Towards A General Approach for Reasoning about Context, Situations and Uncertainty in Ubiquitous Sensing: Putting Geometrical Intuitions to Work [1]

Amir Padovitz[2], Seng Wai Loke[2], Arkady Zaslavsky[2] and Bernard Burg[3]
[2]*School of Computer Science and Software Engineering, Monash University, Australia*
{amirp,swloke,arkady.zaslavsky@csse.monash.edu.au},
[3]*HP Labs, Palo-Alto, bernard.burg@hp.com*

Abstract

Context uncertainty and the incurred complexity of reasoning about context necessitate investigation of context awareness that would provide feasible solutions by means of models, architectures and algorithms.
In this paper we discuss our approach for reasoning about context under uncertain conditions in pervasive environments and explore the theory, design and implementation of a reasoning engine, which makes use of a novel modeling approach for describing and reasoning about context. We discuss both a theory, which is the basis for the reasoning processes as well as analyze the functional activity of the reasoning engine during experimentation.

1. Introduction

The advent of the pervasive computing paradigm has created an emerging necessity for adaptability and automation of systems [12, 4]. Coupled with a desire to enrich user experiences, an investigation of the context in which systems operate and need to adapt to, and the awareness of such systems to changes in this context has emerged.

The complexity associated with pervasive environments and the diversity of possible situations that may affect context-aware applications' behavior and their incurred reasoning complexity, necessitate extensive investigation and research in the field of context-awareness. Such research would need to provide feasible solutions by means of models, architectures and algorithms.

Consequently, emerging research has begun to look at context-aware systems more generally, independently of specific applications, and in recent years, has focused on various aspects concerning context, including context middleware and toolkits [2, 5, 1] that abstract concepts and assist in gathering information for reasoning about context, and ontologies that provide vocabularies to describe context [2, 3]. While middleware and toolkits assist in the acquisition of contextual data (e.g. [14]) and its dissemination across the context-aware platform, they do not handle the complexities associated with the inconsistent and uncertain nature of context. By obtaining and possibly fusing together data, either physical or virtual, systems are mostly capable of dealing with unambiguous and well known situations but face difficulties to adapt to or resolve more intricate or uncertain situations.

Software engineering of this breed of systems is needed in order to facilitate design and development of more adaptable, reasoning-capable and powerful context-aware systems. Engineering such systems is not only a question of toolkits and ontologies but also of approach and methodology, as well as abstractions and concepts [7, 6, 13]. In this paper we discuss our approach for reasoning about context under uncertain conditions in pervasive environments and explore the theory, design and implementation of a reasoning engine, which makes use of a novel modeling approach for describing and reasoning about context. In Section 2 we start by providing a general overview of the functional stages used during the reasoning process. We follow in Section 3 with a discussion of the theory behind the general functional stages, and provide selected parts of our context modeling approach, concerning fundamental modeling concepts and algebraic operations. We refer to this model as the Context Spaces model. The concepts of our modeling approach provide the foundations of the engine's reasoning processes and are used in Section 4 to explain actual functional activities taking place during the experimentation phase. In this section we discuss implementation details and provide results and analysis of selected experimental runs that exhibit the uncertain nature of reasoning about context. We conclude in Section 5.

2. General Overview of the Reasoning Engine

We explore a single component within an overall architecture for context-aware computing system, namely, a reasoning engine, termed ReaGine, its design principles and theoretical foundations, which are based on a novel context modeling approach. We start by providing a general overview of the functional stages that are used during the engine's reasoning processes.

Figure 1 illustrates five basic functional steps taken by ReaGine during the reasoning process. Reasoning starts when information arrives to the reasoning engine either as raw data or as basic reasoned context, often as a result of data fusion, preformed elsewhere. It is then checked for low-level discrepancies (e.g. contradicting evidence

[1] We thank HP Labs for collaboration and financial sponsorship of this work

between sensors). During this stage more elaborate verification techniques are also employed, resulting in recommended alternate values for suspicious sensor readings. At the next stage, the data is synthesized according to concepts drawn from the Context Spaces model and inferred as a collection of what we call *basic situations*. These inferred situations are uncertain, however, as we deal with only a partial view of the model due to the degree of uncertainty associated with context. The functional stage of merging contextual information does not consider the uncertain nature of context, manifested through context inconsistencies between seemingly parallel situations. The reasoning process considers factors related to context uncertainty, such as (1) insufficient data to infer context, which may be the result of cost efficiency considerations, (2) inherent inaccuracy of sensors, (3) the use of not up-to-date information (also affected by the system design and push-pull sensor technology), (4) context ambiguity, which is often the result of limitations in the deployment and types of sensors, (5) unknown contextual situations, which are true situations (possibly relevant to the system) that the system is unable to infer, and (6) conflicting or contradicting low-level data information.

The engine then attempts to discover conflicts (e.g. ambiguous situations or situations that cannot co-exist) at the conceptual level, between these basic situations by performing analytical procedures, using operations that are defined in Context Spaces algebra, and implemented as a set of operations in a separate library. Conflicting situations are dealt with in an additional verification phase, which attempts to resolve or verify the true situation's nature. Finally, situations that were not disqualified during the previous stages are composed into more complex situations and compared with policies predefined by the system designers. These final complex situations need not be predefined but are the result of the five reasoning stages discussed above.

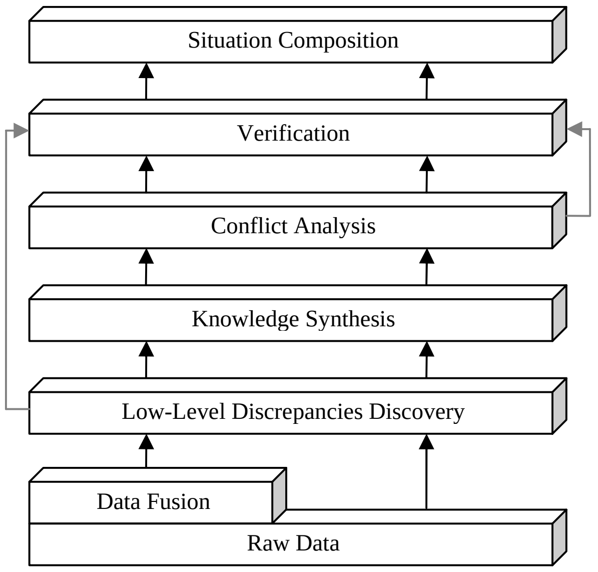


Fig. 1 – Reasoning engine functional stages

3. A Model of Context Spaces

Having broadly discussed the functional stages of the reasoning processes taken by ReaGine, we now explore the theoretical foundation for our reasoning approach. We will examine selected parts of the entire model, containing a sample of basic concepts and sub-concepts, which we will use in a later stage when we discuss implementation and analysis of the reasoning performance in experimental runs.

There are many dimensions of context. Context can be hierarchically structured [13] and/or categorized into different domains (e.g. social, location based, physiological, historical etc. [8, 12]) and can denote different levels of abstraction, depending on the desired situation to be modeled and the types of available information. Models that attempt to capture and express context details often investigate concepts and abstractions that are closely related to the application level. An example of common abstractions of this sort for entities in a pervasive system is those of persons, places and things [9]. These kinds of modeling are important in identifying key abstractions that are central in many applications but which are nevertheless still application-oriented. It has been suggested in [13] that many other information sources other than location and places (and in this manner other key abstractions) often play important roles in context-aware applications.

Our approach uses a more fundamental perspective over context than the above mentioned models, by characterizing context and its behavior from the context attributes level and proposing concepts to model and examine basic elements that make up context. This results in the ability to develop generic reasoning capabilities about context. We then examine concepts and practices that map these ideas into the real-world of application domains. Finally, we use a set of software engineering abstractions similar to other conceptual models, which we make use of according to specific application needs. We defer a short discussion on the latter to the implementation section.

3.1 Basic Terminology and concepts

A fundamental aspect in our modeling approach is the treatment of context as a system, where we distinguish between the current application state in regard to some context and the broader definition of that context and to which the current state belongs. We use the following definitions for the basic concepts in the model:

Context state

We term the application current state in relation to chosen context as *context state* and model it with a vector S_i as a collection of context attributes' values that are used to represent a specific state of the system at time *t*.

We assume a deliberate choice of context attributes by the context-aware system that enables the mapping of their values to a *context state* and therefore presuppose the ability to infer the current situation for any set of attribute values.

Let $S_i^t = (a_1^t, a_2^t, ..., a_N^t)$

Where S_i^t denotes a vector defined over a collection of *N* attribute-values, where each value a_i^t corresponds to an attribute a_i value at time *t*, representing the true context state *i* at time *t*.

A *context attribute* is defined as any data that is used to describe element/s that can be used in the process of inferring situations. A context attribute is often associated with sensors, either virtual or physical, having the value of the sensor readings denote the context attribute value.

Let a_i^t denote an attribute *i's* value at time *t*.

a_i is a specific attribute, which often denotes a specific sensor. When adding time subscript tagging we denote the value of the attribute at specific times.

Situation subspace

We conceive of a vector space $R_i = (a_1^R, a_2^R, ..., a_N^R)$ as a s*ituation subspace i*, consisting of *N* acceptable regions for these attributes. A *situation subspace* represents a static definition for regions of attribute values corresponding to some predefined situation.

An accepted region a_i^R is defined as a set of elements *V* that satisfies a predicate *P*, i.e. a_i^R = {*V* |*P*(*V*)}. For example, in numerical form the accepted region would often describe a domain of permitted real values for an attribute a_i.

While R_i defines permissible values for attributes, it does not automatically associates a *context state* with a specific *situation subspace*. Rather, it negates the possibility of being in other subspaces that do not contain the state. The reason for this is the possibility of having subspaces intersect, having the possibility of the state corresponding to more than one situation. We therefore say that a *context state* belongs to some situation iff some function associates the state to some situation defined by R_i. An example of such function would be a verification process, after which a *context state* is resolved to belong to a particular situation, or a conflict discovery process that approves the consistency of that *situation subspace*.

We can represent the relationships between *context state*s and *situation subspaces* graphically, where specific *context states* (either historical or current) can be visualized within (corresponding to the context description) or outside (contradicting the context description) the *situation subspaces*. A system's *context state* would most often be visualized as a multidimensional single point in space, contained within the tolerable region of acceptable values of some *situation subspace*. The notion of a context state being outside or within a situation subspace is illustrated in figure2.

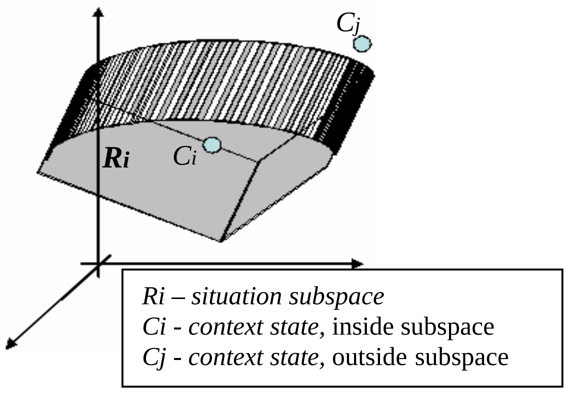

Cj
Ri
Ci
Ri – situation subspace
Ci - context state, inside subspace
Cj - context state, outside subspace

Fig. 2 - Visualization of *situation subspace* and *context state*

Situation types

We categorize situations into three types: (1) *basic situation*, which is the result of a simple reasoning process, by which a *context state* is proved to be contained in some *situation subspace*, (2) *refined situation*, which is the result of consistency checks and verification processes, after which a *context state* is associated with specific non-conflicting situations, and (3) *complex situation*, which is the result of merging together or intersecting existing (basic or refined) *situation subspaces*.

3.2 Algebraic Operations

Having defined basic concepts that capture multidimensional aspects of the contextual situations in terms of state and space, we now specify a set of operations that examine relationships between these concepts and can be used in the process of modeling and reasoning about context. In the context of this work we will only discuss two operations; for more detailed discussions on the algebra and Context Spaces model, the reader is referred to [10, 11].

Space Intersection

The intersection operator results in a new *situation subspace* that contains shared regions of values of the same attributes between two comparable *situation subspaces*.

In order to define the intersection of two *situation subspaces*, we first define the intersection of two accepted regions as follows.

Given accepted regions *A* = {*V* |*P*(V)} $\subseteq T$ and *B* = {*W* |*Q*(*W*)} $\subseteq T$, where *T* is some universal set of values (i.e., A and B are subsets of the same set, effectively of the same type), then the intersection of A and B, denoted by is defined as follows:

$A \cap B \; e \in (A \cap B)$ iff $P(e) \wedge Q(e)$.

Cj

Basically, the intersection of two accepted regions is the set of elements that satisfies predicates P and Q.

The intersection of two context spaces $R = (a_1^R, a_2^R, ..., a_N^R)$ and $S = (b_1^R, b_2^R, ..., b_M^R)$ is denoted by $R \cap S$ (where we rely on context to distinguish from intersection of accepted regions) and is defined as follows:

$$R \cap S = (c_1^R, c_2^R, ..., c_K^R, a_{K+1}^R, ..., a_N^R, b_{K+1}^R, ..., b_M^R)$$

where we assume that the first K (<= min (N,M)) attributes are the same (without loss of generality) though their accepted regions might not be, *i.e.* $c_1 = a_1 = b_1, ..., c_K = a_K = b_K$,

and $c_1^R = a_1^R \cap b_1^R, ..., c_K^R = a_K^R \cap b_K^R$ (the accepted regions for the common attributes are formed by intersecting the accepted regions of the respective attributes from the two context spaces).

We can use the intersection operator to produce new situations that are the product of two or more situations. The contribution of intersecting situations is the introduction of new situations that are unfamiliar to the context-aware system (e.g. *complex situation*s), which we regard as an initial step towards intelligent adaptability and discovery of unfamiliar, not pre-defined situations, by the system.

As an example, we can model *situation subspace*s for 'subject is running' and 'subject is walking' situations with the following attributes: Heart Rate, Respiratory Rate and Body Temperature. The accepted region of values of these attributes for the two situations and their visualization are given in figure 3. A new *situation subspace,* which is the result of the intersection of the basic situations, can be visualized as the shared area between the two spaces.

The intersection operator is useful for identifying ambiguous or conflicting situations, in which a system needs to determine the correct situation or situations, when some of these situations cannot co-exist. In the example above when a *context state* is situated within the intersection space of 'running' and 'walking' and assuming a subject cannot be both running and walking at the same time, a system needs to decide whether the subject is a very fit runner or actually not very fit (and is actually walking).

State-Space Difference

One of the operators that examine the degree of similarity between *context state* and *situation subspace* is the *State-Space Difference* operator, with which we measure similarities between *situation subspaces* and *context states.*

The following operator holds only for quantifiable attributes where such values exist. We make use of other operators, which are described in [11], which account for non-numerical attribute values.

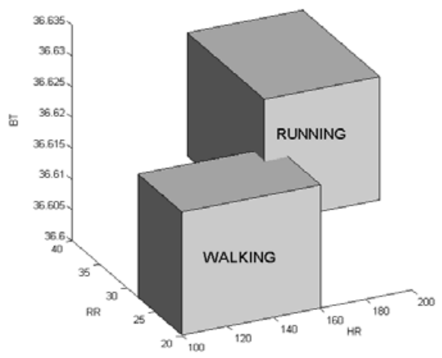
Situation subspace		**Heart rate (BpM)**	**Respiratory Rate (BrpM)**	**Body Temp. (Deg.Celsius)**
R_1	Walking	100 - 160	20 - 28	36.60–36.62
R_2	Running	150 - 200	26 - 40	36.61–36.63
$R_1 \cap R_2$		150 - 160	26 - 28	36.61 – 36.62
				

Fig. 3 - definition and visualization of 'Walking' and 'Running' *situation subspace*s

Let a_i^s and a_i^r denote values of corresponding attributes of the same type for the *context state* and *situation subspace*, respectively.

Let $\hat{a}_i^r \in a_i^r$ define an attribute value that is contained in the set a_i^r, which has the most similar value to a_i^s, where similarity between two identical attribute values is defined by the application designers and reflected with positive numerical values, such that the smaller the value the more similar are the two attribute values.

Let $\hat{a}_i^R$ reflect a numerical value, denoting the acceptable region's absolute size, such that $\hat{a}_i^R = | a_{i_{\min}}^R - a_{i_{\max}}^R |$,

where $a_{i_{\min}}^R$ and $a_{i_{\max}}^R$ denote minimum and maximum values of the accepted region a_i^R, and where for simplicity, we assume a continuous region of acceptable values

We then define the *State-Space Difference* operator as:

$$\textit{State-Space Difference} = \sum_{i=1}^{N} \frac{| a_i^s - \hat{a}_i^r |}{\hat{a}_i^R}$$

This operator is useful for assisting in the adaptation process for unknown situations, where the application can reveal the similarity of a current *context state* to predefined *situation subspaces.* The definition also accounts for normalization needs when comparing different types of attributes with inherently different sizes of regions.

As an example, consider the scenario of 'running' and 'walking' situations in the earlier subsection. Suppose the system encounters sensor readings corresponding to the following state: S_i = (210BpM, 30BrpM, 36.62dc). As this *state* does not belong to any of the known *situation subspaces*, the system tries to adapt and treat the unknown state by finding the most similar *context space.* The calculation yields a much lesser difference to the 'running' *situation subspace* (0.2) than to the 'walking' *situation subspace* (1.083). The system adapts by inferring the 'running' situation and performs appropriate actions, accordingly. The decision to perform this calculation, the allowed difference from which adaptation to a specific situation is performed and whether at all adaptation is sensible, is application specific.

3.3 The Semantic Layer – bridging the application domain gap

We have so far discussed basic concepts in the model and have illustrated the use of operations over these concepts. A problem that needs to be addressed though, is the applicability of the general model to different application oriented scenarios and environments. In other words, ideas, operations and techniques derived from the model are applicable as long as they make sense to the application at hand.

The result of a *Space Intersection* operation for example, may have different meanings and consequences for different applications that operate in different domains. It is therefore important to model the nature of possible interactions and applicability of various elements in the model so that we can translate the general theory into practical uses.

We conceive of a second, higher level of semantic concepts on top of the general model (as described up to this point), which we make use of to map the model into specific application domains. Within this layer we provide information regarding the nature of semantic entities in the model (e.g. situations), such as their compatibility (e.g. can they co-exist?) or provide information regarding mapping of attributes semantic values into numerical ones. While some attributes types can be expressed more naturally semantically, it is possible to represent them numerically as well. We have addressed this issue in [11] where we discussed a technique that enables us to quantitatively specify values and domain of values that correspond to a variety of attributes, including semantic ones. The ability to model context numerically enables us to apply useful techniques over context that require values quantification.

As an example, we will now describe a representation technique that can be used for reasoning and conflict discovery.

Situation Natural Flow

An important aspect we model in the semantic layer of the model is characteristics of *situation subspaces*. An example of such characteristic is *situation natural flows*. - by this, we refer to possible logical evolution of situations, their direction and reasonable time limits of transition between these situations.

The use of this semantic description is illustrated in figure 4a and 4b, where we describe possible situations inferred by the system according to two physiological context attributes. In this example, the current *context state* is situated in the intersection of the 'Running' and 'Sick' *situation subspaces*. We assume that a subject can be both sick and running, since these two situations are defined as compatible elsewhere. The reasoning problem that arises here is whether the subject is only running ($S_i^t \in$ 'Running')? Or only being sick ($S_i^t \in$'Sick')? Or is actually both running and being sick at the same time ($S_i^t \in$'Running' $\cap$ 'Sick')?

Figure 4b provides an illustration of possible conceptual natural flows between defined situations. We also assume that reasonable transition periods between these situations are defined. We can use such knowledge to assist in inferring the correct situation, as follows. A verification procedure searches the knowledge-base for previous inferred situations and discovers that at time $t-\lambda$ (say 10 seconds ago) the subject was in the 'Walking' situation ($S_i^{t-\lambda} \in$'Walking'). It then reveals that at time $t-\lambda$ the subject was not in the 'Sick' situation ($S_i^{t-\lambda} \notin$'Sick'). It then infers that it is more probable for the subject to be currently running (e.g. If $S_i^{t-\lambda} \in$'Walking' AND $S_i^{t-\lambda} \notin$'Sick' then infer the 'Running' situation).

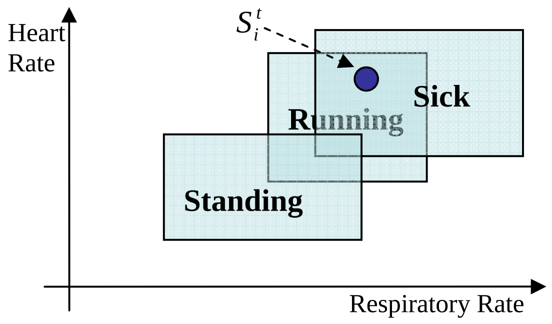


Fig. 4a – situation co-existence problem

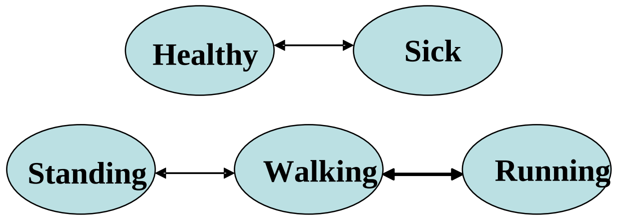


Fig. 4b – situations natural flow

4. Implementation and Experimental Evaluation

The reasoning engine implementation follows the steps illustrated in the functional analysis diagram at

section 2, and was developed to provide reasoning capabilities that partly overcome uncertainty problems associated with context, by leveraging on general techniques based on the Context-Spaces model. A separation of the application semantics from the reasoning procedures and techniques that are built on a generic model that represent context enables easy deployment and use of the engine in a variety of different scenarios.
To illustrate feasibility and advantages of our approach we demonstrate the engine's reasoning for a simulation of an office environment use case, and provide walk-through and analysis of its behavior.

In this set of experiments we deployed several *Monitor* services, physically distributed over the network. Each such *Monitor* service communicated with processes, simulating independent sensors, during which it either actively acquires raw data or passively waits for data to be push by the sensors, according to the sensor type. Each sensor is individually configured to follow specified pattern of behavior that is characterized by value and type of data it produces, how often new data is produced, inherent inaccuracy associated with the sensor (reflected by random errors values added to the basic value with specified variation over time) and type of access available by the sensor (push or pull).

Raw data collected by a *Monitor* service is pushed via distributed communication (and message queuing technology for scaling purposes) to a centralized knowledge-base, which is the source of information for the reasoning engine in this experiment. The reasoning engine itself can both be triggered by events originating from remote *Monitor* services as well as periodically performing reasoning actions. Figure 5 provides the deployment architecture used during the experimentation.

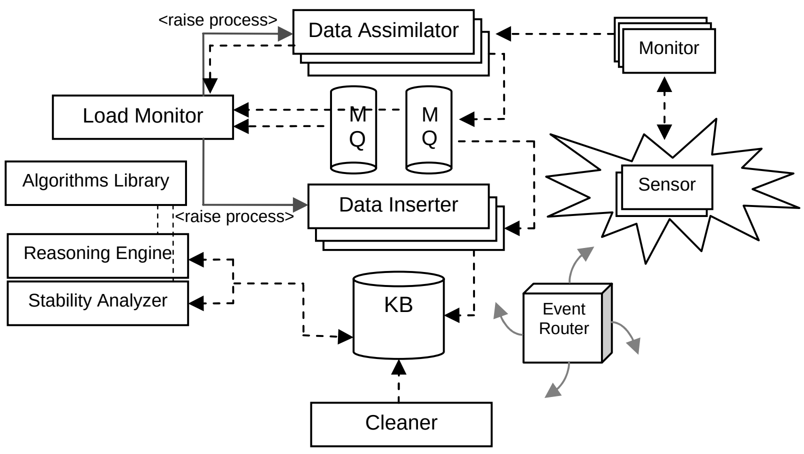
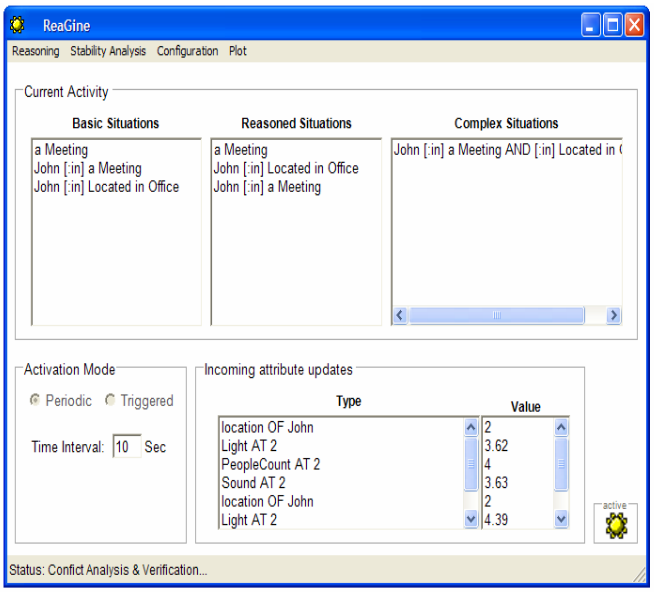


Fig. 5 – The general architecture in which ReaGine operates

In this set of experiments we were interested not only in performing basic reasoning but also in reasoning about situations that were not obvious to the system, either by not fully corresponding to existing definitions, exhibiting conflicting information or displaying ambiguities between possible contradicting situations. Consequently, we expected the system to adapt or verify the true nature of the situation through the use of the reasoning capabilities.

4.1 Sensor inaccuracies

In the first example, *situation subspaces* and semantic configurations were defined to capture context details corresponding to three types of situation abstractions, namely, (1) User related situations, which are situations specifically associated with a user and inferred for a user. (2) Location related situations, which denote situations associated with specific locations. And (3) General situations, which are situation that can take place in several places involving different users at the same time.
We configured the system to denote typical office situations such as meetings, presentations, users of the system and meaningful locations in the networked environment. We will examine simulated sensor readings, corresponding to the general situation of a meeting (that can take place in any employee's office as well as in the meeting room and presentation theatre) and a user 'John' who participates in this meeting at his office.
In this initial stage of the experimental run the reasoning engine successfully inferred the general meeting situation and associated the user with the meeting and with a meaningful location. Figure 6 presents the reasoning engine's results for this run.



Current Activity

Basic Situations	Reasoned Situations	Complex Situations
a Meeting	a Meeting	John [:in] a Meeting AND [:in] Located in (
John [:in] a Meeting	John [:in] Located in Office	
John [:in] Located in Office	John [:in] a Meeting	

Activation Mode

(● Periodic) ○ Triggered

Time Interval: 10 Sec

Incoming attribute updates

Type	Value
location OF John	2
Light AT 2	3.62
PeopleCount AT 2	4
Sound AT 2	3.63
location OF John	2
Light AT 2	4.39

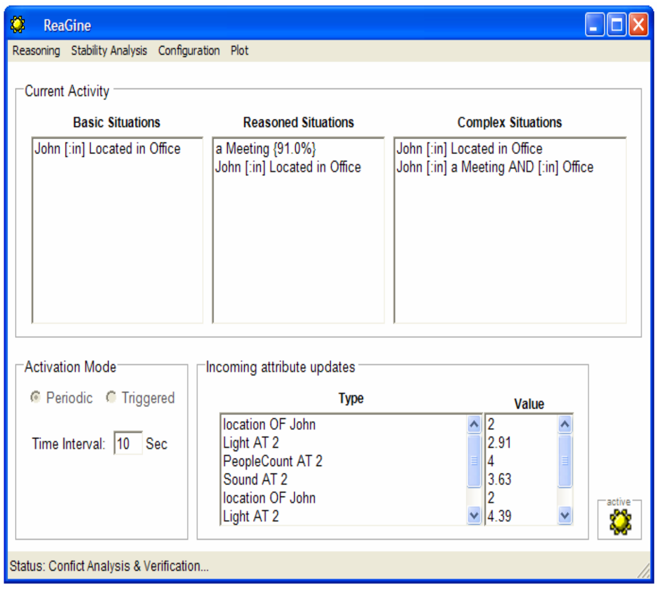
Status: Confict Analysis & Verification...

Fig. 6 – Basic experimental run

The form in figure 6 is designed to denote three levels of reasoning: (1) inferred Basic Situations that are the result of the knowledge synthesis phase. (2) Reasoned Situations, which are situations that were produced after more elaborate reasoning and/or verification processes (initiated when discrepancies or conflicts are revealed). (3) Complex Situations, which are the result of composing together compatible Reasoned Situations.

We then altered the experiment settings and changed some of the sensors typical behavior, such that while still having a meeting situation, sensors values are distorted to emulate high inherent inaccuracies and sporadically reflect data that does not correspond either to any known situation or the meeting situation. Examples of this are changes to the sensed noise level and light intensity in the meeting area. Figure 7 presents the reasoning engine's results for such experimental run.

Note that, with the emulated errors in sensor values, the system now fails to infer the meeting as a basic situation. However, the reasoning process identifies the meeting in its next functional stage when it performs adaptation to possible situations for the *context-state*, assuming possible inaccuracies in sensor readings. Finally, it composes together all reasoned situations, resulting in inference identical to that in the initial scenario. The system has therefore succeeded in handling inaccurate sensor readings and correctly inferring the associated situations (as though there were no errors in sensor readings).



Current Activity

Basic Situations	Reasoned Situations	Complex Situations
John [:in] Located in Office	a Meeting {91.0%}	John [:in] Located in Office
	John [:in] Located in Office	John [:in] a Meeting AND [:in] Office

Activation Mode

[✓ Periodic] Triggered

Time Interval: 10 Sec

Incoming attribute updates

Type	Value
location OF John	2
Light AT 2	2.91
PeopleCount AT 2	4
Sound AT 2	3.63
location OF John	2
Light AT 2	4.39

Status: Confict Analysis & Verification...

Fig. 7 – Uncertain information run

The actual adaptation-oriented reasoning process is the following. After basic synthesis, the engine splits the current *context-state* into sub-states that match *situation subspace*s definitions.

It then computes the *state-space difference* between the context sub-state and a selected *situation subspace*, using the algebra operations library as described in section 3.2. If the revealed difference between the *context state* and *situation subspace* is below a certain threshold specific for that *situation subspace* definition (configurable by the application designers) it deduces the existence of such a situation.

Figure 8a and 8b visually demonstrates the *state-space difference* operator activity illustrating the relationship between the *context state* and *situation subspace*s. In this example a 'meeting' *situation subspace* and 'user working in office' *situation subspace* are defined and inferred by three sensor types:

(1) Light intensity in location, (2) number of people in location and (3) noise level in location. We use the axes in the diagram to denote the dimensions (i.e. the context attributes) of the subspace, and fill the subspace's region of values area in blue. A separate red triangle connecting a specific single value from each context attribute denotes the *context state*. The identical *context state* does not fully match any of the situations but exhibits much greater similarity with the 'meeting' subspace, in which the light intensity readings only slightly deviate from the subspace definition. The *state-space difference* operation results in a very low difference between the state and the 'meeting' subspace, thereby inferring its existence.

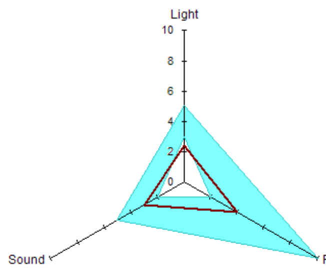


Fig. 8a – A meeting.

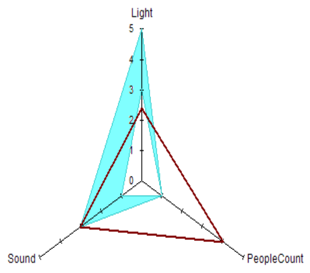


Fig. 8b – User in office.

Figure 9 provides the process's functional flow description.

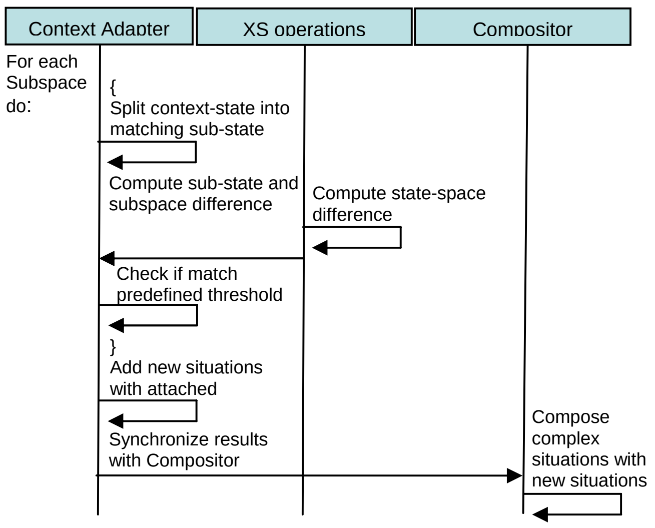

Context Adapter
XS operations
Compositor
For each Subspace do:
{
Split context-state into matching sub-state
Compute sub-state and subspace difference
Compute state-space difference
Check if match predefined threshold
}
Add new situations with attached
Synchronize results with Compositor
Compose complex situations with new situations

Fig.9 – Functional flow of adapting inaccurate readings

5. Conclusion

We have provided a partial view of the Context Spaces model and illustrated how various aspects of the model can be used for reasoning about context in different levels of uncertainty. An important feature of the model, but which is not the main focus of this work, is the dynamic aspects of context, which are naturally captured in the state-space representation approach on which the model is based. Once we have treated context in terms of state and space, other concepts concerning context, such as historical context trajectory, and system stability with regard to specific context as well as various techniques for context prediction and reasoning have been developed and included in the Context Spaces model. For detailed discussions on these features, the reader is referred to [10, 11].

The different features of the models are presented in the reasoning engine, which is built on a modular approach and which makes use of separate libraries that provide operations based on the Context Spaces algebra (as well as algorithms for context reasoning and prediction). The functional stages discussed in section 2 offer useful modularity and efficiency in the reasoning process, by which only uncertain or inconsistent situations are dealt with in advanced stages.

A fundamental design principle, which we apply to the various reasoning mechanisms and the overall system, is the decoupling of the application semantics from the implementation. We store the semantic part of the model, which in part is defined by the application designers, separately. By following this rule our goal is to achieve a model which is reusable and exportable through the semantics. The semantic repository describes elements of the system, policies based on context, elements and operations of the Context Spaces model, and a set of concepts that describe various behaviors and characteristics of situation subspaces (e.g. possible natural flows between situations, situation compatibility, etc.). The use of the Context Spaces model, which is a general model for capturing and reasoning about context, as the underlying model of the algorithms and the separation of the application semantics from the functional components, enables the development of reasoning techniques that are generic and can be used in a variety of settings and application types. Algorithms that make use of the context spaces model treat any kind of context uniformly and acquire the semantic information, specific to the application at hand, from a higher level, external to the algorithms. By following this approach we seek to create an infrastructure for context-aware computing at the reasoning level that is able to perform complex reasoning for different applications once specialized with application semantics.

In future work we intend to combine the centralized reasoning engine with distributed and mobile software components, creating hybrid architecture that work together in sensing the environment and reasoning about relevant context.

References

[1] Campbell R. H., Ranganathan A., A middleware for Context-Aware Agents in Ubiquitous Computing Environments, ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 2003.

[2] Chen H., Finin T., Anupam J., An Intelligent Broker for Context-Aware Systems, Ubicomp 2003, October 2003.

[3] Chen H., Finin T., Anupam J. An Ontology for Context-Aware Pervasive Computing Environments, Workshop on Ontologies and Distributed Systems, IJCAI-2003, Acapulco, Mexico, August 2003.

[4] Cheng S. W., Garlan D., Schmerl B., Sousa J. P., Spitznagel B., Steenkiste P., and Hu N., Software Architecture-based Adaptation for Pervasive Systems, , Conference on Architecture of Computing Systems (ARCS'02): Trends in Network and Pervasive Computing, April 8-11, 2002.

[5] Dey A. K., Salber D., Abowd G. D., A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human Computer Interaction, 16(2-4):97-166, 2001.

[6] Dey A. K., Abowd G. D., Towards a Better Understanding of Context and Context-Awareness, GVU Technical Report GIT-GVU-99-22, June 1999, ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf

[7] Huadong W., Mel S., Sevim A. Sensor Fusion for Context Understanding, IEEE Instrumentation and Measurement, Technology Conference, Anchorage USA, May 2002.

[8] Huadong W., Mel S., Sevim A. Sensor Fusion for Context Understanding, IEEE Instrumentation and Measurement, Technology Conference, Anchorage USA, May 2002.

[9] Kindberg, T., et al.: People, places, things: Web presence for the real world. Technical Report HPL-2000-16, Hewlett-Packard Labs (2000).

[10] Padovitz A., Zaslavsky A., Loke S. W., Burg B., Stability in Context-Aware Pervasive Systems: A State-Space Modelling Approach, Workshop on Ubiquitous Computing (IWUC'04), at ICEIS 2004, Porto, Portugal.

[11] Padovitz A., Loke S. W., Zaslavsky A., Towards a Theory of Context Spaces, Workshop on Context Modeling and Reasoning (CoMoRea), at PerCom 2004, Orlando, Florida, March 2004.

[12] Satyanarayanan M., Pervasive Computing: Vision and Challenges, IEEE PCM August 2001, pp. 10-17.

[13] Schmidt A., Beigl M., Gellersen H. W., There is More to Context than Location, Proceeding of the International Workshop on Interactive Applications of Mobile Computing, Rostock, Germany, November 1998.

[14] Schmidt A., Strohbach M., van Laerhoven K., Friday A., Gellersen H. W., Context Acquisition Based on Load Sensing, UbiComp 2002: Ubiquitous Computing, Gteborg, Sweden.