\vdots}{(R + R') \vdash (W, S)} \; or \; \dfrac{\dfrac{\vdots}{R'' \vdash (W, S)} \; and \; \dfrac{\vdots}{R''' \vdash (W, S)}}{(R'' \otimes R''') \vdash (W, S)}}{(R + R') \oplus (R'' \otimes R''') \vdash (W, S)}$$

We can define an equivalence relationship among systems, denoted by overloading "=", which means having equal recognition power, i.e. $R = R'$ if and only if $\mathcal{R}^R = \mathcal{R}^{R'}$. For expressions, this *operational equivalence* between two composition expressions E and E' can be stated simply as follows:
$E = E'$ if and only if $E \vdash (W, S)$ whenever $E' \vdash (W, S)$, and vice versa.

Also, we have that $R = R' + R''$, whenever $R = (\Sigma, \Pi, \Theta)$, $R' = (\Sigma, \emptyset, \emptyset)$, and $R'' = (\emptyset, \Pi, \Theta)$, i.e. we can effectively decouple a system from its sensors via tight-union. We can envisage systems with specific sensing, interpretation *or* reasoning capabilities. For example, we can compose a system $R = (\Sigma, \Pi, \Theta)$, with each component non-empty and $R' = (\emptyset, \emptyset, \Theta)$, which does not have any sensors or interpretations, but reasons with context and situations to infer situations. So, the system $R + R'$ has greater reasoning capabilities than R alone. On the other extreme, we may have that $R' = (\Sigma, \emptyset, \emptyset)$, i.e., R' is only a collection of sensors so that $R + R'$ has more sensors than R, but maintaining the same reasoning capabilities. Representative of such sensors (or sensor-only systems) would be those available and described using SensorML,[6] and these made public can be discovered and queried by any system.

3.2.3 Extending the Repertoire of Operators

We explore two further operators: we demonstrate how failure to recognize a situation can also be used in defining an operator, and a variant of union where short-circuit evaluation is forcibly avoided (at the same time showing how derived operators can be defined in terms of existing operators).

Overriding-union and sure-union. Given E, W and S, if the relation $E \vdash (W, S)$ does not hold, we denote this by $E \nvdash (W, S)$.

We define a binary operator to represent a system which recognizes situations using one of its operands, and only passes on situations it can't recognize to another, which

[6] See http://www.sensorsmag.com/articles/0403/30/main.shtml and http://www.opengeospatial.org/standards/sensorml

we term, *overrding-union*, denoted by $\lhd$:

$$\frac{R \vdash (W,S) \ \ or \ \ ((R \nvdash (W,S)) \ \ and \ \ (R' \vdash (W,S)))}{(R \lhd R') \vdash (W,S)}$$

This means that even if R' can recognize the situation, R is preferred. Note that overriding-union is equivalent to union in terms of the situations that can be recognized, i.e. they have the same recognition power: $\mathcal{R}^{R \lhd R'} = \mathcal{R}^{R \oplus R'}$. Note that we can generalize this rule to expressions by replacing R with E:

$$\frac{E \vdash (W,S) \ \ or \ \ ((E \nvdash (W,S)) \ \ and \ \ (E' \vdash (W,S)))}{(E \lhd E') \vdash (W,S)} \ (ou)$$

Note that even though *or* in the premise does not explicitly impose an order of evaluation, in evaluating such an expression, one should start with $R \vdash (W,S)$ and if that fails, only then do we need to consider trying to determine $R' \vdash (W,S)$. Assuming this order of evaluation, we can use overriding union to "direct" evaluation to increase depth monotonicity due to the above rule. For example, evaluating a composition such as

$$(R \otimes R') \lhd (R \oplus R') \quad \vdash \quad (W,S)$$

would mean we first try to recognize the situations recognized by the intersection of the systems $R \otimes R'$, and only if this fails, do we then consider each R and R' separately, i.e.

$$((R \otimes R') \lhd (R \oplus R')) =_{bm} \ R \oplus R'$$

but

$$((R \otimes R') \lhd (R \oplus R')) \geq_{dm} R \oplus R'$$

from the definition of intersection. Hence, while with union (and short-circuit evaluation), we stop as soon as the situation is recognized by either one system, with $(R \otimes R') \lhd (R \oplus R')$, we continue to attempt to recognize the situation in the other system even when it has been recognized in one (since we aim first for success with $R \otimes R'$) in order to attain a (possibly) higher certainty measure (e.g., as with the conservative stance as mentioned in Section 3.1). And this is also different from only $R \otimes R'$, since if we fail to recognize the situation in both systems, we can still succeed with either, individually.

We name such a composition *sure-union* and use $\bigtriangledown$ to represent this, i.e. $R \bigtriangledown R'$

is $(R \otimes R') \lhd (R \oplus R')$, or more generally:

$$E \bigtriangledown E' \;=\; (E \otimes E') \lhd (E \oplus E')$$

and state that

$$(E \bigtriangledown E') \geq_{dm} (E \oplus E')$$

and

$$(E \bigtriangledown E') =_{bm} (E \oplus E')$$

Sure-union effectively formalizes the union without short-circuit evaluation discussed in Section 3.1.

More Expressions of Compositions. Adding these new operators yields an extended language describing composite systems:

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

We use sure-union ($\bigtriangledown$) as an example of a derived operator, being defined in terms of the other operators. The list is not exhaustive, further useful operators can be defined and similarly named, as we discuss further in Section 3.3. An operator combining two or more systems captures in a succint form how these system effectively "cooperate" in recognizing a situation.

Given an environment into which is embedded several such context-aware systems, one can evaluate if a situation can be recognized by a composition of these systems, perhaps even by different compositions of the same systems.

3.2.4 Uncertainty

At this point, we note that our computations of $\vdash$ relationships can be augmented to return (un)certainty measures (in the subscript) though in general. For union, it depends on which operand succeeds:

$$\frac{E \;\vdash_u\; (W,S)}{(E \oplus E') \;\vdash_u\; (W,S)} \quad or \quad \frac{E' \;\vdash_{u'}\; (W,S)}{(E \oplus E') \;\vdash_{u'}\; (W,S)}$$

Note, unlike sure-union, where situations recognized by both R and R' have certainty computed from the maximum of that from R and R', in this operational definition of union, we nondeterministically take one of the measures from a successful evaluation (if any).

For intersection, we have

$$\frac{E \vdash_u (W,S) \ and \ E' \vdash_{u'} (W,S)}{(E \otimes E') \vdash_v (W,S)}$$

where $v = max(u, u')$, by definition (recall the definition of intersection from Section 3.1).

For the single system case, $R \vdash_u (W,S)$, we assume that this is built into the system and u is a property of R given W and S. We model this abstractly using a blackbox oracle function $\mathcal{O}$ which takes a system (i.e., a triple) and a situation and returns a certainty measure if the system recognizes the situation (i.e., $\mathcal{O}(R,S) = u$), and is undefined, otherwise.

Then, for tight-union, the certainty measure is also given by the oracle function, that is, $(R_1 + \ldots + R_n) \vdash_v (W,S)$ is such that:

$$v = \mathcal{O}(R_1 + \ldots + R_n, S) = \mathcal{O}(R,S)$$

where $R = R_1 + \ldots + R_n$.

For overriding-union, similar to union, the certainty measure is nondeterministically chosen from a successful evaluation:

$$\frac{E \vdash_u (W,S) \ or \ ((E \not\vdash (W,S)) \ and \ (E' \vdash_{u'} (W,S)))}{(E \lhd E') \vdash_v (W,S)} \ (ou)$$

where v is either u or u'.

For sure-union, being a derived operator, the certainty measure is computed following the definitions of union, intersection and overriding-union.

3.2.5 Operational Algebraic Properties

We summarize the operational algebraic properties of the operators as follows; these properties are operational in that they hold with respect to situation recognition. For example, an operator $\odot$ is said to be *operationally commutative* whenever:

$$R \odot R' \vdash (W,S) \text{ iff } R' \odot R \vdash (W,S)$$

and similarly, *operationally associative* whenever:

$$(R \odot (R' \odot R'')) \vdash (W, S) \text{ iff } ((R \odot R') \odot R'') \vdash (W, S)$$

Following the above operational semantics or rules of evaluation, union, intersection, and tight-union are operationally commutative and operationally associative, but overriding-union and sure-union are not.

3.3 Completeness and Expressiveness

A key question that arises is, given a well-delineated set C of composite systems, whether there would be a set of simple systems together with a complete set of operators that could exhaustively model (or generate) this set C. The set of operators we have presented so far have been guided by intuitions from set theory (e.g., union and intersection) and Brogi's work on compositional operators for logic programming [6]: union and intersection are inspired directly by set theory, tight-union is a variant of union arising from a whitebox modelling of systems, overriding-union is based on Brogi's work and inspired by inheritance from object-oriented programming, and sure-union exemplifies a derived operator for controlling evaluation using overriding-union. One could continue to define further variations, either new primitive operators or derived operators. In theory, there is no limit to the range of operators one could define (and in this sense, enabling extensibility). The larger the set of operators, the greater the expressive power of the language, and the range of composites, but the more difficult it is for programmers or designers to comprehend the full range of possible composites. Figure 1 summarizes the set of operators and rules we have defined in this paper.

However, we can do the reverse more conveniently, which is, given a set of simple systems and a set of operators, we can delineate the (possibly infinite) set of composite systems definable using these operators, by providing an EBNF definition of the corresponding set of expressions. For example, given a set of systems $\{R_1, R_2, R_3\}$ and a set of operators $\{\oplus, \otimes\}$, we can define the following set F of expressions:

$$F \ ::= R_1 \mid R_2 \mid R_3 \mid F \oplus F \mid F \otimes F$$

that corresponds to a well-defined set of composite systems which includes R. Hence, classes of systems can be defined "bottom-up" this way.

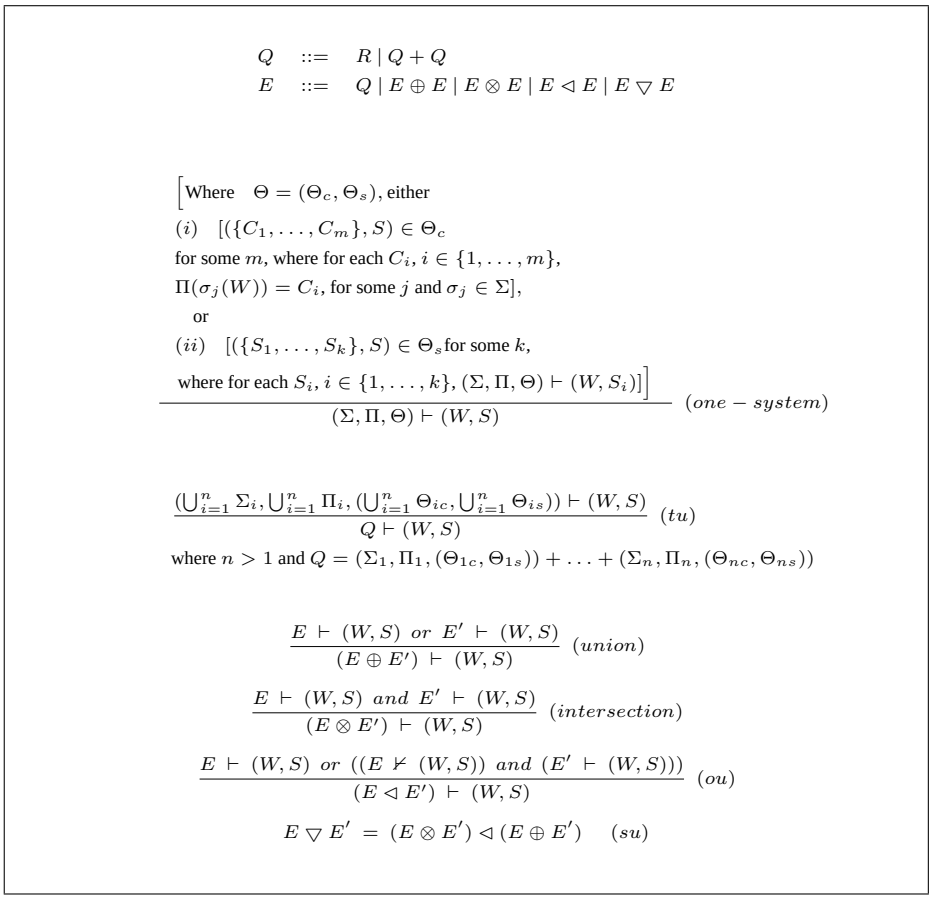

Figure 1: Overview of operators and associated rules (without certainty measures).

4 Examples: Smart Home Scenarios

We consider incrementally constructing three context-aware systems, each stage extending it with additional context-aware behaviours. This section illustrates the abstract formalism we provided in the previous section, providing a concrete example of modelling context-aware systems using our notation and showing how a context-aware system can be built incrementally in our approach.

So far, we have considered situation and context recognition separately from actions. On recognizing a situation or even a context, a system might perform an action. Given a system $R = (\Sigma, \Pi, \Theta)$ and its recognition power $\mathcal{R}^R$, suppose that R recognizes the set of contexts $\mathbf{C}$, then we can define an action module M which maps (sets of) recognized situations and contexts to an action belonging to some set of actions $\mathbf{A}$, i.e. $M : \wp(\mathbf{C}) \times \wp(\mathcal{R}^R) \rightarrow \mathbf{A}$. So, a *context-aware system with actions* is denoted by a pair (R, M). If different actions for the same system (and o, the same set of recognized contexts or situations) is required, we can represent this as (R, M') where $M \neq M'$.

Consider a smart home with sensors and context-aware reasoning. We assume that there is a central system which coordinates the sensing, reasoning and actions for this home. Also, to start with, suppose the home has two types of sensors and context-aware behaviours:

- living room lights turn on automatically when the user enters the living room, and it is dark enough, and
- the bathroom light turns on and the day's news is downloaded whenever the user gets out of bed early in the morning (after 6am).

The above two behaviours can be represented by two separate context-aware systems (R_1, M_1) and (R_2, M_2), respectively, each augmented with a module which maps situations or context to actions.

In more detail, we have $R_1 = (\Sigma_1, \Pi_1, \Theta_1)$, where

$$\Sigma_1 = \{light\_sensor, user\_positioning\_system\}$$

Π_1 would contain mappings of sensor readings to appropriate contexts of being dark

enough or whether the user is or is not in a room:

$$(reading(light\_sensor, t) \leq L,\ dark\_enough\_at\_t) \in \Pi_1$$
$$(reading(user\_positioning\_system, t) = living\_room,\ user\_in\_living\_room\_at\_t) \in \Pi_1$$
$$(reading(user\_positioning\_system, t) \neq living\_room,$$
$$user\_not\_in\_living\_room\_at\_t) \in \Pi_1$$

L is a threshold below which the light sensor reading indicates dark enough. and $\Theta_1 = (\Theta_{1c}, \emptyset)$, where given $(t2 - t1) < \epsilon$, for ϵ less than several seconds, we have that the user enters the living room at time $t2$ when s/he is not in the living room at time $t1$ and is in the living room shortly after at time $t2$:

$$(\ \{user\_not\_in\_living\_room\_at\_t1, user\_in\_living\_room\_at\_t2)\},$$
$$user\_enters\_the\_living\_room\_at\_t2\ ) \in \Theta_{1c}$$

Also, we turn the living room lights on when it is dark enough and when the user is detected to have entered the living room (assuming $t1$ and $t2$ close enough):

$$(\ (\{dark\_enough\_at\_t1\}, \{user\_enters\_the\_living\_room\_at\_t2\}),$$
$$turn\_on\_living\_room\_lights\ ) \in M_1$$

We also have $R_2 = (\Sigma_2, \Pi_2, \Theta_2)$, where

$$\Sigma_1 = \{bed\_sensor, clock\}$$

Π_1 would contain mappings of sensor readings to appropriate contexts of the user getting up from the bed:

$$(reading(bed\_sensor) \leq W,\ user\_off\_bed) \in \Pi_2$$
$$(reading(clock) > 6:00am, early\_morning) \in \Pi_2$$

W is a threshold below which the bed sensor reading indicates that the user is now off the bed, and $\Theta_2 = (\Theta_{2c}, \emptyset)$:

$$(\ \{user\_off\_bed, early\_morning\},$$
$$user\_up\_early\_morning\ ) \in \Theta_{2c}$$

Also, we turn the bathroom room light and download news when the user gets up in

the early morning:

$$((\emptyset, \{user\_up\_early\_morning\}),$$
$$turn\_on\_bathroom\_room\_light \ \& \ download\_news \ ) \in M_2$$

Note that this is a composite action.

We can extend the smart home to include safety features, with additional sensors and capabilities:

- the gas stove cannot be switched on without an adult present in the kitchen and the kitchen window opened
- smoke and appliance on, turn on the fan ventilators, and alert user
- broken glass, intruder detected, and raise an alarm

We illustrate a system (R_3, M_3) for only the first of the above features.

We have $R_3 = (\Sigma_3, \Pi_3, \Theta_3)$, where

$$\Sigma_3 = \{window\_sensor, user\_positioning\_system\}$$

Π_3 would contain mappings of sensor readings to appropriate contexts of the window being closed or a user (an adult) being in the kitchen. For simplicity, here, we assume that the user being tracked is an adult user:

$$(reading(window\_sensor, t) = open, \ kitchen\_window\_open\_at\_t) \in \Pi_3$$
$$(reading(user\_positioning\_system, t) = kitchen, \ user\_in\_kitchen\_at\_t) \in \Pi_3$$

We assume simply $\Theta_3 = (\emptyset, \emptyset)$, and map the contexts to the action (where $t1$ and $t2$ are close enough):

$$((\{kitchen\_window\_open\_at\_t1, user\_in\_kitchen\_at\_t2\}, \emptyset), enable\_gas\_stove \ ) \in M_3$$

Compositions. We have defined three context-aware systems with actions. A union of the three can be written as $R_1 \oplus R_2 \oplus R_3$. By definition of union, such a union represents a system that could recognize the combination of contexts and situations which the constituents can recognize, i.e. the union can recognize the set of contexts such as: $dark\_enough$, $user\_living\_room$, $user\_not\_in\_living\_room$, $user\_off\_bed$, $early\_morning$, $kitchen\_window\_open$ and $user\_in\_kitchen$, and the set of situations: $user\_enters\_the\_living\_room$, and $user\_up\_early\_morning$.

We can then augment the system with an action module M' to act on the combination of these contexts and situations, denoted as $(R_1 \oplus R_2 \oplus R_3, M')$. For example, in M', we have:

$$((\{dark\_enough, kitchen\_window\_open, early\_morning\},$$
$$\{user\_enters\_the\_living\_room\}),$$
$$display\_kitchen\_open\_window\_notice\_on\_tv)$$

which is an action to display a notice (about the kitchen window being open) on the television in the living room, when (i) it is still dark, (ii) the kitchen window is open, (iii) it is early in the morning, and (iv) the user enters the living room, The point of this example is that the context and situation conditions required for the action comes from all the three systems, rather than only a particular one.

Compositions to add reasoning. One could also form more "tightly-coupled" combinations using tight-union, e.g., $R_1 + R_2 + R_3 + R_4$, where R_4 is a newly introduced system of the form $(\emptyset, \Pi_4, \Theta_4)$ which comprises an interpreter and a situation reasoner but not sensors of its own. R_4 will leverage on the sensors already in R_1, R_2 and R_3 to acquire context and infer situations. For example, suppose that

$$(reading(bed\_sensor) \geq W',\ user\_on\_bed) \in \Pi_4$$
$$(reading(clock) > 11:00pm, late\_at\_night) \in \Pi_4$$

and

$$(\{user\_on\_bed, kitchen\_window\_open, late\_at\_night\},$$
$$home\_insecure\ ) \in \Theta_{4c}$$

then, the composition $R_1 + R_2 + R_3 + R_4$ can recognize a situation where the home is insecure.

This process can continue with incremental additions of functionality, creating increasingly complex composite systems. For example, R_5 of the form $(\emptyset, \Pi_5, \Theta_5)$ (for non-empty Π_5 and Θ_5) may be added to form the five system composition $R_1 + R_2 + R_3 + R_4 + R_5$, which has more recognition power than the previous composition, due to the breadth-monotonic property of tight-union, i.e. for any integer n, we have

$$R_1 + \ldots + R_n \quad \leq_{bm} \quad R_1 + \ldots + R_n + R_{n+1}$$

and this property follows clearly from the definition of tight-union.

Compositions to add sensors. Also, suppose new sensors are now added, represented by $R_6 = (\Sigma_6, \emptyset, \emptyset)$, and we want to determine if a situation S is occurring in the current world W, then we can pose a query of the form:

$$(R_1 + R_2 + R_3 + R_4 + R_5 + R_6) \vdash (W, S)$$

which will determine if S is happening using the tight-union of the six systems.

5 Composing Context-Aware Systems from Different Machines

We note that our operators can be used to compose systems on different machines; our model has been independent of the physical location of the component systems. An application of our work is in integrating context-awareness capabilities on the cell phone and the surrounding fixed infrastructure of sensors. Cell phones are changing from mere communication devices to sensing devices.[7] Future cellphones will be equipped with on-board sensors delivering information to users and interested parties. Some of these phones might be extensible with new sensors (as pluggable units). Our operators can then be used to effectively assemble a system based on different sensors available. One could think of some of the component systems as comprising only of sensors or comprising additional knowledge that can take advantage of new sensors which have been added (such as R_5) in the example above, and these components could reside on different hosts. For example, the following expression composes four systems, two systems available locally on the mobile phone and two other systems residing on hosts identified by the Web addresses $URL1$ and $URL2$:

$$(Local : R_1) + (Local : R_2) + (URL1 : R_3) + (URL2 : R_4)$$

We can implement the operators we have described the semantics of, by building each system as a set of loosely coupled components, and providing a front end to these systems, which realizes the semantics of the different operators. The future work section discusses this further.

[7] For example, see http://www.urban-atmospheres.net/ParticipatoryUrbanism/index.html and http://www.escience.cam.ac.uk/mobiledata/

6 Related Work

The recent years have seen much work in representing and reasoning with context and situations, and in context-aware systems (including the specialized workshop series on Context Modelling, Reasoning and Management (COMOREA)[8] and Modeling and Reasoning with Context (MRC)[9]).

A category of such work uses ontologies to help standardize the way context and situations are described (e.g., [3, 2, 24, 22, 7, 23]). Situations have been used as a key abstraction in context-aware systems such as in the work on situation hierarchies [25, 15]. Ontologies also provide a vocabulary of concepts to use, without the developer having to start from scratch. The work here is complementary to the use of such ontologies. We have assumed that two systems are comparable as the situations they recognize come from the same ontology or can be mapped to the same ontologies since the same situation or contexts can be described in different ways (even using different keywords).

There has also been work which uses simple first-order logic for reasoning about context and situations, and for conveniently representing rules that map context to actions (e.g., [20, 15]). While the rule-based declarative paradigm helps in representing knowledge about context and situations, in a way which facilitates reasoning, the work here presents a componentized coarse-granularity view of systems. Both our blackbox and whitebox models allow details to be encapsulated within each component system, albeit with the blackbox model exposing less internal functionality.

Fuzzy logic have been employed (e.g., [1]) as well as other ad hoc formalisms to deal with uncertainty (e.g., [18]). We have merely indicated how our operators manage uncertainty measures, and will deal further with different types of uncertainty in future work.

Middleware, frameworks and infrastructures for context-awareness have been developed in the past decade, including the Context Toolkit [21], the framework in [12], and others as reviewed in [11]. Such frameworks aim to simplify the development of context-aware applications by providing programming abstractions and reusable modules. Our work here aims to facilitate reuse and compositionality in a formal manner, via the operators, as well as to encourage development of context-aware pervasive systems in an incremental manner (where the system can be systematically grown). This paper aims to fill a gap as there has not been much work (to the author's knowledge) in attempting to specifically address the requirements on building context-aware per-

[8] See http://nexus.informatik.uni-stuttgart.de/COMOREA/2008/index.html
[9] See http://events.idi.ntnu.no/mrc2008/

vasive systems incrementally and in a formal compositional manner.

Another category of work relates to context-aware calculi (e.g., [5, 4]), providing formal operational specifications of interacting context-aware systems in the spirit of Milner's process calculi and bigraphical models. Such calculi are useful for formalizing context-aware behaviour but typically uses a process model of a context-aware system and do not have elaborate representations of context and situations.

7 Conclusion and Future Work

We have demonstrated an approach to building context-aware pervasive systems incrementally and in a compositional manner, facilitating reuse in a formal way. While there has been much work on software engineering composition of components and in various algebraic composition formalisms (e.g., [6]), we think that this is a first paper in attempting to provide a specialized formalism, tailored to an incremental and compositional way of building sensor-based context-aware systems.

An implementation of our language of operators in this paper can be based on the operational semantics given. An abstract architecture for a system which can be used to realize the above composition operators is illustrated in Figure 2. The idea is to have a front-end component which queries the n component systems according to the semantics of the composition operators. The architecture is abstract in that it does not mandate any particular representation for contexts, situations, or sensor values, nor do we mandate the use of any particular programming platform. We assume that this is a distributed system with the front-end networked to the component systems. An in-

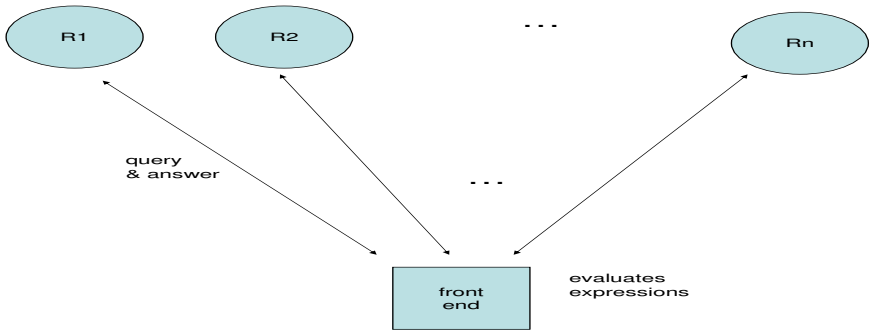

Figure 2: Architectural sketch of a system for realizing composition operators.

terpreter for expressions in our langauge resides on the front-end. On evaluation of

expressions, the front-end queries the respective systems via an interface realized by these systems. Each component system in composition should provide a three part interface so that the (i) sensors, (ii) interpreter and (iii) situation reasoner of each system can be exposed and be accessed by the front-end, especially to enable tight compositions. The interface can be specified loosely as a set of methods (or services) with arguments, and can be be specified as Web services which the front-end can invoke. Effectively, a membership check in the rules such as $(\{C_1, \ldots, C_m\}, S) \in \Theta_c$ translates into a Web service query to the system $(\Sigma, \Pi, (\Theta_c, \Theta_s))$. Future work will address the specific algorithms and architecture for building a distributed system that implements these operators, including optimizations. The issue of scalability is also a concern when possibly hundreds of systems are composed.

A situation is either occurring or not occurring. We have not explored unary operators analogous to negation. We think that failing to recognize a situation and recognizing the non-occurrence of a situation are different matters, and systems might need to be built which ascertains that a situation is not occurring. We intend to develop a more comprehensive theory of systems which recognizes situations and those which recognizes the non-occurrence of situations (but for now, indeed, the non-occurrence of a situation can itself be treated as a situation to be recognized). Furthermore, other operators can be explored that take into account temporal properties and uncertainty.

Acknowledgements. The author wishes to thank the reviewers for the detailed and insightful comments which helped greatly to improve the paper.

References

[1] 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.

[2] F. Ay. Context Modeling and Reasoning using Ontologies. 2007. Available at http://www.aywa.de/cmaruo/cmaruo.pdf.

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

[4] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical Models of Context-Aware Systems. In *Proceedings of Foundations of Software Science and Computation Structure (FoSSaCS)*, pages 187–201. Springer Verlag, 2006.

[5] P. Braione and G.P. Picco. On Calculi for Context-Aware Coordination. In *Proceedings of COORDINATION*, pages 38–54, 2004.

[6] A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini. Modular Logic Programming. *ACM Transactions on Programming Languages and Systems*, 16(4):1361–1398, 1994.

[7] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Knowledge Engineering Review*, 18(3):197–207, 2004.

[8] 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.

[9] K. Devlin. Situations as Mathematical Abstractions. In J. Barwise, J.M. Gawron, G. Plotkin, and S. Tutiya, editors, *Situation Theory and its Applications*. CSLI, 1991.

[10] A.K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[11] C. Endres, A. Butz, and A. MacWilliams. A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. *Mobile Information Systems*, 1(1):41–80, 2005.

[12] K. Henricksen and J. Indulska. Developing Context-Aware Pervasive Computing Applications: Models and Approach. *Journal of Pervasive and Mobile Computing*, 2(1):37–64, 2006.

[13] J. Hightower, B. Brumitt, and G. Borriello. The Location Stack: A Layered Model for Location in Ubiquitous Computing. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Washington, DC, USA, 2002. IEEE Computer Society.

[14] H. Liu, J. Srivastava, and S.-Y. Hwang. PSRA: A Data Model for Managing Data in Sensor Networks. In *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06)*, pages 540–547, Washington, DC, USA, 2006. IEEE Computer Society.

[15] S.W. Loke. Representing and Reasoning with Situations for Context-Aware Pervasive Computing: a Logic Programming Perspective. *Knowledge Engineering Review*, 19(3):213–233, 2004.

[16] S.W. Loke. *Context-Aware Pervasive Systems: Architectures for a New Breed of Applications*. Auerbach Publications, 2006.

[17] S.W. Loke. On Representing Situations for Context-Aware Pervasive Computing: six ways to tell if you are in a meeting. In *PerCom Workshops*, pages 35–39, 2006.

[18] A. Padovitz, S.W. Loke, and A. Zaslavsky. On Uncertainty in Context-Aware Computing: Appealing to High-Level and Same-Level Context for Low-Level Context Verification. In *Proceedings of the International Workshop on Ubiquitous Computing*, pages 62–72, 2004.

[19] G. Plotkin. A Structural Approach to Operational Semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004. Available at http://homepages.inf.ed.ac.uk/gdp/publications/sos_jlap.pdf.

[20] A. Ranganathan and R.H. Campbell. An Infrastructure for Context-Awareness based on First-Order Logic. *Personal Ubiquitous Computing*, 7(6):353–364, 2003.

[21] D. Salber, A.K. Dey, and G.D. Abowd. The Context Toolkit: aiding the Development of Context-Enabled Applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM.

[22] X.H. Wang, D.Q. Zhang, T. Gu, and H.K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 18–22, Washington, DC, USA, 2004. IEEE Computer Society.

[23] 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.

[24] J. Ye, L. Coyle, S. Dobson, and P. Nixon. Ontology-Based Models in Pervasive Computing Systems. *Knowledge Engineering Review*, 22(4):315–347, 2007.

[25] J. Ye, L. Coyle, S. Dobson, and P. Nixon. Using Situation Lattices to Model and Reason about Context. In *Fourth International Workshop on Modeling and Reasoning in Context (MRC 2007)*, pages 1–12, 2007. Available at http://www.cs.ucd.ie/UserFiles/publications/1183397975555.pdf.