THE CONTEXTOR: A COMPUTATIONAL MODEL FOR CONTEXTUAL INFORMATION

Gaëtan Rey, Joëlle Coutaz

CLIPS-IMAG
385 rue de la Bibliothèque
F-38041 Grenoble cedex 9, France
{Gaetan.Rey, Joelle.Coutaz}@imag.fr

James L. Crowley

Laboratoire GRAVIR, INRIA Rhône Alpes,
655 Ave de l'Europe, F-38330 Montbonnot, France
Jim.Crowley@ imag.fr

ABSTRACT

This position paper proposes the contextor as a computational abstraction for modeling and computing contextual information. Here, context must be understood as a state vector of observables. An observable is a variable whose value can be acquired through sensing technology and or computed by the system. 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 ramanence. They can be combined as oriented graphs or encapsulated into higher computational units. We show how they fit into the Arch reference architectural model.

KEYWORDS : contextor, interaction context, context modeling, software architecture modeling, Human Computer Interaction, ubiquitous computing.

1. INTRODUCTION

Context is an old friend. The literature reveals a large body of research based on the notion of context, typically in Linguistics, AI, Computer Vision, and HCI. With the emergence of ubiquitous computing, researchers are rediscovering this notion, using different perspectives for serving distinct purposes. It is not surprising then than no one agrees on a common definition. From the seminal work developed on context computing [Moran 02], one can draw the following four lessons:

. Lesson 1: Context can only be defined in relation to a purpose. In this position paper, context is defined for the purpose of computational perception (e.g., perception of user's implicit actions, sensing physical environment, self-discovery of local devices). Context is thus a state vector of observables. An observable is a variable whose value can be acquired and or computed by the system.

. Lesson 2: Context is an information space that serves interpretation [Winograd 02]. In our work, interpretation is performed by the system for perceiving for users' benefits.

. Lesson 3: Context is a shared information space. In our work, context sets a common ground between a system and a user. It is therefore observable by humans.

. Lesson 4: Context is an open information space: it evolves. As a result, we make a distinction between a situation (i.e., a snapshot of observables) and the composition of situations, which, in turn, defines a context.

In [Crowley 02], we present an ontology of situation and context for machine perception as well as a computational model for perceptual processes. In this position paper, we propose the *contextor* as a generalisation of the perceptual processes, as well as of the context widget implemented in the Context Toolkit [Salber 99]. Because in HCI it is good practice to analyze a problem from at least two complementary perspectives (the user and the system perspectives), we need first to introduce the notions of *use rcontext* and *system context*. We then present the *contextor* as the building block for developing system context, and show how contextors can be combined as richer computational units. Finally, we present how federations of contextors fit into the Arch reference architectural model.

2. SYSTEM CONTEXT AND USER CONTEXT

As shown in Figure 1, the notions of *user context* and *system context* are considered for a user U in relation to a task T. User and system contexts are related to the *gross* and *net* contexts. The *gross context*, $context^{U,T}_{G}$, covers the universal facts (variables and their relations) that relate to U for performing T. The *system context*, $context^{U,T}_{S}$, is the subset of the gross context that concerns the system. Similarly, the *user context* that relates to U for performing T, $context^{U,T}_{U}$, is the subset of the gross context that concerns the user U for T. The *net context*, $context^{U,T}_{N}$, is the subset of the gross context that the user and the system have in common for T: it constitutes the common ground mentioned in the introduction.

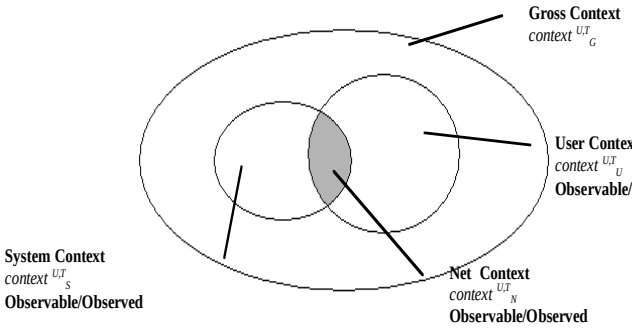

Figure 1. Relations between Gross, Net, User and System contexts.

As shown in Figure 1, 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 state variables that should be observed at run time. Since the state variables are potentially observed, we refer to the *observable context*. At run time, the 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. We then refer to the *observed context*.

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's sensitivity to users' expectations. A cardinality close to 0 would mean that the context sensitivity of the system is of nearly 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.

3. THE NOTION OF CONTEXTOR

A *contextor* is a software abstraction that models a relation between variables of the Observed System Context. 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 1, a contextor is composed of a *functional core* and of *typed input* and *output communication channels*.

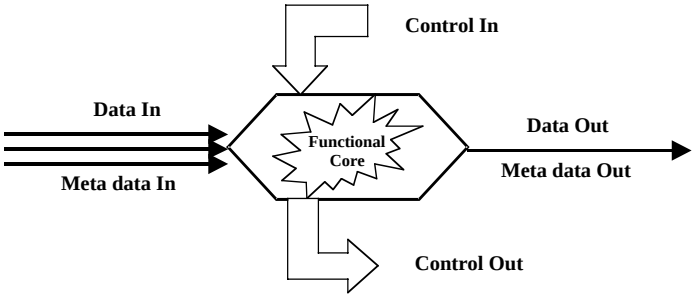

Figure 2. Graphical representation of a contextor.

The *functional core* of a contextor implements a relation between variables of the Observed System Context.

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 input 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. Or, it may receive a QoS request that expresses the level of precision of the values required for 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 (e.g., resolution, latency, sample rate, stability, field of perception, field of action, autonomy, etc.).

. *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.

. Implementation of contextors must ensure reflexivity and remanence.

4. COMPOSING CONTEXTORS

Contextors may be composed in two ways: by connecting data channels and by means of encapsulation.

4.1. Data Channels Connection

Data-In channels can be connected with compliant Data-Out channels to form a federation. Two channels are compliant if they convey data of the same type. As shown in Figure 3, the resulting oriented graph forms a federationwhere source interactors at the base of the graph are elementary contextors, and where contextors at the top of the graph provide applications with contextual data at the appropriate level of abstraction. The Control-in channel of a contextor can be 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*.

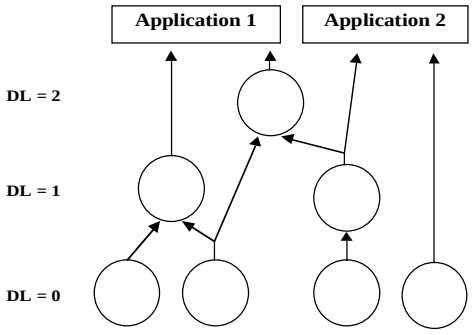

Figure 3. A federationof contextors.

In a federation, a contextor can be characterized by a dependence level (DL) that can be exploited to evaluate the cost of the dynamic reconfiguration of the colony.

4.2. Encapsulation

Encapsulation is used to group a federationof contextors as a new class of contextor whose internal composition is hidden to other contextors. Figure 4 shows an example of an abstraction contextor built from a federationof simpler contextors.

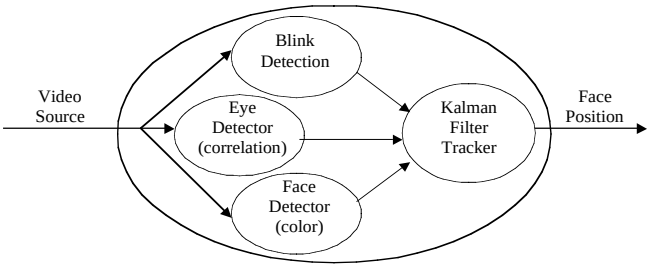

Figure 4. A federationof contextors encapsulated as a reusable class of contextors.

Having presented the composition of contextors, let us see how these compositions are integrated in the architecture of an application using Arch as a reference model.

5. THE AUGMENTED ARCH MODEL

The Arch model is an efficient conceptual model for devising the overall functional structure of an interactive system [Arch 92]. With PAC-Amodeus, we have extended, refined, and exploited Arch in many ways for the development of multimodal user interfaces [Nigay 95]. Similar in spirit to Salber's proposal and in accordance with the ontology presented in [Crowley 02], we propose to extend Arch with a four-layer "context branch" (See Figure 5).

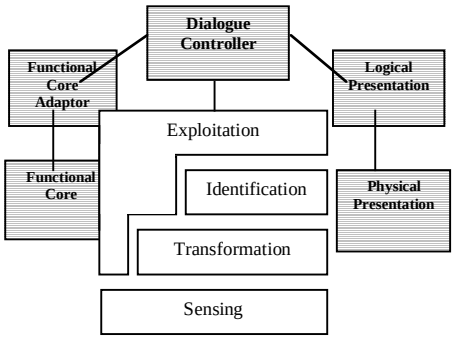

Figure 5. The augmented Arch model for context-aware computing.

The *Sensing* layer is built from elementary contextors, i.e., contextors that encapsulate physical sensors. It is to context what the Physical Presentation Component is to classic user interfaces.

The *Transformation* layer, built from chains of contextors and/or from encapsulation, provides contextual information independent from the physical sensing technology and at the "right" level of abstraction. It is to context what the Logical Presentation Component is to classic user interfaces.

The *Identification* layer detects situation and context changes, and identifies the current situation and context. This layer can be implemented using a blackboard approach as in [Winograd 02].

The *Exploitation* layer acts as an adaptor between the Dialogue Controller and the context-aware computing portion of the system. For performance reasons, it may skip the "Identification" layer and exchange information directly with the "Transformation" and "Sensing" layers.

As in Arch, the slinky meta-model applies: some layers may not exist, and functions may shift between layers. For multi-modal interaction, each modality supported by the system (e.g., direct manipulation, speech), gives rise to a branch. But a run time mechanism, such as Nigay's fusion engine, manages the dependencies between the branches [Nigay 95]. Similarly, the "context branch", which appears as an independent functional branch in Figure 5, calls upon a run time mechanism to express relationships with its neighbour branches. For example, a sensing layer that detects the presence of multiple PDA screens aligned close to each other, may inform the Physical Presentation Layer of the availability of a large screen resource built from a mosaic of small screens. The graphical user interface may then adapt accordingly [Calvary 01]. Conversely, keyboard and mouse inputs may be used by the "Sensing" layer to inform higher layers of the "Context branch" that the user is currently active.

6. CONCLUSION

The models and principles presented in this position paper are based on our experience in the development of multimodal interaction as well as of computer vision-based sensing technology. By analogy with interators, contextors are motivated by the benefits of the object-oriented distributed technology. We are currently implementing contextors for the European project GLOSS using a P2P approach. As discussed in Section 5, we do not promote a single uniform paradigm for context computing. Instead, we suggest that the higher levels of the "Context branch" should draw upon AI-based techniques such as the blackboard.

7. REFERENCES

[Arch 92] UIMS Tool Developers' Workshop. A meta-model for runtime architecture of an interactive system. SIGCHI Bulletin, 24(1):32{37, 1992.

[Calvary 01] 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.

[Coutaz 02] J. Coutaz, G. Rey Foundations for a theory of contextors. In Proc CADUI02, ACM Publ., 2002, pp. 283-302.

[Crowley 02] J. L. Crowley, J. Coutaz, G. Rey. Perceptual Components for Perceptual Computing. In Proc. **Ubicom (Ubiquitous Computing)**2002.

[Moran 02] Moran, T. P. and Dourish, J. P. (editors). Special Issue on Context-Aware Computing. Human Computer Interaction, Volume 16, Numbers 2-4. Erlbaum.

[Nigay 95] L. Nigay, J. Coutaz. A Generic Platform for Addressing the Multimodal Challenge, CHI'95, ACM New York, Denver, May 1995, pp. 98-105.

[Salber 99] 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.

[Winograd 02] T. Winograd. Architecture for Context, Human Computer Interaction, Lawrence Erlbaum Ed., Vol. 16, pp. 401-419.