Using Situation Lattices to Model and Reason about Context*

Juan Ye, Lorcan Coyle, Simon Dobson, and Paddy Nixon

System Research Group, School of Computer Science and Informatics,
UCD, Dublin, Ireland
juan.ye@ucd.ie

Abstract. Much recent research has focused on using situations rather than individual pieces of context as a means to trigger adaptive system behaviour. While current research on situations emphasises their representation and composition, they do not provide an approach on how to organise and identify their occurrences efficiently. This paper describes how lattice theory can be utilised to organise situations, which reflects the internal structure of situations such as generalisation and dependence. We claim that situation lattices will prove beneficial in identifying situations, and maintaining the consistency and integrity of situations. They will also help in resolving the uncertainty issues inherent in context and situations by working with Bayesian Networks.

1 Introduction

Context-aware computing systems provide adaptive services or behaviours according to different contexts. Context can be sensed from physical sensors, profiled by users, or derived from application or meta-information existing in systems. This context, acquired without any further interpretation, is called *low-level* context. 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 [11].

It is difficult to build behaviours that adapt directly to low-level context. It is more attractive that a context-aware system is aware of *situations*, which are external semantic interpretations of context [4]. Compared to context, situations are meaningful, relatively stable, and certain. 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. A meeting detection application (such as when Sensay [12] attempts to detect whether a meeting is taking place or not) should not be overly concerned with

* 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".

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; whether the ambient noise levels are high? When a new type of context is introduced that can influence the situation (e.g., a meeting is scheduled in the calendar), then the situation specification is modified. However, its associated actions (e.g., change the mode of the attendances' mobile phones) will not be affected.

Situations, as an integral unit of semantics, are considered crucial in determining a system's actions. It is beneficial to define system behaviours only on situations, and make any context or contextual change transparent. Therefore, a promising context-aware computing system tends to be situation-aware.

As the study of situations has become popular, a huge number of situations are produced in an *ad hoc* way (for example, those outlined in Section 2). In order to benefit 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 *generalisation* relation: for example, a meeting situation is considered more general than a conference 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, i.e., there is a *temporal order* between the situations.

Changes in situations may cause the system to adaptively change 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 situation specifications, and study the dynamic evolution of situations. These challenges are also proposed as future work by Loke [8].

This paper does not aim to provide a novel representation for situations: we use the typical representation – logical predicates. We focus on how to study the characteristics of situations by applying lattice theory. Situation lattices will be used to analyse the relations between situations. They will help to maintain the consistency and integrity when defining situations. This can avoid checking and modification of situation specifications when the errors are detected at runtime. 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 design of situation lattices, and provides analysis of their characteristics. Section 4 discusses two approaches for dealing with uncertainties in situation identification. Finally, Section 5 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 raw context. Early attempts such as Gu *et al.*'s ontology-based model [13] used first-order logical predicates to define situations. These attempts simply composed context and situation with logical operators.

Yau *et al.* analysed the semantics of situations and gave them formal representations [16]. 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.

Costa *et al.* [3] studied the classification of situations in terms of their composition. A situation can be an *intrinsic context situation* – that is immediately derived from a single piece of context; it can be a *relational context situation* – that is used to associate multiple pieces of context in a certain relation; it can be a *formal relation situation* – that is defined by applying formal relations between two pieces of context directly, such as greater than, subset of, and distance; or it can be *combined situation* – that is made up from situations.

Loke [8] proposed a novel way of representing situations by decoupling the inference procedures of reasoning about context and situations from the acquisition procedure of sensor readings from context-aware systems. They apply a logic programming approach to characterising situations, which helps the system designer in naturally individuating and identifying situations for an application. It also provides a high level of programming and reasoning situation for the developers.

Thomson *et al.* provided a reusable library of situation specifications that helps to automatically determine situations [14]. They expressed 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.

3 Situation Lattice

This section examines the application of lattice theory [1] to the organisation of situations so as to study the characteristics of situations, such as generalisation, and dependence.

3.1 Construction of Situation Lattices

Situation lattices are inspired by Woods' use of lattice theory to recognise situations in linguistics [15]. A complete lattice is a partially ordered set where each subset of elements have the least upper bound and the greatest lower bound. The lattice theory is useful studying the structures with partial order. This paper will apply this formal theory to study situations.

Definition 1. *A **situation lattice**, L, is defined as $L = (S, \leq)$, where S is a set of situations and the partial order $\leq$ is a generalisation relation between situations. Each situation is associated with a logical description $l(t_1, \ldots, t_m)$, where t_i is a context predicate, or a basic or composite situation. $s_i \leq s_j$, $s_i, s_j \in S$ defines s_j to be a more general situation than s_i, iff any logical description being satisfied by s_j will be also satisfied by s_i. If two situations have no generalisation relation between them, then they are called **disjoint** situations.*

In S, a unique top situation $s_\top$ is a universally true situation. $s_\top$ is the most general situation so that $\forall s_i \in S$, $s_i \leq s_\top$. Dually, a unique bottom situation $s_\bot$ is a universally false situation. $s_\bot$ is the most specific situation so that $s_\bot \leq s_i$. Each situation in S is a basic or composite situation (except $s_\top$ and $s_\bot$). All basic situations are immediately under the top situation $s_\top$, which are derived from pieces of context through a general mapping function $f(c^*) = s$. The function takes a single piece c or a composition $c^* = c_1 \times c_2 \times \ldots \times c_n$ of context to derive a basic situation. A composite situation is made up of basic or other composite situations and sits under the basic situations in S.

Given $s_i, s_j \in S$, a situation s_j is more general than s_i iff the logical description l_j that is satisfied by s_j will also be satisfied by s_i. That is, $s_i \leq s_j$ implies that a logical description l_i subsumes l_j, labelled as $l_j \sqsubseteq l_i$. The relations $\leq$ and $\sqsubseteq$ have the same meaning w.r.t. the *partial order*. Thus, l_i can be rewritten by substituting l_j for the corresponding part in l_i: $l_i = l_j \wedge l_i^* \wedge l_i'$, where l_i^* is a logical description particular to l_i. l_i' is the residual part of the logical description, for besides s_j there may be other immediately more general situations above s_i.

To simplify the logical descriptions of situations, the specific situations automatically inherit the logical descriptions from its immediately general situations. This approach is used to build and maintain a situation lattice, whose formal proposition is expressed as follows.

Proposition 1. *At the appropriate level of generality of L, a situation s is specified with a logical description l^* that is only particular to itself. Its complete logical description can be obtained from $l = (\bigwedge_{i=1}^{m} l_i) \wedge l^*$, where l_i is the complete logical description of s_i that is the immediately more general situation than s:*

$s \preceq s_i$. In turn, $s_n \leq \ldots \leq s_1$ holds if all of the complete logical descriptions satisfy the following condition: $l_1 \sqsubseteq \ldots \sqsubseteq l_n$.

Figure 1 shows an example of a situation lattice for the meeting scenario discussed in Section 1. The basic situation $s_{highNoise}$ is identified by evaluating the noise degree sensed from the noise sensor and its logical description is `noise(conference_room, greaterThan, 3)`. The context from the positioning sensor is evaluated to identify the basic situation s_{np2}: the number of current people present is over two. The situation s_{mCal} will be true if there is a meeting scheduled for now in a sensed calendar. The situations s_{projOn} and s_{speOn} will be true if the projector and the speaker are turned on respectively.

A composite situation $s_{meeting}$ is used to evaluate whether a meeting is going on, whose logical description is expressed as $l_{mCal} \wedge l_{np2}$. The meeting situation $s_{meeting}$ has two more specific situations: a group meeting s_{gm} and a conference meeting s_{cm}. Each of these situations has inherited the logical description from $s_{meeting}$ and extended it with their particular descriptions. The logical description of a group meeting situation $l_{gm} = l_{meeting} \wedge l_{highNoise} \wedge l_{gp2}$, must satisfy that from another two situations: the noise level is above the third level, and there are at least two group members. The logical description of a conference meeting $l_{cm} = l_{meeting} \wedge l_{projOn} \wedge l_{speOn} \wedge l_{np10}$, must satisfy that from another three situations: the projector and the speaker are in use, and there are more than ten people present. Particularly, the situations s_{gp2} and s_{np10} are more specific situations relative to s_{np2}. s_{gp2} inherits the number requirement on involved people and extends them to the group identities of the people. s_{np10} simply constrains the number requirement on people present – at least ten people. In this lattice, s_1 is the unique top situation, and s_0 is the unique bottom situation.

For any two situations in a situation lattice, the join situation is the most specific situation among their more general situations, whose logical description should contain the common part of their logical descriptions. The meet situation is the most general situations among their more specific situations, whose logical description should contain the conjunction of their logical descriptions.

In whole, the essential characteristic of this situation lattice is the ability to represent situations of various degrees of generality. It explicitly represents the inheritance relationships between corresponding constituents of those situations.

3.2 Analysis of Situation Lattices

Exploring Dependence Relationships Between Situations A dependence relation between situations is discussed in most context modeling research (such as the research of Gu *et al.* [13] and Henricksen *et al.* [7]. We use situation lattices to capture this relationship.

The situation lattice is regarded as a specialisation structure with respect to the generalisation if it is observed downwards from the top down. A situation $s \in S$ is more *general* relative to all its sub-situations. It also can be considered as a dependence structure if it is observed upwards from the bottom. A specific

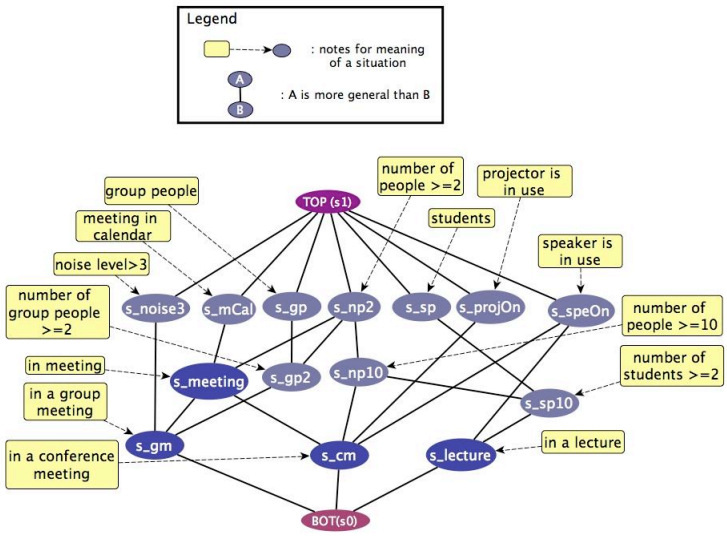

Fig. 1. Meeting situations in a situation lattice

situation can be decomposed into a few of more general situations. Its satisfiability *depends* on the evaluation of the satisfiablity of all its immediately more general situations.

In Figure 1, the satisfiability of a situation $s_{meeting}$ depends on that of its component situations: s_{mCal} and s_{np2}. The satisfiability of s_{mCal} and s_{gp2} depends on that of s_1. The top situation stores all the proper states of a system, and it holds if a system is running properly. Conversely, the bottom situation s_0 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, in a given place at a given time, s_0 holds if two situations s_{gm} and s_{cm} are identified by the system.

Maintaining the Consistency and Integrity of Situations Context-aware computing systems typically involve a large quantity of context, based on which a huge 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. For instance in Figure 1, it is not possible for a logical description of a group meeting situation s_{gm} to conflict with that of a general meeting situation $s_{meeting}$. Integrity means that logical descriptions should not be satisfied by any two disjoint situations. For example, the logical description that is satisfied by a group meeting s_{gm} in a room should not be satisfied by a conference meeting s_{cm}, or a lecture situation $s_{lecture}$ in that room at the same time.

Once the errors of inconsistency and non-integrity are detected at runtime, the system designers will be forced to rewrite situation specifications. This repetitive checking and modification takes a lot of time, therefore, it would be advantageous if these problems could be spotted and avoided when defining 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. According to Proposition 1, a new situation s is specified in a logical description: $l = \bigwedge_{i=1}^{m} l_i \wedge l^*$, where l_i is the complete logical description of one of its immediately more general situation 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. s is considered as an acceptable situation if its logical expression is different with that of its siblings: $l \neq l_j$.

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. To identify a meeting situation, a system will collect all the perceptible context, for example, noise level and the number of people in this room. If the context satisfies the conditions 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 logical description with the given context.

While in many real applications, where there are many possible situations, it is not always practical to locate a situation beforehand. In this case, **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, situations can be shared which avoids 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. When a set of most specific situations $\{s_i, \ldots, s_k\}$ are located, the target situation is the join of all these situations.

4 Situation Lattices and Uncertainties

When dealing with real-world context data, there is no guarantee that situations will be identified with complete certainty. The uncertainty of context is subject to sensor failure, noise, delays, disconnected sensor network, infrequent update in response to changes [7]. Context is considered uncertain, if it is

- *incomplete*, when some information is unknown or missing. There may not be enough evidence to determine a the correct situation;
- *imprecise*, when the resolution of the context cannot satisfy the requirement of applications;
- *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, especially when dealing with real-world context.

4.1 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 extended to more specific situations by adding requirements in its logical description (e.g., from s_{np2} to s_{gp2}), tightening the constraints (e.g., from s_{np2} to s_{np10}), or uniting with other situations (e.g., from s_{sp} and s_{np10} to s_{sp10}). If some context is too incomplete to support a given situation then this situation cannot be identified. However, its general situations will be checked until the context is satisfied by a situation. Therefore, when a system fails recognising a specific situation, it can loosen the requirement to locate a more general one. If the context is conflicting to each other, it generates some disjoint situations. These situations satisfy different conditions that are difficult to determine which is proper, while the conditions satisfied by all of them are considered correct.

As a result, the join of these disjoint 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.

4.2 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 mechanism. 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 [6].

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 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 (CPT) given all possible combination 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 the 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 [6] 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* [9] is applied for the conditional probability distribution. For each non-root node s, one of its discrete state is written as ω, its parent nodes are $s_1, \ldots, s_n$, and one of its conditional probability is calculated as follows:

$$p(s = \omega \mid s_1 = \omega_1, \ldots, s_n = \omega_n) = \frac{N(s = \omega, s_1 = \omega_1, \ldots, s_n = \omega_n)}{N(s_1 = \omega_1, \ldots, s_n = \omega_n)}, \quad (1)$$

where $N(s = \omega, s_1 = \omega_1, \ldots, s_n = \omega_n)$ is the number 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. In a meeting scenario example in Figure 1, the prior probability of a root situation s_{np2} is 0.74. For the group meeting s_{gm}, one of its conditional probabilities $p(s_{gm} = true \mid s_{highNoise} = low, s_{meeting} = true, s_{gp2} = true)$ is 0.87. When the noise level is sensed lower than the third level, the meeting situation takes place, and more than two group members are located, the probability of a group meeting is 0.87.

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: $p(s_i = \omega) = \prod_{k=1}^{n} p(s_k \mid parent(s_k))$. For example, if the situations s_{np2} is recognised *true* and s_{mCal} is recognised *true*, and other situations are uncertain, then the probability of a meeting situation $s_{meeting}$ is 0.98, and that of a group meeting situation s_{gm} is 0.64.

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.

Up to now, we assume that the Bayesian network is operating on a closed world, with a structure that is known *a priori*. However, in a real-world context-aware system this structure may not be known or will change over time as sources of context are added and removed by the environment. The assumption that inputs are certain is also unrealistic given the inherent uncertainty of context data [7]. 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.

5 Conclusion and Future Work

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

The situation lattice reflects the generalisation relation of situations and captures the dependence between situations. We believe it will be helpful when maintaining the consistency and integrity of situations, however, the involved computation may be huge when faced with lots of situations. This paper only presents a simple situation lattice with a limited number of situations, while we will attempt to design an algorithm to make the checking procedure scalable and efficient. The situation lattice will also be beneficial when identifying the situations using backward and forward chaining approaches. However, the situation lattice only reflects the static structures of situations. We have discussed the dynamic evolution of situations with a fibration theory in earlier work [4]. In the future, we will investigate how situation lattices and fibrations can be made to work together. *Chu Space* [17] is anther interesting worth studying, which may be useful for exploring the temporal order between situations, and for preserving situations' structures during the dynamic evolution.

In dealing with the issue of uncertainty, the situation lattice supports a coarse-grained approach and a fine-grained approach by working with Bayesian networks. Bayesian networks work well if situations are limited to a small number, and if the context sources are relatively fixed. (Gu *et al.* [5] and Ranganathan *et al.* [10] have successfully applied Bayesian networks to deal with uncertainties in contextual reasoning.) However, this assumption is contradictory to the nature of context-aware computing systems. A system may contain thousands of situations so as to satisfy various kinds of customised applications. If a non-root situation node has m parent situations, then the size of its conditional probability table is $2^m \times (m+1)$ (if each variable has only two discrete states). Considering the complexity of situations, the computation of conditional probability tables will be large.

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. What's more, if learning the structure of nodes is required, the NP-hard problem underlying the Bayesian network will become an obstacle [2].

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.

References

1. G. Birkhoff. *Lattice Theory*. Providence, R.I. : American Mathematical Society, 3rd ed edition, 1967.
2. E. Charniak. Bayesian networks without tears: making bayesian networks more accessible to the probabilistically unsophisticated. *AI Mag.*, 12(4):50–63, 1991.
3. P. D. Costa, G. Guizzardi, J. P. A. Almeida, L. F. Pires, and M. J. van Sinderen. Situations in conceptual modeling of context. In *EDOC 2006 workshop proceedings*, pages 6–16, October 2006.
4. S. Dobson and J. Ye. Using fibrations for situation identification. In T. Strang, V. Cahill, and A. Quigley, editors, *Pervasive 2006 workshop proceedings*, pages 645–651. Springer Verlag, 2006.
5. T. Gu, H. K. Pung, and D. Q. Zhang. A Bayesian approach for dealing with uncertain contexts. In *Procs of Pervasive 2004*, 2004.
6. D. Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, Washington, June 1996.
7. K. Henricksen and J. Indulska. Modelling and using imperfect context information. In *Procs of PERCOM'04*, page 33, 2004.
8. S. W. Loke. Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective. *Knowl. Eng. Rev.*, 19(3):213–233, 2004.
9. K. Murphy. A brief introduction to graphical models and bayesian networks. http://www.cs.ubc.ca/ murphyk/Bayes/bayes.html, 1998.
10. A. Ranganathan, J. Al-Muhtadi, and R. H. Campbell. Reasoning about uncertain contexts in pervasive computing environments. *IEEE Pervasive Computing*, 03(2):62–70, 2004.
11. B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.
12. D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, and F. L. Wong. Sensay: A context-aware mobile phone. In *Procs of the 7th IEEE ISWC*, page 248, 2003.
13. Tao Gu and Xiao Hang Wang and Hung Keng Pung and Da Qing Zhang. An Ontology-based Context Model in Intelligent Environments. In *Procs of CNDS 2004*, pages 270–275, January 2004.
14. G. Thomson, S. Terzis, and P. Nixon. Situation determination with reusable situation specifications. In *PerCom 2006 Workshop Procs*, pages 620–623, 2006.
15. W. A. Woods. Taxonomic lattice structures for situation recognition. In *Proceedings of the 1978 workshop on Theoretical issues in natural language processing*, pages 33–41, 1978.
16. S. S. Yau, D. Huang, H. Gong, and Y. Yao. Support for situation awareness in trustworthy ubiquitous computing application software: Papers from compsac 2004. *Softw. Pract. Exper.*, 36(9):893–921, 2006.
17. G.-Q. Zhang. Chu spaces, concept lattices, and domains. *Electronic Notes in Theoretical Computer Science*, 83, 2004.