Chapter 3

FOUNDATIONS FOR A THEORY OF CONTEXTORS

Joëlle Coutaz, Gaëtan Rey
CLIPS-IMAG, BP 53, 38 041 Grenoble Cedex 9, France
E-mail:{Joelle.Coutaz, Gaetan.Rey}@imag.fr

Abstract This article proposes an operational definition of the notion of context for the design and development of context-sensitive systems. Our definition draws upon the distinction between the notion of an instant snapshot of observables (a situation) and the composition of these observables over time (a context). Observables and their relationships, which are elicited at the design stage of the development process, can be mapped, at the implementation phase, as colonies of contextors. A contextor is a software abstraction that models relationships between observables. Contextors share a common I/O structure including control channels and meta-data to ensure and express QoS (e.g., precision, stability), as well as common properties such as reflexivity and remanence. They can be combined as oriented graphs or encapsulated into higher computational units.

Keywords: Context aware system, ubiquitous computing.

1. INTRODUCTION

Since the mid-eighties, context has been considered as an important element in the design process of interactive systems. However, context was assimilated to the workplace and the workplace served as the foundation for contextual design [3]. With the advent of mobile devices and the availability of worldwide digital networks, the workplace is no longer limited to the confined space observed at design time. Tasks that have been modeled for a particular environment may now be accomplished in multiple settings using different interaction devices. From *context-confined*, interactive systems are progressively becoming *context-sensitive* and/or *context-aware*.

Early experiences show that context-sensitivity is a tricky problem. For example, Cheverst et al. report that using context to simplify users' tasks is sometimes perceived by users as system preemption [7]. If the research community aims at developing context-sensitive systems that are effective for humans, we need to devise an operational model for the notion of context that can be used and refined at every step of the development process. The framework developed by Thevenin et al. is an example of how such a context model can be used in the development process of plastic user interfaces [5, 6, 21].

Although many papers in the literature make reference to context, there is no clear widely agreed definition [24]. In the next section, we present a brief analysis of the state of the art in context-aware computing that elicits the key concepts related to context. From these concepts, we then elaborate our own definition in Section 3. This conceptual abstract definition is then refined in Section 4 from an external user-centered perspective into an internal computational model called a contextor.

2. ANALYSIS OF THE STATE OF THE ART

In 1994, Schilit and Theimer define context as the location and the identity of "the collection of nearby people and objects, as well as changes to those objects over time" [18]. According to Schilit et al., context analysis boils down to answering the Quintilian questions such as "Where are you, Who you are with, Which resources are nearby? When this happens? Etc." [1,19]. Environmental entities including the season of the year, time of the day, and room temperature, are part of the context [4]. From these early studies, we observe that *state* and *time* are two key underlying concepts for context awareness [22].

Later, Pascoe introduces the notion of *relevance*: context is a subset of physical and conceptual states that may be "relevant to a particular entity" [15]. According to Dey, an entity is "a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the applications themselves" [8, 9]. Surprisingly, it is only recently that relevancy is considered in relation to the user's task.

In summary, the state of the art related to context-aware computing brings in the following key concepts: *state*, *time*, *relevancy* to user's *task* and *entities* of which *location* and *physical environment* are first class components [10, 12, 22]. None of the definitions makes it explicit the composition of states over time and none of them explicitly considers the user as central.

We propose an operational definition for the notion of context that bundles the key concepts drawn from the literature, but where the *user* and the

task are first class entities, and where relevant states are composed over time. Instead of referring loosely to context (without specifying its purpose), we consider more appropriate to use the term *context of interaction*. Thus, our definition of context is with respect to interaction.

3. THE NOTION OF CONTEXT

We elaborate our own definition of context in five points:

- The definition per se in general terms,
- The refinement of the definition in terms of gross and net contexts, of user and system contexts, and of observed and observable contexts,
- The notion of peripheral state variables, one of the key items of a context,
- The composition function used to express the effect of past on context,
- A generalization of our definition of context in relation to ubiquitous computing, extending our user-centered perspective to an agent view whether agents are humans or artefacts.

3.1 Definition

Our definition of context draws upon the notion of situation. A situation at time t is a state vector, that is a set of observables at t. Context at t is a composition of multiple situations over a period of time. In addition, a situation as well as a context do not exist as autonomous things: they are always related to something. In the case of interest, they are related to users involved in a particular task. More precisely:

Given a set of users, U, a task, T, and two instants of observation, t_0 and t, where t_0 is the temporal reference for observations, the *Context* at t that relates to U for performing T, is the composition of the *Situations* observed between t_0 and t that relate to U for performing T.

context ^{U,T} (t) = COMPOSITION (situation ^{U,T} (t_0), … , situation ^{U,T} (t))

where :

situation ^{U,T} (t) is the Situation at t that relates to U for performing T.

The *Situation*, situation ^{U,T} (t), is the set of the values observed (or observable) at t of the peripheral state variables that relate to U for performing T, as well as their relations.

Peripheral state variables denote the entities that are *not central* to U at t for performing T, but that may have an impact on T, *now* (i.e., at t) and/or in the *future*(i.e., at t+dt).

The *impact of peripheral state variables* on a task may be one of the following:

- The task is aborted,

– The task is suspended, then resumed,
– The task is carried out but the nominal expected quality of the task is likely to be altered (for better or for worse). The quality of the task execution may be measured in terms of the quality of its product, the duration, the robustness (e.g., the number of user and/or system errors), etc.

The choice of U, the selection of the instants of observation t_0 and t, the granularity of the task and time, as well as the nature of the peripheral state variables are left opened: It is up to the designers to make the decision depending on the step in the design process as well as on the level of precision required. For example,

– U may denote a single user, a class of users, multiple classes of users, the whole humanity. The choice depends on the degree of precision that the designers are seeking for;
– t_0, which is the temporal reference for elaborating a context, may denote the starting time of the session. It may correspond to the beginning of the execution of the task, to a new day or year, or to any event relevant to the case at hand;
– The *granularity of a task* can be measured by its depth within the task model that the computer system is supposed to support. (A computer system is composed of multiple software components that interoperate. It ranges from closed applications as we now them today, up to world-wide computing as envisioned by Weiser [23]) In classic HCI, a task model is a hierarchical decomposition of tasks where a task is defined by the couple "goal, procedure". The goal describes the system state that the user desires, and the procedure corresponds to the plan(s) available to reach the goal. The designer may decide that only specific subtasks of the task tree should be concerned with context-sensitivity. Conversely, the root of the tree will be considered if, from the analysis of the users' needs, context-sensitivity applies to the whole system.
– The *granularity of time intervals* ($[t_0, t]$, dt) depends on the types of the state variables to be observed and/or the quality requirements defined early in the development process. For example, considering changes of light conditions, a time interval of 1 minute is reasonable for supporting the self-calibration of a computer vision algorithm, whereas 15 minutes is appropriate to automatically control the lights in the home.
– *Peripheral state variables* are discussed next.

3.2 Peripheral state variables

The *peripheral state variables* that relate to a user U for performing a task T are those variables that are not central but that may have an impact on the task, now or in the future. The central state variables that relate to U for

performing T denote the concepts that U necessarily manipulates (literally or mentally) for achieving T.

For example, consider the task of withdrawing money from a teller machine. The amount of money to be withdrawn and the state of the bank account are central to the task, whereas people waiting in line are peripheral. However, because the line is long, one may decide to skip the subtask that consists of checking the money left in the bank account. The peripheral state variable, length of the line, has an impact on the execution of the task. In our example, a subtask of T, checking the bank account, is not accomplished. The impact of the length of the line is the immediate abortion of T.

Note that state variables that are peripheral for T and U may be central for another task and/or for another set of users. Central and peripheral states variables are identified by the designers using sound design approaches such as Suchman's Situated Action approach [20], Activity Theory [2], Distributed Cognition [11], or Contextual Design [3, 13]). Central variables are subsequently conveyed in task models (e.g., GOMS, ADEPT, DIANE), as task parameters or objects. Unfortunately, peripheral state variables are not explicitly expressed in current task models. As a result, they are lost during the development process. In ARTstudio, a software tool that supports the development process of plastic user interfaces, we have extended ConcurTaskTree [14] by ordering state variables so that when screen estate is insufficient, secondary variables are not observable but browsable [5].

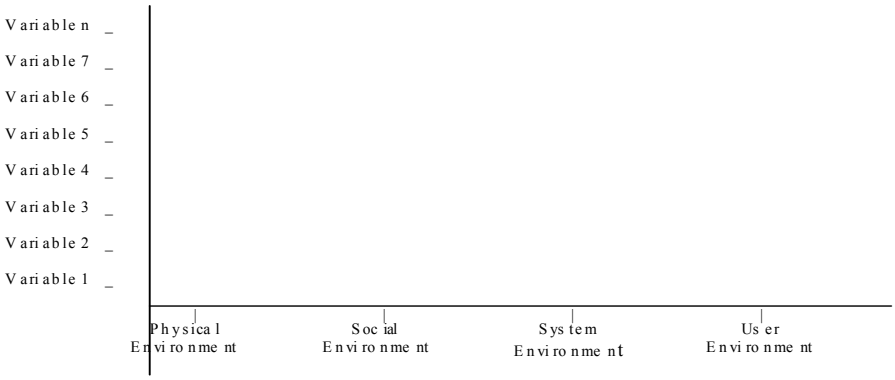

Figure 1. A simple classification space for eliciting and organizing peripheral state variables.

Because peripheral state variables potentially cover the entire universe, it is important that they be identified early in the design process. To assist the identification and classification process, we suggest to use the design space

shown in Figure 1. A peripheral state variable V_i may belong to one or several of the following classes:

- The *physical environment* covers the peripheral state variables that characterize the physical spaces and places [12] where task T is likely to be performed by users U. Such characteristics include light conditions, temperature, and noise level. They cover architectural characteristics such as connectedness, as well as social values (a house is distinct from "home"). They also include real world objects such as chairs, pencils, and toys.
- The *social environment* denotes the people that are in the physical environment in relation to T and U: people waiting in line, colleagues that may come in, friends that we are likely to meet at the airport, family members, a companion robot, etc.
- The *system environment* covers the computational, communicational and interactional resources available or potentially available to U for accomplishing T (e.g., a PDA versus a PC or a mobile phone, or any augmented object.
- The *user environment* corresponds to the characteristics of U with regard to the task T (e.g., expert in the domain covered by the task, special needs and preferences in relation to T, etc.).

3.3 Refinement: System and User Contexts, Gross and Net Contexts, Observed and Observable Contexts

In HCI, it is good practice to analyze a problem from at least two complementary perspectives: that of the user and that of the system. For example, one should check whether the dialogue controller of a running system is compliant to the task model defined during the design phase. For this reason, we need to consider the context and the situation both from the user and the system perspectives. As shown in Figure 2, we introduce the notions of *user context* and of *system context* and relate them to the *gross* and *net* contexts. The same lines of reasoning hold for situations.

The *gross context* at t that relates to U for performing T, $context^{U,T}_{G}(t)$, is the composition of all of the gross situations that have been observed between t_0 and t and that relate to U for performing T.

The *gross situation* at t that relates to U for performing T, $situation^{U,T}_{G}(t)$, includes all of the universal facts (variables and their relations) observed (or observable) at t that relate to U for performing T.

The *system context* at t that relates to U for performing T at t, $context^{U,T}_{S}(t)$, is the subset of the gross context, $context^{U,T}_{G}(t)$, that the system is concerned with at t.

Similarly, the *user context* at t that relates to U for performing T, *context* $^{U,T}{}_{U}$ *(t)*, is the subset of the gross context, *context* $^{U,T}{}_{G}$ *(t)*, that the user U is concerned with at t.

The *net context* at t that relates to U for performing T, *context* $^{U,T}{}_{N}$ *(t)*, is the subset of the gross context, *context* $^{U,T}{}_{G}$ *(t)*, that the user and the system have in common at t. Thus, by definition:

- context $^{U,T}{}_{i}$ (0) = situation $^{U,T}{}_{i}$ (0) for $\forall i \in [G,S,U,N]$
- $\forall t>0$, context $^{U,T}{}_{i}$ (t) = situation $^{U,T}{}_{i}$ (t) $\otimes$ context $^{U,T}{}_{i}$ (t-1) where $\otimes$ denotes the composition operator and $i \in [G,S,U,N]$
- $\forall t \geq 0$, situation $^{U,T}{}_{N}$ (t) = situation $^{U,T}{}_{U}$ (t) $\cap$ situation $^{U,T}{}_{S}$ (t) = situation $^{U,T}{}_{S}$ (t) $\cap$ situation $^{U,T}{}_{U}$ (t)

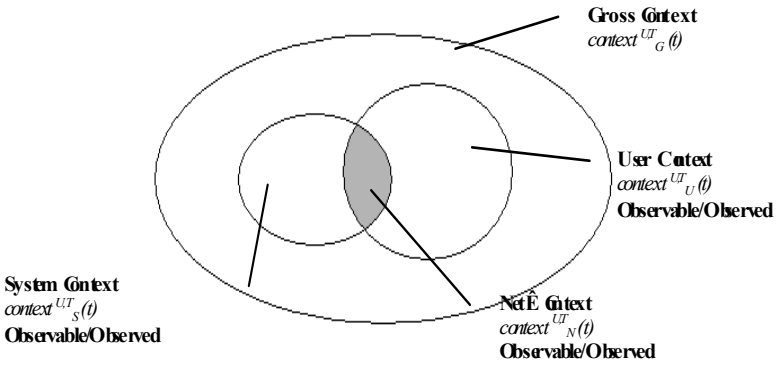

Figure 2. Relations between Gross, Net, User and System contexts.

As shown in Figure 2, the adjectives « observable » and « observed » make explicit the distinction between the design phase and the run time phase of the development process. In the design phase, designers specify the peripheral state variables that should be observed at run time. We then refer to the *observable context* and to the *observable situation*. They are potentially observed. At run time, they are effective: peripheral state variables, either are observed effectively or not observed at all due, for example, to some system or user failure or to a loss of conformity between the design and the implementation phases.

The refinement of the notion of context (situation) as observed or observable, the distinction between Gross, Net, System and User contexts provide a means to define *quality metrics for context-aware computing*. For example, the cardinality of the Net Context is a way to measure the conformity of the system context sensitivity to users' expectation. A cardinality close to 0 would mean that the context sensitivity of the system is nearly of no use to

the user. Similarly, the cardinality of the System Context compared to that of the Gross Context expresses the intrinsic sensitivity of the system. A cardinality of 0 denotes a context-unaware system. The comparison between an observable context and its corresponding observed context may provide some useful insights into the design process and/or on the human and system behaviors.

Having defined the foundations for our notion of context, we need to present one of its key components: the peripheral state variables.

3.4 Composing Situations

According to our definition, a context that relates to users U for a task T results from the composition of a set of situations. As shown in Figure 3, the composition can be modeled as a two-step process:

- For every situation S considered as part of the context, the peripheral state variables of S are labeled with the time of S. Then the situations considered in the context are combined using the Union operator on sets. So far, the relationships between state variables are kept unchanged.
- The second step consists of modifying the relations between state variables: new relations are created whereas others may be deleted.

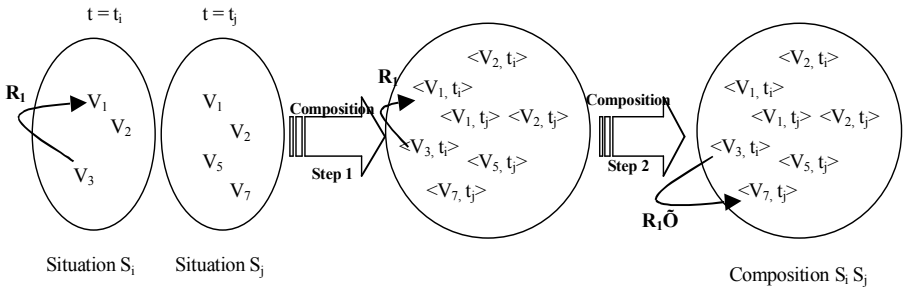

Figure 3. Capitalizing Situations to elaborate a Context.

For example, in Figure 3, Variables V_1 and V_3 express the temperature of the room in Farenheit and Celsius respectively. In the situation S_i, relation R_1 indicates that at t_i, V_1 is derived from V_3 whose value is produced by a physical sensor (e.g., a thermometer). At t_j (t_j>t_i), because the Celsius thermometer is out of order, V_3 does not belong to S_j anymore. However, another device has been able to record the value of V_3 at time t_i. This value is represented by V_7 in S_j. In addition, at t_j, a Farenheit thermometer has replaced the faulty Celsius device, hence the presence of V_1 in S_j. The rightmost picture of Figure 3 shows the set that describes the context at t_j after the composition function has been applied to S_i and S_j: the faulty thermometer

has been replaced by a new one, hence the presence of the couple "V_1, t_j". Because the value of V_1 at t_i may be useful in the future, it is kept in the context. Relation R_1 between V_3 and V_1 at t_i has been suppressed and replaced by a relation between V_3 and V_7.

Composition is a kind of history function that involves the emergence of new relations and peripheral state variables as well as the destruction of old ones. Although the concept of composition is easy to grasp, it requires well-thought design methods as well as sophisticated run time mechanisms. Run time mechanisms must be able to support the dynamic creation of relationships as well as the capacity to synthesize new peripheral state variables. Conversely, they must be able to eliminate ("garbage collect") useless state variables and relations. The risk is to erase information that may be useful in the future.

3.5 Generalization of the Notion of Context

We believe that context and situation can only be defined with respect to an entity for a given purpose. So far, we have developed the concept of context (and situation) in relation to a particular class of entities, the "user", for a particular class of purposes, the users' "task".

One way to generalize our definitions is to extend the scope of our first class entity to that of agents. An agent may be a human or a computational artefact such as an augmented physical object that interacts with the world, including humans. Thus:

Given a set of agents, A, a task, T, and two instants of observation, t_0 and t, where t_0 is the temporal reference for observations, the *Context* at t that relates to A for performing T, is the composition of the Situations observed between t_0 and t that relate to A for performing T:

$$\text{context}^{A,T}(t) = \text{COMPOSITION}(\text{situation}^{A,T}(t_0), \ldots, \text{situation}^{A,T}(t))$$

This generalization opens new issues including:

- Agents have different time scales. How can these be reconciled?
- What are the appropriate agents?
- What are the relationships between agent contexts?
- From an ethical perspective, how can we insure that the human agent has control over the computational world that is supposed to serve him?

In the following section, we describe the notion of contextor, a computational model we have derived from our definition of context.

4. CONTEXTOR: A COMPUTATIONAL MODEL FOR CONTEXT

Our notion of contextor is a conceptual extension of the context widget implemented in the Context Toolkit [17]. We first present the definition of a contextor followed in 4.2 by a taxonomy based on the functional aspect of contextors. Then, in 4.3, we show how contextors can be combined into richer computational units. Whether contextors are simple or composed, their properties, described in 4.3, have an impact on implementation mechanisms illustrated in 4.4.

4.1 Definition

A *contextor* is a software abstraction that models a relation between variables of an Observed System Context. As defined in Section 3.3, the Observed System Context is the composition of situations as observed by the system. From the values of a set of variables of an Observed System Context, a contextor returns the value of a variable (or of a set of variables) that belongs to that context.

As shown in Figure 4, a contextor is comprised of a functional core and of *typed input and output communication channels*.

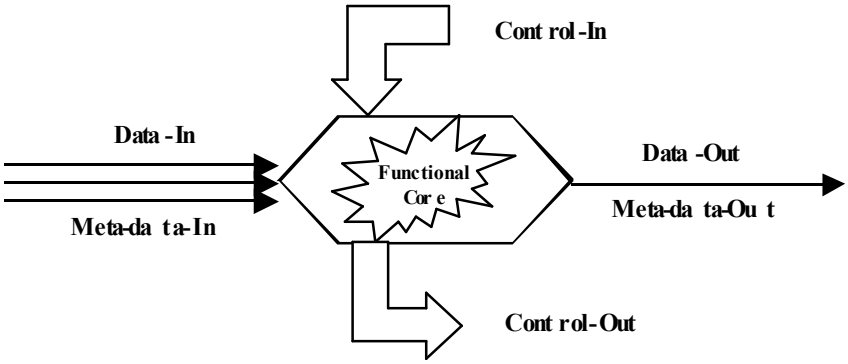

Figure 4. A graphical representation of a contextor.

The *functional core of a contextor* implements a relation between variables of the Observed System Context. For example, in Figure 3, relation R1 between variables V1 and V3 of situation Si can be computationally represented by a contextor whose functional core consists of translating Farenheit data into Celsius representation.

The input channels of a contextor are of two types:

– *Data-In* corresponds to the variables of the Observed System Context that are used as inputs by the functional core of the contextor. Every in-

put value is decorated with a *Meta-data-In* that expresses the quality of the input value.

- *Control-In* corresponds to commands received from other contextors to set the internal parameters of the contextor. These parameters concern the functional behavior of the contextor as well as the non-functional behavior such as the QoS (Quality of Service) expected by other contextors. For example, a contextor may receive a "switch off" command on its Control-In because it has been recognized as faulty by another one. Or, it may receive a QoS request that expresses the level of precision of the values delivered on the Data-out channel.

Symmetrically, the output channels of a contextor are of two types:

- *Data-Out* corresponds to the values of some variables of the Observed System Context returned by the contextor. As for input, output data are decorated with *Meta-data-Out* that describes the quality of the output produced by the contextor.
- *Control-Out* is used by the contextor to send control commands to other contextors. For example, based on the meta-data associated with the data received from a contextor C, a contextor may decide to send a "switch off" command to C.

In addition,

- a Data-In channel receives data from a Data-out channel whose data type is compatible with that of the Data-In channel.
- a Control-In channel receives data from a Control-Out channel whose data type is compatible with that of the Control-In channel.
- The connections between input and output channels may be static (i.e., wired by the implementer) or semi-static (i.e., computed at run time when the system is launched), or transient (i.e., may be changed dynamically).

Given a contextor C, a *source contextor* is a contextor that provides C with Data-In values, and a *sink contextor* is a contextor that receives Data-Out values from C.

4.2 Taxonomy

By analogy with the software tools and concepts developed in Human Computer Interaction, we propose a taxonomy for contextors based on the recurrence of the functional requirements of context aware systems. Table 1 shows an overview of the different types of contextors that we present in more detail in the following subsections.

Table 1. Contextor types based on their functional core. X, Y, Z denote distinct data types, but Z is at a higher level of abstraction that X and Y. {X} represents a set of values of type X.

Contextor type	Sensor connectivity	Number of input channels	Number of output channels	Input data types	Output data types
Elementary	Yes	0	1	-	X
History	No	1	1	X	{X}
Threshold	No	1	1	X	Boolean
Translator	No	1	1	X	Y
Fusionor	No	2 or more	1	X,X	X
Abstractor	No	1 or more	1	X,Y	Z

4.2.1 Elementary Contextor

An elementary contextor has no Data-In Channel. It offers a convenient way to encapsulate a physical sensor such as a thermometer, or any computational component that can be used as a source of physical data.

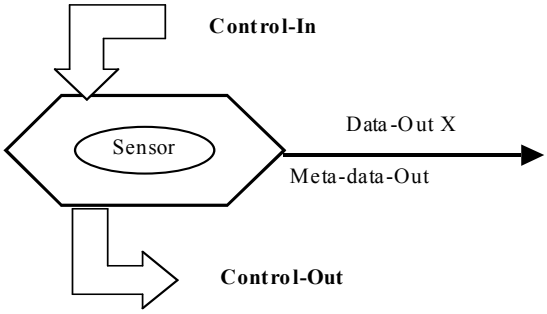

Figure 5. The elementary contextor.

For example, suppose we encapsulate a video camera within an elementary contextor. Then, Data-out corresponds to the video flow of images, Control-in allows other contextors to set the video rate, as well as the pan-tilt-zoom factors of the camera. Meta-data-Out can be used to express the resolution, the number of bits per pixel, and more generally, any factor that qualifies the images delivered by the contextor. Control-out is used by the contextor to acknowledge the execution of the commands received through the Control-in channel.

4.2.2 History Contextor

A history contextor saves the values and their meta-values that it has successively received through its Data-In channel. For example, the Data-In channel of a history contextor receives temperature values provided by a Thermometer elementary contextor. By definition, the history contextor saves the values it receives. Its Control-in channel allows other contextors to set the number of values to be maintained or the time interval before initiating garbage collection. Through its Control-Out, the history contextor is able to notify source contextors to slow down their emission flow rate because out of memory is nearly reached.

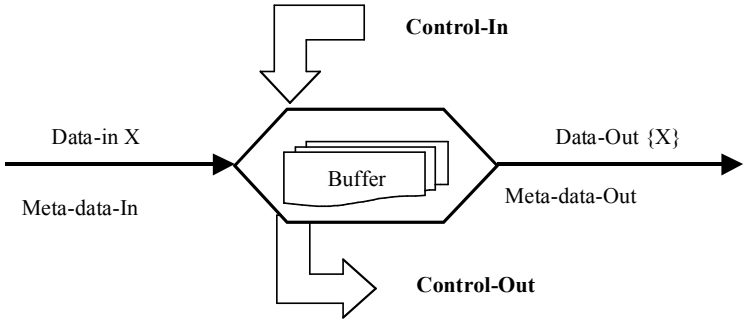

Figure 6. History contextor.

4.2.3 Threshold Contextor

A threshold contextor returns true if the Data-In value satisfies a threshold condition, false otherwise.

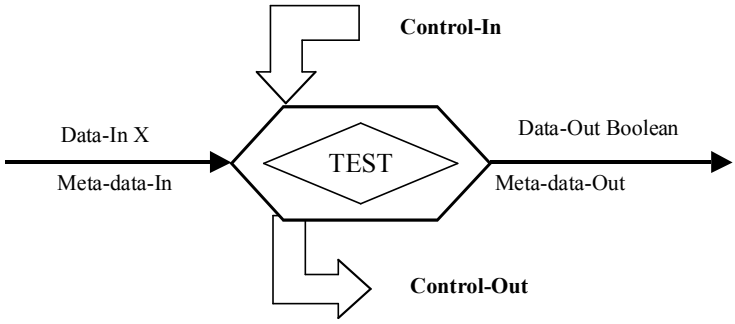

Figure 7. The Threshold contextor.

For example, suppose a threshold contextor receives temperature values from a thermometer elementary contextor. Data-Out expresses the answer to the question "Is it hot?". The Control-In channel allows the sink contextor to set the temperature threshold. Typically, in winter, the threshold would be set to 22^0C whereas 35^0C would be more adequate for summer. Through the Control-Out channel, the threshold contextor can notify its source contextor to change the flow rate of its Data-Out channel.

4.2.4 Translation Contextor

A translation contextor performs type recasting but does not change the meaning, nor the level of abstraction of the values received on the Data-In channel.

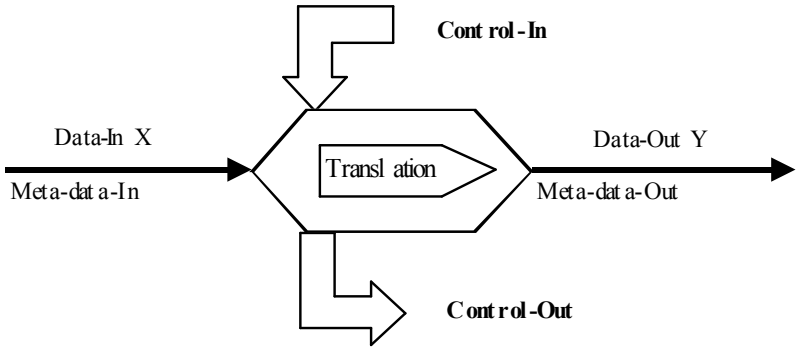

Figure 8. The Translation contextor.

For example, a translation contextor is used to transform input temperatures from one representation system to another (for example Celsius, Farenheit, Kelvin). The Control-In channel allows the sink contextor to set up the desired data type for the Data-Out channel.

4.2.5 Fusion Contextor

The Fusion contextor has multiple Data-In of the same type, each one with its own meta-data. The role of the fusion contextor is to produce a single Data-Out of the same type whose quality has been improved over that of the input data. Typically, a fusion contextor receives the number of persons in a room both from a video tracker and an audio tracker. It renders the number of people in the room but with a better confidence factor than that provided by the video and the audio trackers. Control-In is used to express the relative importance of the source contextors, whereas the Control-out sends commands to source contextors such as start/stop sending values.

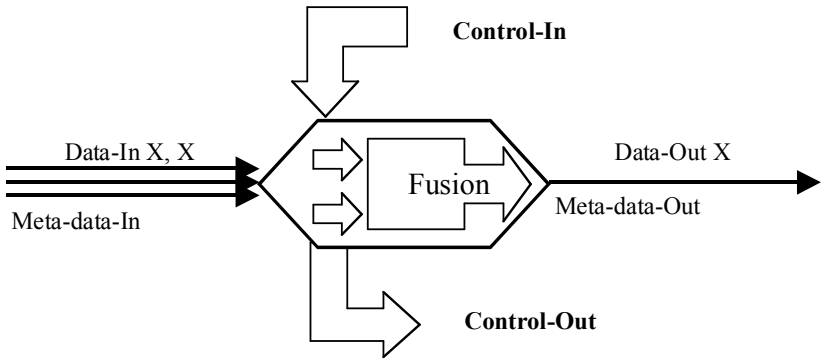

Figure 9. The fusion contextor.

4.2.6 Abstraction Contextor

The Fusion contextor has multiple Data-In. The role of the abstraction contextor is to produce a single Data-Out whose type is at a higher level of abstraction than that of the input data types.

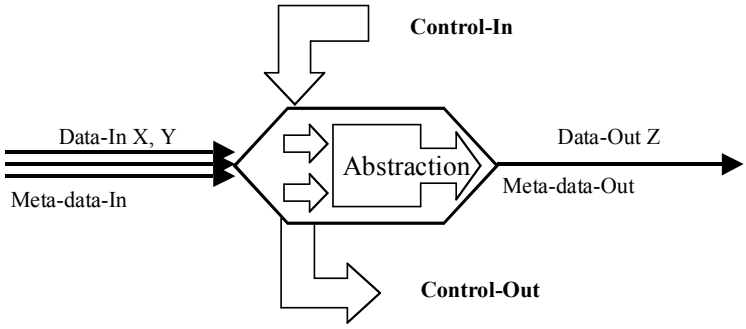

Figure 10. The abstraction contextor.

Example: A source contextor provides an abstraction contextor with the weight of moving things on the floor of a museum room. Another one provides the abstraction contextor with the ambient temperature of the room. From these two input streams, the abstraction contextor returns the level of occupancy of the room.

So far, we have introduced the basic classes of interactors. In the following section, we consider their composition.

4.3 Composing Contextors

Contextors may be composed in two ways: through data channels connection and via encapsulation.

4.3.1 Data Channels Connection

Contextors are composed by connecting Data-In channels with compliant Data-Out channels to form an oriented graph. Two channels are compliant if they convey data of the same type. As shown in Figure 11, the resulting oriented graph forms a colony of contextors. A colony of contextors is an oriented graph where Data-In channels are connected to compliant Data-out channels. Source interactors at the base of the graph are elementary contextors. Contextors at the top of the graph provide applications with contextual data at the appropriate level of abstraction.

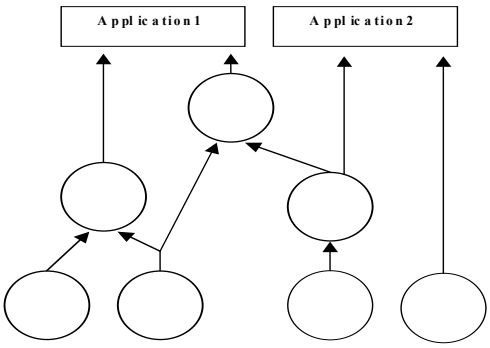

Figure 11. A colony of contextors.

We have not devised yet the connection rules between control channels. For now, we can say that the Control-in channel of a contextor is connected to the Control-out channel of its sink contextors. Consequently, the Control-out channel of a contextor is connected to the Control-in channel of its source contextors.

4.3.2 Encapsulation

Encapsulation is used to group a colony of contextors as a new class of contextor whose internal composition is hidden to other contextors. Figure 12 shows an example of an abstraction contextor built from a colony of simpler contextors.

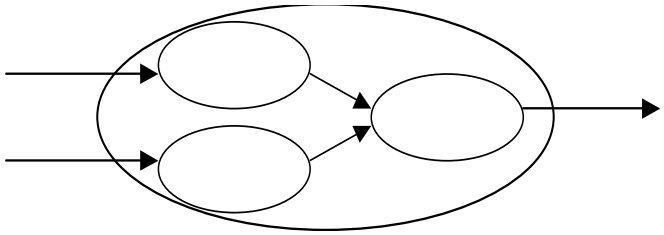

Figure 12. A colony of contextors encapsulated as a reusable class of contextors.

4.4 Quality of Service and Properties

In a world of uncertainty, the system must be able to auto-evaluate the quality of the services it provides in order to adapt its behavior. As a result, contextors are capable of measuring their own QoS.

So far, we have identified metrics for elementary contextors. These include: resolution, latency, sample rate, stability, field of perception, field of action, autonomy, etc. (see [16] for more details).

In addition, a contextor satisfies the following properties:

- Reflexivity: a contextor is auto-descriptive so that its class as well as its input and output channels are discovered by other software components (including other contextors). In addition, reflexivity provides the contextor with the capability to adapt its behavior based on the QoS requests received through its Control-In channel.
- Remanence: a contextor is able to disappear from its colony and, if necessary, to restore its state when resuming its execution within the colony (or within another colony).
- Mobility: a contextor may migrate within its colony by dynamically reconfiguring the connectivity of its input and output channels.

4.5 Illustration

To test our notion of contextor, we have implemented a server of contextors in Java. In this environment, a contextor is an autonomous software component with a life cycle that includes the following states: the contextor exists as a file but is unknown from the server; the contextor is identified by the server but is not running; the contextor is running; the contextor is suspended. The details of the contextors server can be found in [16].

Figure 13 shows a simple contextor-based system that we have developed as an answer to a real life experience. In our lab, coffee cups are shared resources. As such, they are supposed to be kept in the cafeteria, at all time. However, we often end up sipping our coffee in the office and forget to bring the cups back at the appropriate location. As a result, flame messages are

sent through email by the first person with no coffee cup available. Our system is aimed at automatically sending a warning email message when the number of cups available in the cafeteria is below a threshold.

Our contextor-based system includes 2 elementary contextors whose data-Out channel is connected to the data-In channels of an abstraction contextor: the first elementary contextor encapsulates a camera sensor whose function is to count the number of cups in the cafeteria. The second elementary contextor computes the threshold whose value depends on the time of the day as well as on the date. The abstraction contextor synthesizes a new variable "cups will soon be missing" at the appropriate level of abstraction. The value of the variable is sent to the application which in turn can broadcast a warning message to the people of the lab

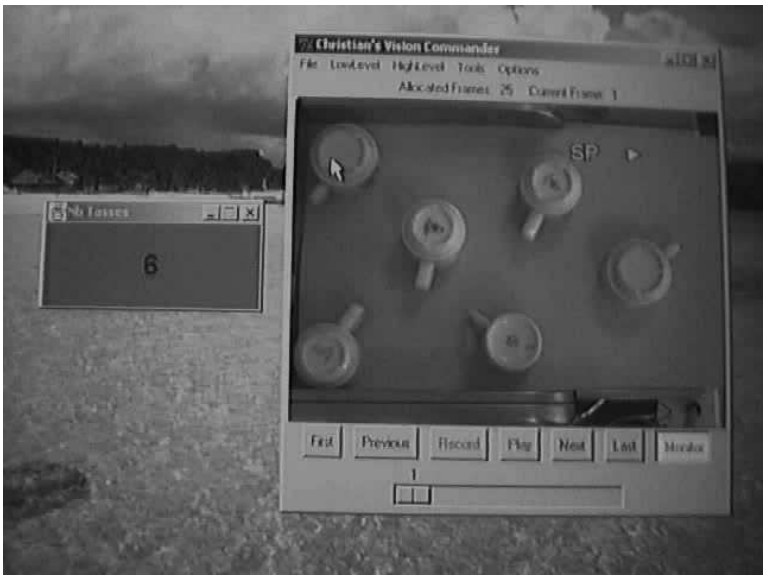

Figure 13. On the right, the scene as captured by the video contextor. On the left, the small red windows pops up to notify that cups should be brought back to the cafeteria.

5. CONCLUSION

In this article, we have presented a working definition for the notion of context for the design and development of context-sensitive systems. This definition draws upon the following key ideas:

– The distinction between the notion of an instant snapshot of "things" in the form of a state vector (a situation a t), and the composition of these state vectors over time (a context at t),

- The distinction between the user's view and the system's view of context (and situation), and their intersection to form the net context (situation). In turn, the cardinality of the net context can be used as a basis for evaluation,
- The distinction between the designer's view (the observables) and the run time view of context (the effectively observed variables) and their intersection usable as evaluation metrics,
- The transposition of the relations between the observables as software abstractions (the contextors),
- A common I/O structure for all contextors including control channels to ensure and express QoS (e.g., precision, stability), as well as common properties such as reflexivity and remanence,
- The definition of core contextors serving recurring functional requirements in ubiquitous computing (sensor encapsulators, translators, abstractors, etc.),
- The composition of contextors as oriented graphs or encapsulated into higher computational units.

There are very few tools available for the development of context aware systems. The Context Toolkit is a notable exception [17]. Although similar in spirit, the contextor model applies in a more systematic way at every level of abstraction. One important aspect of the contextor is the notion of meta-data and control. Salber introduced similar ideas, but with subtle differences. In particular, in the contextor model, meta-data are not sent on a separate channel: they are decoration of the data in order to insure synchronicity. In addition, we generalize the role of the control channels as a way to dynamically express Quality of Service.

Although our model of context and contextors needs to be evaluated against large scale applications, it sets the foundations for a systematic approach to context-aware computing.

ACKNOWLEDGMENT

This work has been partly supported by the EEC project TMR TACIT (ERB-FMRX-CT-97-0133) and by the FET GLOSS project (IST-2000-26070). It has been conducted with the participation of G. Calvary, P. Reignier and J.L. Crowley.

REFERENCES

[1] G. Abowd. Towards a Better Understanding of Context and Context-Awareness Workshop on The What, Who, Where, When, Why and How of Context-Awareness *Extended Abstract of the 2000 Conference on Human Factors in Computing Systems (CHI'2000)*, G Szwillus & T Turner (eds), ACM Press, 2000, pp. 371.

[2] J. E. Bardram. Plans as Situated Action: An Activity Theory Approach to Workflow Systems.Proceedings of the *ECSCW'97 Conference*, Lancaster UK, September 1997.

[3] H. Beyer, K. Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufman Publ., 1998.

[4] P.J. Brown, J.D. Bovey, X. Chen. Context-Aware applications: From the Laboratory to the Marketplace. *IEEE Personal Communications*, 4(5), 1997, pp.58-64.

[5] G. Calvary, J. Coutaz, D. Thevenin. A Unifying Reference Framework for the Development of Plastic User Interfaces. IFIP WG2.7 (13.2) Working Conference, EHCI01,Toronto, May 2001, Springer Verlag Publ., LNCS 2254, M. Reed Little, L. Nigay Eds, 2001, pp.173-192.

[6] G. Calvary, J. Coutaz, D. Thevenin. Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism, *in Proc. HCI-IHM 2001*, A. Blandford, J. Vanderdonckt, P. Gray Eds.,BCS conference series, Springer Publ., pp. 349-363.

[7] K. Cheverst, N. Davies, K. Mitchell, C. Efstratiou. Using context as a Crystal Ball: Rewards and Pitfalls, *Personal and Ubiquitous Computing*, Springer Ed., 5(1), 2001, pp. 8-11.

[8] A. Dey, G. Kortuem, D. Morse, A. Schmidt. Situated Interaction and Context aware Computing. *Personal and Ubiquitous Computing*, 5(1), Springer publ., 2001, pp. 1-3.

[9] A. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1), Springer publ., 2001, pp. 4-7.

[10] G. W. Fitzmaurice. Situated Information Spaces and Spatially Aware Palmtop Computers, *Communication of the ACM*, 7(36), July 93, pp. 39-49.

[11] C.A. Halverson. *Distributed Cognition as a theoretical framework for HCI: Don't throw the Baby out with the bathwater - the importance of the cursor in Air Traffic Control*. Tech Report No. 94-03, Dept. of Cognitive Science, University of California, San Diego, 1994.

[12] S. Harrison, P. Dourish. Re-Place-ing Space: the roles of Place and Spaces in Colaborative Systems. *In proc. CSCW'96*, ACM publ., 1996, pp. 67-76.

[13] B. Nardi, *Context and Conciousness. Activity Theory and Human Computer Interaction*, MIT Press, Cambridge, Massachusetts, 1996.

[14] F. Paternò. *Model-based Design and Evaluation of Interactive Applications*, Springer Verlag, 1999.

[15] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. Proc. Of the *2nd International Symposium on Wearable Computers*, 1998, pp. 92-99

[16] G. Rey. *Systèmes Interactifs Sensibles au Contexte. Ecole doctorale de Mathématiques et Informatique*, DEA d'Informatique, Systèmes et Communications, Université Joseph Fourier et Institut National Polytechnique de Grenoble, June, 2001, 84 pages.

[17] D. Salber, A.K. Dey, G. Abowd. The Context Toolkit: Aiding the development of context-enabled Applications. *In Proc. CHI99*, ACM Publ., 1999, pp. 434-441.

[18] B.N. Schilit, N.I. Adams and R. Want. Context-Aware Computing Applications. Proceedings of the IEEE Workshop on *Mobile Computing Systems and Applications* (WMCSA'94), IEEE Press, 1994, pp. 85-90.

[19] A. Schmidt. Implicit Human Computer Interaction Through Context. *Personal Technologies* Volume 4(2&3), June 2000, pp.191-199.
[20] L. A. Suchman. *Plans and situated actions. The problem of human-machine communication*. Cambridge: Cambridge University Press, 1987.
[21] D. Thevenin, J. Coutaz. Plasticity of User Interfaces: Framework and Research Agenda. In Proc. *Interact99*, Edinburgh, A. Sasse & C. Johnson Eds, IFIP IOS Press Publ., 1999, pp.110-117.
[22] Ward, A. Jones, A. Hopper, A New Location Technique for the Active Office. IEEE *Personnal Communications*, 1997, pp. 42-47.
[23] M. Weiser.The Computer for the 21rst century, *Scientific American*, 265(3), 1991, pp. 94-104.
[24] Dagstuhl Seminar on Ubiquitous Computing, http://www.inf.ethz.ch/vs/events/dag2001/, sept. 2001.