Representing and Manipulating Situation Hierarchies using Situation Lattices

Juan Ye — Lorcan Coyle — Simon Dobson — Paddy Nixon

System Research Group, School of Computer Science and Informatics, UCD, Dublin, Ireland

{juan.ye, lorcan.coyle, simon.dobson, paddy.nixon}@ucd.ie

ABSTRACT. *Situations, the semantic interpretations of context, provide a better basis for selecting adaptive behaviours than context itself. The definition of situations typically rests on the ability to define logical expressions and inference methods to identify particular situations. In this paper we extend this approach to provide for efficient organisation and selection in systems with large numbers of situations having structured relationships to each other. We apply lattice theory to define a specialisation relationship across situations, and show how this can be used to improve the identification of situations using lattice operators and uncertain reasoning. We demonstrate the technique against a real-world dataset.*

RÉSUMÉ. *Les situations, les interprétations sémantiques du contexte, fournissent une meilleure base pour sélectionner des comportements adaptatifs que le contexte lui-même. La définition des situations repose typiquement sur la capacité de définir des expressions logiques et des méthodes d'inférences pour identifier des situations particulières. Dans ce papier, nous étendons cette approche pour fournir une organisation et une sélection efficaces à des systèmes avec un très grand nombre de situations entretenant des relations structurées entre-elles. Nous appliquons les treillis de Gallois pour définir une relation de spécialisation sur les situations, et nous montrons comment le résultat peut être utilisé pour améliorer l'identification de situations utilisant les opérateurs du treillis et le raisonnement incertain. La technique présentée est finalement validée sur un ensemble de données de taille réelle.*

KEYWORDS: *Situation, Lattice Theory, Context-aware Computing, Uncertainty, Bayesian Network*

MOTS-CLÉS : *Systèmes, Treillis, Systèmes contextuels, Incertitude, Réseaux Bayésiens*

RSTI - RIA — 22/2008. Modelling and Reasoning on Context, pages 647 to 667

1. Introduction

Pervasive computing attempts to make human lives simpler through digital environments that are sensitive, adaptive, and responsive to human needs (Saha *et al.*, 2003). Pervasive computing systems operate under dynamic and ever-changing environments, which require systems to react to dynamic changes in a seamless and unobtrusive manner. Context-aware computing is an enabling technology for pervasive computing. Context-aware computing systems provide adaptive services or behaviours according to different contexts. Any information that can be acquired from the environment can be considered as context (Loke, 2004). It can be sensed from physical sensors, profiled from users, or derived from application- or meta-information existing in systems. Context is acquired without any further interpretation and it may be meaningless, trivial, vulnerable to small changes, or uncertain. A system might not necessarily be expected to adapt its behaviour to each and every change of context. If the context is incorrectly reported, or is considered irrelevant to applications, a problem will occur when a system makes a responsive action in reaction to real-time contextual changes (Schilit *et al.*, 1994). Therefore, it is impractical to design behaviours that adapt directly to low-level context (Dobson *et al.*, 2007).

Situations capture the particular states that are interesting to applications. They are a semantic interpretation of context (Coutaz *et al.*, 2002; Dobson *et al.*, 2006). They are defined on contexts with a logical description that represents the invariant characteristics of contexts and their combinations (Weisenberg *et al.*, 2006). When individual pieces of context satisfy a situation's logical description, then this situation is considered to be *identified or occurring*. Context is concrete and trivial data from sensors, while situations are inferred and abstracted by evaluating context against a logical description. By abstracting contexts into situations, it is easier to resolve from imperfect context, capture meaningful contextual changes, and make it transparent to add or remove context sources. Therefore, situations are more meaningful, stable, and certain, so they are considered more crucial than individual pieces of context in determining a system's actions.

It is beneficial to define system behaviours on situations, and make any context or contextual change transparent. For example, a meeting detection application (e.g., Sensay (Siewiorek *et al.*, 2003)) should not be overly concerned with individual pieces of context, such as noise levels; rather it should concern itself with what the actual situation is – in this case, whether or not a meeting is taking place. A "meeting" situation can be composed with specific contexts: whether there are more than two people in a designated place; whether the current time is during office hours; or whether the ambient noise levels are high. When a new type of context is introduced, for example, the introduction of a calendar sensor that can detect when meetings are scheduled, then the situation specification is modified. However, its associated actions (e.g., change the mode of the attendances' mobile phones) will not be affected.

As the study of situations has become more popular, a huge number of situations are produced in an *ad hoc* way (for example as outlined in Section 2). In order to ben-

efit from using situations, it is necessary to analyse the internal relationships between situations. A situation can be decomposed into a set of smaller situations, which is a typical *composition or dependence* relation between situations. One situation can be considered more general than another situation, which is a *specialisation* relation: for example, a conference meeting situation is considered more specific than a meeting situation, because the conditions inherent in the conference meeting situation subsume or imply the conditions in a meeting situation. Alternatively, a situation may be required to precede another situation; that is, there is a *temporal order* between the situations.

Changes in situations may cause the system to adapt its behaviour, and in turn, this change in behaviour could lead to the generation of new contexts leading to new situations. Dealing with the rich internal relations between situations requires an efficient approach to organise situations, detect inconsistent logical descriptions between situations, and study the dynamic evolution of situations. These challenges are also identified as future work by Loke (2004).

This paper does not aim to provide a novel representation for situations: we use the typical representation – logical predicates. We focus instead on how to study the characteristics of situations by applying lattice theory. Situation lattices will be used to analyse the relationships between situations. They will help to maintain consistency and integrity when defining situations. This makes it possible to prevent runtime errors from occurring since the consistency and integrity constraints will already have been checked at design time. We also study the issue of uncertainty based on situation lattices.

The remainder of this paper is organised as follows. Section 2 introduces the current state of research in studying situations; Section 3 details the definition and construction of situation lattices; Section 4 analyses the dependence relationship between situations, illustrates how to maintain the consistency and integrity of situation descriptions, and demonstrates how to improve forward chaining with the use of the situation lattice. Section 5 discusses two approaches for dealing with uncertainties in situation identification, and provides the evaluation result using a real dataset. Finally, Section 6 draws a conclusion to the paper and outlines the future direction of this research.

2. Related Work

Past research on context-aware systems placed emphasis on modeling low-level context. More recently, the interest is on how to abstract, represent, and identify situations from the low-level context. Early attempts such as Gu *et al.*'s ontology-based model used first-order logical predicates to define situations (Gu *et al.*, 2004; Ranganathan *et al.*, 2004). These attempts simply composed context and situation with logical operators.

Yau *et al.* analyse the semantics of situations and gave them formal representations (Yau *et al.*, 2006). Context is considered as any instantaneous, detectable, and relevant property of the environment, the system, or users. A situation is a set of contexts over a period of time that is relevant to future device actions. A situation can be atomic or composite. An atomic situation is composed of contexts in terms of context operators, including function, arithmetic or comparison operators, and time constraints. The time constraints involve `forAny`, `exists`, `time-stamp`, `offset`, and `interval`. A composite situation is composed of atomic or other composite situations in terms of logical operators and time constraints. This helps application designers to specify situations using formal expressions. Yau *et al.* develop situation ontologies following these formal representations. This facilitates the analysis and specification of situation-awareness requirements of trustworthy ubicomp application software.

Costa *et al.* study the classification of situations in terms of their composition (Costa *et al.*, 2006). In their theory there exist three fundamental concepts: entity, context, and situation. Context is what can be said about an entity, and it cannot exist by itself. Context is characterised as either *intrinsic* or *relational*. An intrinsic context defines a type of context that belongs to the essential nature of a single entity and does not depend on the relationship with other entities. An intrinsic context is immediately derived from a single piece of context. A relational context associates multiple pieces of context in a certain relation. A formal relation is defined between two pieces of context directly without any intervenient entity, such as *greater than*, *subset of*, *nearness*, and *distance*. Situations model particular states of affairs that are of interest to applications. They are composite concepts whose constituents are entities, intrinsic and relational contexts, and formal relations. Situations can be composed of situations themselves. In addition, since a situation exhibits temporal properties, it is framed by a *chronoid* that defines a temporal duration. With this conceptual modeling, developers are able to model relevant changes in the state of affairs of a context-aware application's universe of discourse.

Yau *et al* and Costa *et al* elaborate the natural characteristics of situations, including the relationship between context and situations, the definition and composition of situations, and the internal relationship between situations. This provides a solid foundation of our work and inspires our idea of organising situations with respect to their internal relations.

Loke presents a declarative approach to representing and reasoning with situations at a high level of abstraction (Loke, 2004). A situation is characterised by imposing constraints on the output or readings returned by sensors. A situation occurs, when the constraints imposed on this situation are satisfied by the values returned by sensors. For example, a "in_meeting_now" situation occurs when a person is located with more than two persons and there is an entry for meeting in a diary. These constraints are represented as a logic program. This approach is based on the logical programming language LogicCAP that embeds situation programs in Prolog, which provides a high level of programming and reasoning situation for the developers. The logical theory

makes it amenable to formal analysis, and decouples the inference procedures of reasoning about context and situations from the acquisition procedure of sensor readings from context-aware systems. This modularity and separation of concerns facilitates the development of context-aware systems. Loke's work deals with an individual situation program, while our work focuses on organising a set of situations. He presents the logic definitions of completeness and soundness of a context-aware system, and pointed out that it is up to developers to verify soundness and completeness with respect to various systems. However, it is a hard task for developers, so we propose an approach to check these automatically during the procedure of creating situations.

Thomson *et al* provide a reusable library of situation specifications that helps to automatically determine situations (Thomson *et al.*, 2006). They express different levels of granularity of a situation through specification inheritance. New specifications are created as variations of existing ones so that the same situation can be interpreted at different levels of abstraction. We apply a similar approach to expressing situations through inheritance, however, the situation lattice we propose is a higher level structure that can be used to organise the specifications and further exploit richer characteristics in situations.

Most of the current work studies the composition of situations and formal representations. However, none of them have proposed a formal mechanism to organise the situations, based on which consistency and integrity can be maintained, and uncertainty issues are explored.

3. Situation Lattices

Situation lattices are inspired by Woods' use of lattice theory to recognise situations in linguistics (Woods, 1978). Building on the basic concepts of lattice theory, we will define the situation lattice and illustrate its construction from the ground.

3.1. *Lattice Theory*

A lattice is defined on a partially ordered set (poset). A poset is a set with a partial order, which is a binary relation R over a set P that satisfies reflexivity, antisymmetry, and transitivity. A poset (P, $\leq$) is a *lattice* if for any pair of elements in P, there exists the least upper bound (join) and greatest lower bound (meet) for them in P.

Lattice theory is useful studying the structures with a partial order. It has many practical applications in distributed computing such as the works done by Charron-Bost (1991), Tarafdar *et al.* (1999), and Garg *et al.* (2003). Garg *et al.* applied lattice theory to analyse the partial order between traces of a distributed program. This is used to determine whether this program satisfies a given temporal logic formula, to detect a predicate in a computation, and to compute the slice of a computation with respect to the predicate.

Lattice theory is also applied in the *Formal Concept Analysis* (FCA) (Ganter *et al.*, 1997). FCA is a method for data analysis, knowledge representation and information management. In FCA, the data are structured into a formal abstraction – a concept that is constituted by its extension and intension: its extension is the collection of objects belonging to the concept and its intension is the collection of all attributes common to its extension. The *concept lattice* organises the formal concepts with respect to the subconcept-superconcept relation. This concept order is based on a coupled extensional and intensional order. A concept A is called a subconcept of B if and only if the extent of A is a subset of that of B, and the intent of B is a subset of that of A. The concept lattice is used to unfold data, making their conceptual structure visible and accessible. This is helpful in finding patterns, regularities and exceptions in data.

3.2. *Construction of a Situation Lattice*

A context-aware computing system usually attempts to gather all the contexts from available sensors. These contexts can be meaningless or imperfect so that they cannot be directly used to trigger a system's new behaviour or service. They need to be composed and decoded to a set of meaningful and stable elements, *situations*, which abstract and refine the contexts. In this section, we will construct a situation lattice in which situations are organised in terms of their internal relationship. The essential characteristics of the situation lattice are the ability to support the reuse of logical descriptions between situations and represent situations in various levels of abstraction.

Each situation is characterised with a logical description that takes context or other situations as input and that defines logical operators between them. A logical description takes the form of $l(t_1, \ldots, t_m)$, where t_i can be a context predicate, or a situation. Here, a context predicate represents a piece of context as a first-order logic predicate (Ranganathan *et al.*, 2004), e.g, $hasLocation(Erica, UCD)$, indicating a person named "Erica" is located on campus "UCD". A situation is regarded *occurring or identified*, iff its logical description is satisfied by the input context. The specialisation relationship of situations is defined as follows.

Definition 1. *A situation $s_i \in S$ is **more specific** than another situation $s_j \in S$, labelled as $s_i \leq s_j$, iff s_i's logical description l_i entails s_j's logical description l_j, labelled as $l_i \vdash l_j$.*

The above definition implies that when the logical description of s_i is satisfied, then the logical description of s_j will be satisfied as well. Therefore, the identification of a situation s_i implies the identification of its more general situation s_j. Alternatively, s_j is called a **more general** situation than s_i. If two situations do not have a specialisation relationship between them, then they are called *disjoint* situations.

A particular case of the specialisation relationship is the *immediately more specific* relation. If a situation s is defined directly under a situation s', then it is called

immediately more specific than s'. If a situation s is defined as an immediately more specific situation from a set of disjoint situations $\{s_1, \ldots, s_n\}$, then its complete logical description l will inherit the descriptions from these situations and be extended with a new description, which is written as $l = l_1 \wedge \ldots \wedge l_n \wedge l^*$, where l_i is the complete logical description of a situation $s_i (1 \leq i \leq n)$ and l^* is an extended logical description particular to the situation s. When creating a situation, only the immediate specialisation relationship is defined between situations, while the general specialisation relationship can be derived from the former by transitivity.

We made some assumptions on the constructed situation set. Among all the defined situations S,

– there does not exist any two situations that have the same complete logical description. $\forall s_i, s_j \in S$, if $l_i = l_j$, then $s_i = s_j$.

– there exists a unique top situation $s_\top$, which is the most general situation. That is, $\exists s \in S$, if $s_\top \leq s$, then $s = s_\top$. This top situation is identified when a system is running properly. Its logical description is the tautology T.

– there exists a unique bottom situation $s_\bot$, which is the most specific situation. That is, $\exists s \in S$, if $s \leq s_\bot$, then $s = s_\bot$. This bottom situation is identified when a system is unstable or is running improperly. Its logical description is the contradiction F.

Before we define a situation lattice, we will demonstrate that the specialisation relationship is a partial order.

Proposition 2. *This specialisation relationship is a partial order.*

Proof. We need to prove that this specialisation relationship satisfies *reflexivity*, *anti-symmetry*, and *transitivity*.

– *Reflexivity* Given a situation s, its logical description satisfies $l \vdash l$, So, $s \leq s$.

– *Anti-symmetry* Given two situations s_i and s_j, if $s_i \leq s_j$, then $l_i \vdash l_j$, so there exists some logical description l' such that $l_i = l_j \wedge l'$; if $s_j \leq s_i$, then $l_j \vdash l_i$, so there exists some logical description l'' such that $l_j = l_i \wedge l''$. Then $l_j = l_j \wedge l' \wedge l''$, so $l' = l'' = T$, when s_i (or s_j) is not the unique top situation. Thus $l_j = l_i$, so $s_i = s_j$.

– *Transitivity* if $s_i \leq s_j$, then $l_i \vdash l_j$; if $s_j \leq s_k$, then $l_j \vdash l_k$. $l_i \vdash l_k$, so $s_i \leq s_k$.

□

Definition 3. *A situation lattice, L, is defined as $L = (S, \leq)$, where S is a set of situations and the partial order $\leq$ is a specialisation relation between situations.*

Since this is a lattice, there exists a join and meet for any pair of situations in S. If this pair of situations have the specialisation relationship between them, then their join is the more general one while their meet is the more specific one. If this pair of situations are disjoint with each other, then their join is the most specific one among all of their more general situations, while its logical description contains the disjunction

of that of these two situations. Their meet is the most general one among all of their more specific situations, while its logical description contains the conjunction of the descriptions of these two situations.

3.3. *An Example of a Situation Lattice*

This section demonstrates an example of how to construct a situation lattice as shown in Figure 1. This situation lattice aims to describe the in-office activities of a postgraduate student. The situations are abstracted from the context currently available in our environment. There are three sensors producing context:

– *Ubisense* is a tag-based positioning sensor network, which is used to track an object's real-time location in an indoor environment. Ubisense provides a precise location of a person in the form of coordinates. The coordinates can be mapped to a place with a human-understandable name.

– *Activity sensors* sense the activities of keyboard and mouse. If the keyboard or mouse is used, then this sensor will send an "active" state to indicate that the computer is being used at the moment. The frequency that it sends the data implies the states of the computer being used.

– *Calendar sensors* capture the scheduled events in the personal and group calendars. An event can be coarse-grained, for example, a person is on holiday during a period; while it can also be fine-grained, for example, a meeting event is scheduled, by specifying its attendees, starting time, end time, location, and content.

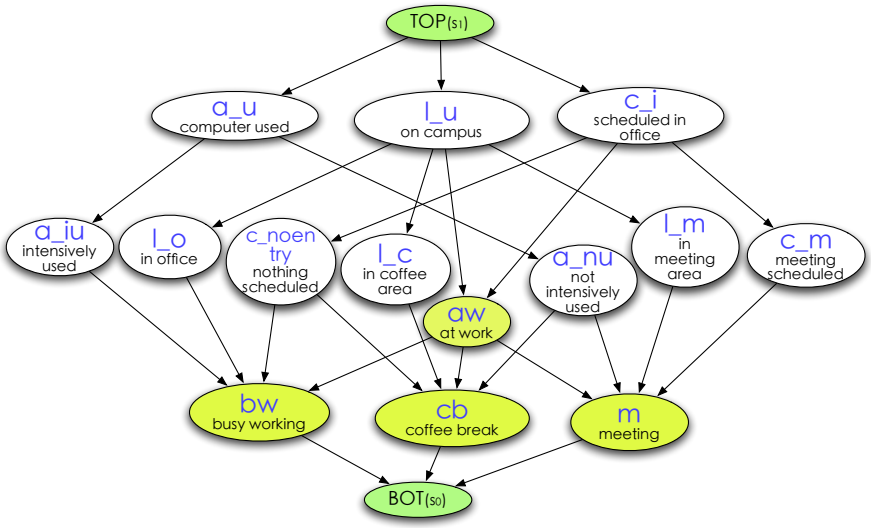

Figure 1. *An example of a situation lattice*

From these contexts, we can create a few *basic* situations, which interpret single pieces of the context. They are immediately under the top situation $s_\top$. For example, the "on campus" situation l_u indicates that the location of this person is on campus. Basic situations can be constrained and composed into *composite* situations. For example the "at work" situation aw indicates that the person is at work, which is composed of basic situations: the "on campus" situation l_u and the "scheduled in office" situation c_i. The conjunction of their logical descriptions is contained in the logical description of aw. Also the basic situations can be refined; for example, the situation l_u is refined to be the " in office" situation l_o, indicating that a person is in her office, while her office is contained in the university. Its logical description extends that of l_u with a new one: $(\exists o)(hasOffice(p, o) \wedge isContainedIn(lo, o))$.

Given more detailed contexts, more specific situations are generated. For example, the "busy working" situation bw indicating that a person is busy at working, when her computer is intensively used (a_iu), she is located in her office (l_o), and her calendar does not have scheduled events at the moment ($c\_noentry$).

The meet of a pair of situations encapsulates the least conjunction of logical descriptions of these situations. This is the most general situation whose logical descriptions are reused by their more specific situations. For example, the meet of the situations l_u and c_i is the "at work" situation aw, which can be specialised into more specific working situations, like the "busy working" situation bw and the "meeting" situation m.

The join of a pair of situations encapsulates the greatest overlapping logical description of these situations, which is the most specific situation above them. For example, the join of the computer being "intensively used" situation a_iu and the computer being "not intensively used" situation a_nu is the "computer used" situation a_u. The situations a_nu and a_iu are covered (or contained) in the latter situation a_u.

4. Analysing Situation Lattices

This section will demonstrate that the situation lattice helps to explore the *dependence* relationships of situations, to maintain the consistency and integrity of situations, and to improve the efficiency of a forward chaining algorithm.

4.1. *Exploring Dependence Relationships Between Situations*

A dependence relation between situations is discussed in most context modeling research by Gu *et al.* (2004) and Henricksen *et al.* (2004). This relationship can also be reflected in the situation lattice.

The situation lattice is regarded as a specialisation relationship between situations if it is observed downwards from the top. A situation $s \in S$ is more *specific* relative

to all of the situations above it. It can also be considered as a dependence structure if it is observed upwards from the bottom. A specific situation can be decomposed into a few more general situations. Its satisfiability *depends* on the evaluation of the satisfiability of all its more general situations.

Example 4. *In Figure 1, the satisfiability of a "meeting" situation m depends on that of its constituent situations: the "meeting scheduled" situation c_m ; the "at work" situation aw; the "in meeting area" situation l_m; and the computer being in the "not intensively used" situation a_nu. Upwards, the satisfiability of c_m depends on that of a more general situation c_i that the calendar records that a person is working in her office, whose satisfaction depends on that of the top situation.*

The top situation stores all the proper states of a system, and it holds if a system is running properly. Therefore, the appropriateness of a system is the precondition of the satisfaction of any basic situation. Corresponding to the top situation, the bottom situation $s_\perp$ stores all the improper states of a system, and it holds if there is anything wrong with the system. Therefore, the bottom situation holds if inconsistent situations are detected. For example, $s_\perp$ will be identified if two situations bw and m are regarded occurring at the same time for the same person.

4.2. *Maintaining Consistency and Integrity*

Context-aware computing systems typically involve a large quantity of context, based on which a number of situations can be created and specified. The question is: how can situations be kept consistent and integral? Consistency means that logical descriptions should be compatible between non-disjoint situations. Section 3.2 has defined that a specific situation inherits the logical descriptions from its more general situations. To maintain consistency the extended logical description of the specific situation should not conflict with any description of its more general situations.

Example 5. *In Figure 1, considering that the "meeting" situation m is one of the more specific situations of the "at work" situation aw, it is impossible that the logical description of m conflicts with that of aw.*

Integrity means that non-conflicting context data cannot result in conflicting situations. A situation s_i is conflicting with another situation s_j, if $l_i \wedge l_j = F$, where F is a contradiction. This requires that for any situation, the logical descriptions of its immediately more specific situations should conflict with one another. Thus given the correct context, there exists the unique most specific situation among all the identified situations.

Example 6. *Given non-conflicting context data, if the logical description of the "busy working" situation bw is satisfied, then the description of the "coffee break" situation cb should not be satisfied. These two situations are conflicting, since their more general situations a_iu and a_nu are conflicting. The descriptions of these two situations*

are opposite. a_iu holds if the time gap between two continual readings is less than 30 seconds; while a_nu holds if the time gap between two continual readings is not less than 30 seconds. Therefore, the conflicting situations bw and cb should not be identified at the same time for the same person.

Consistency and integrity are the necessary conditions if a system aims to identify a situation appropriately. Once the errors of inconsistency and non-integrity are detected at runtime, the system designers will be forced to rewrite logical descriptions. This repetitive checking and modification is expensive; therefore, it would be advantageous if these problems could be spotted and avoided when creating situations. From the top situation, each of its immediately more specific situations should not only satisfy logical descriptions of the top situation, but also contain logical descriptions exclusive from that of other siblings. This checking will be conducted recursively through the whole process of construction. In Section 3.2, a new situation s is specified in a logical description: $l = \bigwedge_{i=1}^{m} l_i \wedge l^*$, where l_i is the logical description of one of its immediately more general situations s_i. If l is evaluated to be false, then there is a part of l^* conflicting with l_i, which implies that s breaks the consistency requirement. The integrity will be checked by comparing l with any logical description l_j of its sibling situations that share the same immediately more general situations. The situation s is considered as an acceptable situation if its extended logical description is different from that of its siblings: $l \neq l_j$. This procedure is described in Algorithm 1.

4.3. *Identifying Situations*

There are two ways of recognising a situation. **Backward chaining** starts with a list of situations and works backward to see whether the available context supports the requirements of any of the situations. Backward chaining is a typical mechanism used in current context-aware computing systems. To identify a "meeting" situation, a system will collect all the perceptible context, for example, scheduled events in the calendar and a person's location. If the context satisfies the logical descriptions of a "meeting" situation, then it is identified. This backward chaining is useful only when a situation to be determined is chosen beforehand.

In the situation lattice, the logical description is defined particularly for each situation, and increasingly inherited from its general situations. Backward chaining is carried out by evaluating this incrementally formed logical description against the given context.

In many real applications, where there are many possible situations, it is not always practical to locate a situation beforehand. Thus, **forward chaining** should be used: this starts with the acquired context and applies inference rules to arrive at a situation. In this circumstance, faced with a large number of inference rules, it is infeasible to find the rules that match a certain situation by systematically checking each rule. It is necessary to find out a way of reducing the computational load and locating a situation efficiently. In the situation lattice, logical descriptions can be shared, which avoids

Algorithm 1 Create a situation $create(S, l^*)$

INPUT: a set S of disjoint situations; an extended logical description l^*
OUTPUT: a situation s being created; otherwise, return null
// check the consistency of the logical description between the input situations
$l = true$
for all $s_i \in S$ **do**
  // l_i is the corresponding logical description of a situation s_i
  $l = l_i \wedge l$
  if evaluate l to be false **then**
    fail
  end if
end for
// check the consistency between the extended logical description and the inherited description
if evaluate $l \wedge l^*$ to be false **then**
  fail
end if
// check the integrity
for all $s_i \in S$ **do**
  for all s_i': immediately more specific than s_i **do**
    if l^* is equal to the extended description of s_i' **then**
      fail
    end if
  end for
end for
return s

repetitive evaluation of situations. The forward chaining does not have the problem of infinite loops in the situation lattices either.

The situation lattice will be suitable for the forward chaining. A system starts by identifying basic situations from the given set of context. Only the logical description l^* particular to a situation will be checked, rather than its complete logical description. If the description is satisfied, the satisfied context will be removed from the original given context set and the chaining will continue checking its more specific situations. In this way, only the minimum descriptions will be evaluated every time without repetition, and the given context set is reduced continually. This will reduce the computation load and improve the efficiency. Algorithm 2 describes the process of forward chaining that will end up with a set of situations all of which are identified. Further, another procedure, *refine*, will locate a set of the most specific situations $\{s_i, \ldots, s_k\}$ among the returned situations. If this refined set is a singleton, then the target situation is the single situation; otherwise, the target situation is the join of all these situations.

Algorithm 2 forward(s, C, S)

INPUT: a situation s being evaluated; an input context set C; a set S including all the qualified situations.
OUTPUT: a set of qualified situations
if $S.contain(s) = false$ **then**
  // evaluate the extended logical description l^* of s against C
  if $eval(l^*, C) = true$ **then**
    $S.add(s)$
    // reconfigure C by removing the context data satisfied by l^*
    $C' =$ reconfig(l^*, C)
    for all s_i: the immediately more specific situation than s **do**
      if $s_i \neq s_\perp$ **then**
        $forward(s_i, C', S)$
      end if
    end for
  end if
end if

5. Situation Lattices and Uncertainties

When dealing with real-world data, there is no guarantee that situations will be identified with complete certainty. Data can be imperfect, for example, due to sensor failure, noise, delays, disconnected sensor networks, or infrequent update in response to changes (Henricksen *et al.*, 2004). Context is considered uncertain, if it is

– *incomplete*, when some information is unknown or missing. There may not be enough evidence to determine the correct situation;

– *imprecise*, when the resolution of the context cannot satisfy the requirement of applications. A more specific situation may not be able to be located;

– *conflicting*, when there are several inconsistent pieces of information from different sources, which may result in multiple disjoint situations being determined;

– *incorrect or meaningless*, when the information is erroneous compared to the actual state or reality, which may result in an incorrect situation being determined;

– and *out-of-date* when the information is stale and is not updated in response to changes, which may result in an incorrect situation being determined.

Many of these uncertainties are amplified when using inference rules to reason about context, as well as the typical insensitivity of rules to noisy inputs. Another concern is the difficulty in defining and maintaining accurate inference rules. These uncertainties can result in incomplete, inconsistent, and incorrect situations being identified. In the following, we will propose two approaches to resolve uncertainty in the situation lattice.

5.1. *Coarse-Grained Approach to Resolving Uncertainty*

A coarse-grained approach is introduced to resolve uncertainties with respect to the characteristics of a lattice. Compared to a specific situation, a general situation has fewer or looser requirements (or conditions). The general situation can be constrained to more specific situations by

– adding requirements in its logical description (e.g., the "at work" situation aw can be considered as being defined by adding the location requirement – the "on campus" situation l_u to the "in office" situation c_i);

– tightening the constraints (e.g., the computer being "intensively used" situation a_iu is extended from the computer being "used" situation a_u with the constraints on the frequency of receiving the readings from the activity sensor);

– or uniting with other situations (e.g., the "busy working" situation bw is defined by composing the "not intensively used" situation a_iu, the "in office" location situation l_o, aw and the "nothing scheduled" calendar situation $c\_noentry$).

If some context is too incomplete or too imprecise to support a given situation then this situation cannot be identified. However, its more general situations will be checked upward the situation lattice until a certain situation is identified, whose logical descriptions are satisfied by the context. Therefore, when a system fails to recognise a specific situation, it can loosen the requirement to locate a more general one.

If the input context is contradictory, then conflicting situations will be generated. Each of these situations satisfies a part of the given context. It is difficult for the lattice to determine which part of the context is proper if there is no clue about the reliability of each piece of the context. As a result, the join of these situations will be returned to resolve the inconsistent uncertainty.

The system is kept stable using the coarse-grained approach because it always tends to choose the inviolable situation, even though this is not always the most appropriate situation. In the pathological case, when all of the derived disjoint situations are conflicting, the join of them is the most general situation $s_\top$. That implies the system does not detect any situation and will not take any particular behaviour, so it is considered insensitive to situations or context. However, among the disjoint situations, if uncertainties of context were incorporated into the lattice, it might be appropriate to select the situation with the highest degree of confidence. The system should then carry out the behaviours specified for that situation. This responsive system is more suitable for real-world applications. Consequently, we propose a fine-grained approach to quantify the confidence of generated situations, which helps to determine the situation that is most likely to occur.

5.2. *Fine-grained Approach to Resolving Uncertainty*

The typical fine-grained approach attempts to quantify the uncertainties underlying situations using probabilities. These probabilities attempt to capture the uncertainty caused by imperfect context and error-prone deriving mechanisms. The situation lattice represents the dependence relationship between situations, so a promising approach is to represent the probabilities on both situations and dependence relationships and then reason on them with these probabilities. Bayesian Networks have a causal semantics that encode the strength of causal relationships with probabilities (Heckerman, 1996).

Bayesian networks are usually used to calculate the probabilities for decision making under uncertainty. A Bayesian network is a directed acyclic graph in which each node represents a variable that can be discrete or continuous, and each arc is the causal relationship between nodes. If there is an arc from a node A to another node B, then A is called a *parent* of B, implying that the variable B is regarded as depending directly on A. If a node does not have a parent, then it is called *root*. Each root node is associated with an *a priori* probability. Each non-root node is associated with a conditional probability distribution (CPD). If the variables are discrete, then the CPD is represented with a conditional probability table given all possible combinations of their parent nodes: $p(x \mid parent(x))$, where $parent(x)$ is a parent set of a node x.

It is obvious that a situation lattice has a very similar structure to a Bayesian network. The lattice can be converted to a Bayesian network in a straightforward manner: each node in a Bayesian network corresponds to a situation, and each arc to a dependence edge. In this Bayesian network, the root nodes are considered basic situations that are immediately under the top situation $s_\top$. After building the graphical model of Bayesian network, we will assess the prior probability for each root node and the conditional probability for each non-root node. The Bayesian probability of an event is a degree of belief in this event (Heckerman, 1996) and it can be obtained from the domain expert or observations.

Considering the uncertainty and dynamism, the probabilities will be evaluated by training a set of real data. For the probability of a root node, a simple but straightforward approach is $p(\omega) = \frac{N'}{N}$, where N' is the times that a certain state ω takes place and is recognised, and N is the total number of observations. To simplify the computation, it is assumed that the structure of the model is known and the full observations are possible, so the *maximum likelihood estimate* is applied for the conditional probability distribution (Murphy, 1998). For each non-root node s, one of its discrete states is written as ω, its parent nodes are $s_1, \ldots, s_n$, and one of its conditional probability is calculated as follows:

$$p(s = \omega \mid s_1 = \omega_1, \ldots, s_n = \omega_n) = \frac{N(s = \omega, s_1 = \omega_1, \ldots, s_n = \omega_n)}{N(s_1 = \omega_1, \ldots, s_n = \omega_n)},$$

where $N(s = \omega, s_1 = \omega_1, \ldots, s_n = \omega_n)$ is the number of times that s is recognised

in one of its states ω, and all of its parents are in one of its own states ω_i; and $N(s_1 = \omega_1, \ldots, s_n = \omega_n)$ is the number that all of its parents are in one of its own states ω_i.

Bayesian inference is the process of updating the probabilities based on the relationships in the model and the recent evidence. The new observation is applied to the model by assigning a variable to a state that is recognised from the observation. Then the probabilities of all the other variables that are connected to this variable will be updated. The new probability is called *posterior* probability that reflects the new levels of belief.

Under the conditional independence assumption, the joint probability distribution is applied to compute the probability of the resultant situations given the causal situations: $p(s_i = \omega) = \prod_{k=1}^{n} p(s_k \mid parent(s_k))$.

With Bayesian networks, a system will not only return a more general situation through the above coarse-grained approach, but it will also return a specific situation with the highest possibility. If the highest possibility is beyond the threshold that is specified by a system, the behaviours corresponding to that situation will be carried out.

5.3. *Demonstration and Evaluation*

To demonstrate this work, we have gathered context data from three sensors that capture the movements of a research student over a period of five working days. Over the same period, the student kept a diary of her on-campus behaviour, capturing whether she was working, in meetings, or on coffee breaks. The entries in the student's diary record her activities like this: "working at my desk from 09:36:30 to 10:03:55 On 23rd Oct 2007" [1]. The situation lattice in Figure 1 was used to order the situations. We then use a Bayesian approach to handle uncertainties in the mappings between contexts and situations in this Lattice. Conditional probabilities, which capture uncertainties are learned using the diary data as a ground truth. The situation lattice is then tested using held-back diary data to evaluate the approach and show that it was capable of learning the appropriate mapping between raw contexts and application-usable situations. For the purposes of demonstration, we will show how the conditional probabilities were calculated for the "busy working" situation, which captures whether the user was sitting at their desk working.

As introduced in Section 3.3, the "busy working" situation is informed by the "intensively used", "in office", and "nothing scheduled" contexts. It is also informed by the "at work" situation in the lattice. The "busy working" situation cannot be true if the "at work" situation is not also true. In this sense the "busy working" situation is indirectly dependent on the "on campus" and "scheduled in office" contexts, but for

1. The sample data are published online here: `http://kind.ucd.ie/~juanye/datasets/Ria2008Dataset.zip`.

the purposes of this demonstration we will assume that the mapping between these contexts and "at work" situation have already been learned.

Each of the three contexts that inform "busy working" have associated uncertainties that must be learned. The "in office" context is generated from a Ubisense installation. Ubisense does not always provide accurate location data, due to its own technical limitations including the install environment, the battery level of tags, and the sample frequency (Coyle *et al.*, 2007). For example, if a Ubisense tag is located in the environment where there is interference (which is the case around our test user's desk), then its accuracy will be decreased. The "intensively used" situation is informed by an activity sensor that indicates the level of activity recorded at a computer by a logged-in user (it monitors the time of the last keystroke). If the last keystroke was recent (in the last 30 seconds) then the "intensively used" context will be true, otherwise it will be false. The "nothing scheduled" context is informed from an sensor that detects the presence (and type) of events scheduled in a person's online calendar. If there is no entry scheduled, this context will be false. Uncertainty from this sensor comes from the fact that meetings are often rescheduled, do not occur on time, or are not explicitly scheduled in a calendar. One issue for this context in our dataset is that not many meetings were scheduled — so this context has limited predictive power.

We calculate the conditional probabilities for the "busy working" situation using maximum likelihood estimation. We sample ground truths for the "busy working" situation from the diary at thirty second intervals and use the available context at those times to calculate the conditional probabilities. The conditional probabilities calculated on the whole dataset are shown in Table 1. The conditional probability column also contains the number of times this context combination occurs in the dataset (i.e., at no time in the dataset is the "intensively used" true when both the "nothing scheduled" and "in office" contexts are false). The probabilities of identifying the "busy working" situation tend to be higher if there is support from multiple sensors, for example, if all three contexts support the "busy working" situation the conditional probability is 0.99.

nothing scheduled	in office	intensively used	probability(busy working)
false	false	false	0.0 (94)
false	false	true	0.0 (0)
false	true	false	0.87 (23)
false	true	true	0.97 (64)
true	false	false	0.14 (431)
true	false	true	0.99 (241)
true	true	false	0.88 (1000)
true	true	true	0.99 (1367)

Table 1. *The conditional probabilities of the "busy working" situation. The total number of times each context combination occurs is in brackets*

We use the same approach to learn the conditional probabilities for each situation in the lattice. In order to test the accuracy of the situation lattice we divide the dataset into training and test data: we use the training data to calculate conditional probabilities and test data to evaluate the approach's accuracy. We use cross validation when producing our evaluation, using four days to calculate conditional probabilities, and use the data from the fifth day as a test fold to calculate the accuracy. Table 2 shows the results for each of the five test days. For example, the probability 0.91 is the Bayesian network's accuracy in predicting the "busy working" situation correctly with the fifth day (using the data from the first, second, third, and fourth days as training data). Using this dataset, we can see that the overall accuracy for the situation lattice is 0.91.

Day	1	2	3	4	5
busy working	1.0 (397)	1.0 (609)	1.0 (291)	1.0 (683)	0.91 (596)
coffee break	0.92 (66)	0.86 (88)	0.84 (57)	0.70 (99)	0.93 (89)
meeting	0.0 (0)	0.0 (0)	0.0 (0)	0.96 (91)	0.88 (8)
Overall	0.99 (463)	0.98 (697)	0.97 (348)	0.96 (873)	0.91 (693)

Table 2. *The probabilities of correctly identifying each of the situations in the cross validation tests. The number of times each situation actually occurs in in brackets*

This dataset is used as a proof of concept for learning the conditional probabilities of the situation lattice, but it has quite a few limitations — there is no data for meetings on three of the days, and there is very little data for coffee breaks, which include lunch (only one day has more than an hour of data for this situation). The vast majority of data tested is of a single situation — "busy working", which leads to a bias in favour of that situation and probably skews the overall accuracy upwards. Since the lattice presented here only contains three situations, the training and testing ignores any other situation that occurs in the diary. The diary records just over 15 minutes of these miscellaneous situations per day (these situations include travelling to the printer and visiting other desks). Our next steps will involve gathering a larger dataset, covering more days, with additional situations, which will include these miscellaneous situations, as well as new situations, covering specific types of meetings, lectures, and an out-of-office situation. We expect this additional complexity to reduce the accuracies presented here and force us to make changes in the lattice structure presented here.

6. Conclusion and Future Work

As reasoning with situations becomes more popular, system designers tend to specify a large number of rules to identify various situations in an *ad hoc* way. An efficient approach is needed to organise and manage these situations so that their logical descriptions maintain the consistency and integrity requirements. This paper applies a formal structure using lattice theory to organise situations.

The situation lattice reflects the specialisation relation of situations and captures the dependence between situations. We believe that it will be helpful when maintaining the consistency and integrity of situations, however, the involved computation may be huge when faced with a large number (for example, hundreds or thousands) of situations. This paper only presents a simple situation lattice with a limited number of situations, while we will attempt to improve the algorithm to make the checking procedure scalable and efficient. The situation lattice will also be beneficial when identifying the situations using forward chaining approaches. However, the situation lattice only reflects the static structures of situations. We have discussed the dynamic evolution between context changes and situation transitions with a fibration theory in our earlier work (Dobson *et al.*, 2006). In the future, we will investigate how situation lattices and fibration can be made to work together.

In dealing with the issue of uncertainty, the situation lattice supports a coarse-grained approach and a fine-grained approach. The structure of the situation lattice has a natural association with the Bayes' Principle, which makes it amenable to the uncertainty resolving techniques such as Bayesian Networks or Hidden Markov Models. This paper demonstrates how the situation lattice works with Bayesian networks, and shows preliminary results that demonstrate that Bayesian networks can accurately identify a single situation from a small set of possibilities.

Bayesian networks work well if the context sources are relatively fixed, situations are limited to a small number, and the structure of Bayesian networks is known *a priori*. However, this assumption is contradictory to the nature of context-aware computing systems. For the acquisition of context, context-aware systems should watch all the potential context in the environment. This is a big issue when applying these systems in reality, and potentially not solvable at the current research stage. In these environments, new context sources often enter and leave. This frequent churn in context sources will quickly render the original Bayesian network useless and require the system to frequently retrain itself. If there are a large number of nodes in the Bayesian network, the cost of training will be prohibitive.

The promise of using Bayesian networks with situation lattices is that they could be used to learn the underlying structure of situations, which would make it possible to reconfigure the situation lattice. If learning the structure of nodes is required, the NP-hard problem underlying the Bayesian network will become an obstacle (Charniak, 1991). Considering the above disadvantages, we will design the algorithms to optimise the performance of Bayesian networks based on the particular characteristics of context-aware computing systems. One solution may be to explore the use of more knowledge about the environment (e.g., using a location model Ye *et al.* (2007), an activity model Wren *et al.* (2007), or social network information Bottazzi *et al.* (2007)), which may be used to learn the structure of Bayesian network and provide prior probabilities. This may help to decrease the amount of training data required.

To move towards the real situations, we will apply our approach to a larger dataset that is gathered in our environment and a third party dataset (that is, the PlaceLab

dataset (Logan *et al.*, 2007) that comprised of a set of common household activities over a period of time). These larger datasets will require the size of a situation lattice to be hundreds or thousands, which will increase the complexity of training a Bayesian network, predicting situations and constructing a situation lattice.

When there are a large number of situations involved, it is infeasible to build a situation lattice manually. To decrease the complexity of construction, we will attempt to design a tool that encapsulate and automate the following three processes: generating a situation lattice from the initial input by developers; converting it to a Bayesian network; and training it with initial sample data to arrive at an applicable situation lattice. This tool will be expected to ease the procedure of constructing a situation lattice for developers.

Acknowledgements

This work is partially supported by Science Foundation Ireland under grant numbers 05/RFP/CMS0062 “Towards a semantics of pervasive computing” and 04/RPI/1544 “Secure and predictable pervasive computing”.

7. References

Bottazzi D., Montanari R., Toninelli A., “ Context-Aware Middleware for Anytime, Anywhere Social Networks”, *IEEE Intelligent Systems*, vol. 22, n° 5, p. 23-32, 2007.

Charniak E., “ Bayesian networks without tears: making Bayesian networks more accessible to the probabilistically unsophisticated”, *AI Magazine*, vol. 12, n° 4, p. 50-63, 1991.

Charron-Bost B., “ Concerning the Size of Logical Clocks in Distributed Systems”, *Information Processing Letters*, vol. 39, p. 11-16, July, 1991.

Costa P. D., Guizzardi G., Almeida J. P. A., Pires L. F., van Sinderen M., “ Situations in Conceptual Modeling of Context”, *Proceedings of EDOCW’06*, IEEE Computer Society, Hong Kong, China, p. 6-16, October, 2006.

Coutaz J., Rey G., “ Foundations for a Theory of Contextors”, *Computer-Aided Design of User Interface (CADUI02)*, 2002.

Coyle L., Ye J., Loureiro E., Knox S., Dobson S., Nixon P., “ A proposed approach to evaluate the accuracy of tag-based location systems”, *Ubicomp 2007 Workshops Proceedings*, Innsbruck Austria, p. 292 - 296, September, 2007.

Dobson S., Coyle L., Nixon P., “ Hybridising events and knowledge as a basis for building autonomic systems”, *IEEE TCAAS Letters*, 2007. To appear.

Dobson S., Ye J., “ Using fibrations for situation identification”, *in* T. Strang, V. Cahill, A. Quigley (eds), *Pervasive 2006 workshop proceedings*, Springer Verlag, p. 645-651, 2006.

Ganter B., Wille R., Applied Lattice Theory: Formal Concept Analysis, Technical report, Technical University of Dresden, Germany, 1997.

Garg V. K., Mittal N., Sen A., “ Applications of Lattice Theory to Distributed Computing”, *ACM SIGACT News*, September, 2003.

Gu T., Wang X. H., Pung H. K., Zhang D. Q., " An Ontology-based Context Model in Intelligent Environments", *Proceedings of CNDS 2004*, p. 270-275, January, 2004.

Heckerman D., A tutorial on learning with bayesian networks, Technical Report n° MSR-TR-95-06, Microsoft Research, Redmond, Washington, June, 1996.

Henricksen K., Indulska J., " Modelling and Using Imperfect Context Information", *PERCOM'04 Workshops Proceedings*, IEEE Computer Society, p. 33 - 37, 2004.

Logan B., Healey J., Philipose M., Tapia E. M., Intille S. S., " A Long-Term Evaluation of Sensing Modalities for Activity Recognition", *Proceedings of Ubicomp 2007*, Springer, Innsbruck, Austria, p. 483-500, September, 2007.

Loke S. W., " Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective", *Knowledge Engineering Review*, vol. 19, n° 3, p. 213-233, 2004.

Murphy K., " A Brief Introduction to Graphical Models and Bayesian Networks", , http://www.cs.ubc.ca/ murphyk/Bayes/bayes.html, 1998.

Ranganathan A., Al-Muhtadi J., Campbell R. H., " Reasoning about Uncertain Contexts in Pervasive Computing Environments", *IEEE Pervasive Computing*, vol. 03, n° 2, p. 62-70, 2004.

Saha D., Mukherjee A., " Pervasive Computing: A Paradigm for the 21st Century", *Computer*, vol. 36, n° 3, p. 25-31, 2003.

Schilit B., Adams N., Want R., " Context-Aware Computing Applications", *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

Siewiorek D., Smailagic A., Furukawa J., Krause A., Moraveji N., Reiger K., Shaffer J., Wong F. L., " SenSay: A Context-Aware Mobile Phone", *Proceedings of ISWC'03*, IEEE Computer Society, Washington, DC, USA, p. 248, 2003.

Tarafdar A., Gar V., " Software Fault Tolerance of Concurrent Programs Using Controlled Re-execution", *Proceedings of the 13th Symposium on Distributed Computing*, Bratislava, Slovak Republic, p. 210 -224, September, 1999.

Thomson G., Terzis S., Nixon P., " Situation Determination with Reusable Situation Specifications", *Proceedings of PERCOM'06 Workshops*, Los Alamitos, CA, USA, p. 620-623, 2006.

Weisenberg N., Gartmann R., Voisard A., " An Ontology-Based Approach to Personalized Situation-Aware Mobile Service Supply", *Geoinformatica*, vol. 10, n° 1, p. 55-90, 2006.

Woods W. A., " Taxonomic lattice structures for situation recognition", *Proceedings of the 1978 workshop on Theoretical issues in natural language processing*, Association for Computational Linguistics, Morristown, NJ, USA, p. 33-41, 1978.

Wren C. R., Ivanov Y. A., Kaur I., Leigh D., Westhues J., " SocialMotion: Measuring the Hidden Social Life of a Building", *in* J. Hightower, B. Schiele, T. Strang (eds), *Location- and Context-Awareness*, vol. 4718 of *LNCS*, Springer, p. 103-120, 2007.

Yau S. S., Huang D., Gong H., Yao Y., " Support for situation awareness in trustworthy ubiquitous computing application software", *Software: Practice and Experience*, vol. 36, n° 9, p. 893-921, 2006.

Ye J., Coyle L., Dobson S., Nixon P., " A Unified Semantics Space Model", *in* J. Hightower, B. Schiele, T. Strang (eds), *Location- and Context-Awareness*, vol. 4718 of *LNCS*, Springer, p. 103-120, 2007.