An Architecture for Developing Context-aware Systems

Kaiyu Wan, Vasu Alagar, and Joey Paquet

Concordia University, Montreal, Canada
{ky_wan,alagar,paquet}@cse.concordia.ca

1 Introduction

In this paper we discuss an architecture, based on a three-tiered formalism, and a development method for constructing *context-aware* systems. As an illustration of the principles involved in developing a context-aware system, we present a formal architecture of the *Antilock Braking System* (ABS).

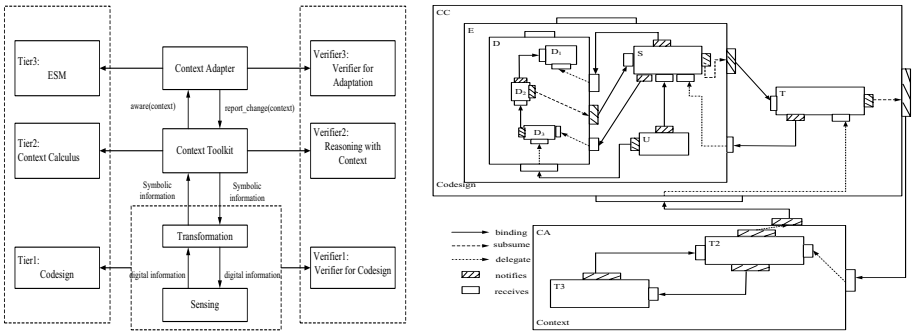
The distinguishing features of a context-aware system are *perception* and *adaptation*. The perception feature makes the system aware of the entities in the region of its governance, and triggers context-driven interaction among the entities. The nature of interaction is in general heterogeneous, with varying levels of synchrony, concurrency, and delay. However, the system is to be fully controlled and guided by the time-varying contextual conditions and system's progress should remain both predictable and deterministic. In order to achieve determinism and predictability, the system adapts to the stimuli from its environment by learning the changing relationship among the entities and acting to fulfill the intentions expressed by the entities.

The example of a context-aware system taken up for illustration in this paper is the ABS available in most of the modern day cars. We discuss ABS architecture in Section 4. The rest of the paper is organized as follows: In Section 2 we discuss the rationale for an architecture based on a three-tiered formalism, provide an overview of each tier and give a formal description of the architecture. In Section 3 we show the abstract architecture of the context-aware systems. We conclude the paper in Section 5 with a review of our ongoing work.

2 Three-tiered Formalism

Context-aware systems are notoriously heterogeneous in terms of device types, context interpretations, and adaptation requirements. We introduce a three-tiered formalism as a solution to this heterogeneity problem. Perception involves the objects perceived, the devices used for observing the objects, and the observational measurements. The objects and the devices are low level abstractions belonging to one tier. The observations of the device signals are converted to symbolic forms. We formally describe the structures composed from elementary symbolic representations using mathematical concepts. This description belongs to one tier, which links the low level abstraction to the high-level adaptation abstraction. Thus we end up with a three-tiered formalism. The architecture that we propose encapsulates the three tiers into two components, one of which deals with context construction and the other deals with context adaptation.

Tier 1 formalism is a description of "*see*, *gather*, *control*, and *modify*" features of perception abstractions. It describes the assembled *symbolic* representations of the observations and *notifies* Tier 2. Tier 2 *receives* the notification from Tier 1, and constructs contexts that reflect the current awareness. The formal basis of Tier 2 functionality is the context theory explained in [6]. Tier 2 uses the *context calculus toolkit* provided by the theory to construct general contexts, deconstruct and modify them. Tier 2 *notifies* current context information to Tier 3. Tier 3 *receives* current context from Tier 2 and determines the corrections to be made to the current awareness. The modified context information is given to Tier 1 through Tier 2. Figure 1(a) shows the three tiers and their formalisms. Below we give an overview of the formal notation used in the three tiers.



(a) Three-tier Formalism (b) Architecture of Context-aware System

Fig. 1. Formalism and Architecture

Tier 1 formalism is a description of "*see*, *gather*, *control*, and *modify*" features of perception abstractions. It describes the assembled *symbolic* representations of the observations and *notifies* Tier 2. In describing Tier 1, we consider the environment, sensing mechanism, and functional transformation of the observed raw data. The entities in the environment seen by Tier 1 include users, programmable parts, sensors, and actuators. We filter away the heterogeneity in the set of entities by encapsulating the behaviour of each entity separately as a *Codesign Finite State Machine*(CFSM) [1]. The CFSMs communicate asynchronously through message (event) passing. To ensure quality of service, the entire design will be optimally partitioned for software and hardware implementations. In this paper we do not discuss hardware/software partitioning issues.

Tier 2 discusses formalism based on set theory and relations for contexts. The context information is in general *multidimensional*, where in each dimension there exists several choices to be considered. As an example, a physical context for a braking device includes certain values (rotation per minute for wheels), situation (wheels locked or not), environmental conditions (friction on the road, gradient of road surface), and user characteristics (applying brakes or not). Here we have enumerated four dimensions. In each dimension, there are several possible ways to represent information. We

say information in each dimension is *tagged*, and given the set of dimensions (dimension names), and a tag set for each dimension, then it is easy to see why the context should be defined as a of a finite union of relations. We define contexts as subsets of a union of relations which in turn characterize the possible worlds [6]. The rationale for the choice of formalism is that relational semantics provides a formal basis for supporting the representation and manipulation of different types of contexts that arise in context-aware applications. Contexts arising in both discrete and continuous interaction of entities can be casted as relations. Both descriptive and prescriptive context information can be modelled as relations. Although finiteness is not insisted, in our discussions we deal with contexts having a finite number of dimensions. Hence, both an enumeration syntax and a presentation syntax are used for context presentation. The context toolkit includes the mathematical definition and representation of contexts, context operators, and context expressions.

The adaptive control mechanism in Tier 3 is modelled by an *Extended State Machine* (ESM) model, which has been used to model real-time reactive systems [5]. At each state, the ESM may receive context information composed by T_2, query the context database and/or compute a mathematical function, and output the results to *CC*. The state transition semantics is

$$\frac{s_i \ \wedge \ e \ \in \mathcal{E}(s_i) \ \wedge \ var_g(s_i) \ \wedge \ con_g(s_i) \ \wedge \ tc(t)}{s_i \ \xrightarrow{e} \ s_j \ \wedge \ var_a(s_j) \ \wedge \ rtc(t)}$$

where (1) var_g is a predicate formula, in which each atomic predicate expresses a constraint on one variable in state s_i; (2) con_g is also a predicate formula, in which each atomic predicate expresses a constraint on context information in state s_i; and (3) tc is a conjunction of linear time predicates of the form $Lower \ \leq \ t \ \leq \ Upper$, where t is the valuation of a local clock. An execution in the ESM is a sequence of transitions starting from an initial state. The behaviour of the ESM is the set of executions.

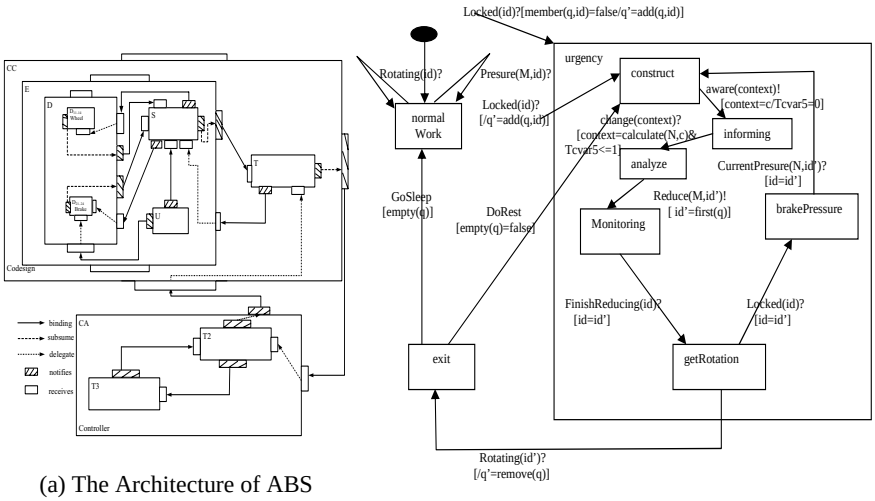
3 Architecture of the System

In this section we give a component model of the three tiers, following the terminology and notation in [4]. A component type $\mathbf{T} = \langle \mathbf{F}, \mathbf{A} \rangle$ defines a black-box view, called frame $\mathbf{F}$, and a grey-box view, called architecture $\mathbf{A}$. The type $\mathbf{T}$ may include more than one grey-box view. The black-box defines the interface types, and the particular grey-box view $\mathbf{A}$ as a structured implemented version of $\mathbf{F}$. An interface of a component is an instance of either *notifies-interface* type or *receives-interface* type. The four kinds of ties between the interfaces of two component instances A and B are as follows: (1). *binding*($\xrightarrow{bind}$) of a receives-interface to a notifies-interface between two components at the same level of nesting, (2). *delegating* ($\xrightarrow{delg}$) from a receives-interface of a component A to a receives-interface of a subcomponent B of A on the adjacent level, (3). *subsuming* ($\xrightarrow{subs}$) from a notifies-interface of a component B to a notifies-interface of a component A, where B is a subcomponent of A on its adjacent level, and (4). *exempting* ($\xrightarrow{exem}$) an interface of a component from participating in the architectural connection.

Following the above conventions we describe the architecture construction of a context-aware system. We encapsulate Tier 1 as a component CC, which constructs contexts out of the observables. Component CC is a composition of two components: component E modelling the environment and the primitive component T modelling the transformation unit that constructs elementary contexts from the observables in the environment. Component E is composed from component D modelling the collection of devices, the primitive component U modelling the user, and the primitive component S modelling the sensing mechanism, which is a central unit sensing all the information the system needs. The component D is composed from subcomponents D_i, $i = 1, \ldots, k$, where each component modelling one device is primitive. We compose the primitive component T_2 corresponding to Tier 2 and the primitive component T_3 corresponding to Tier 3. The resulting component CA is the context adapter. A context-aware system is a composition of CC and CA, as shown in Figure 1(b).

4 Case Study - Antilock Braking System (ABS)

ABS is a typical example of a context-aware system, in which both situated and environmental information should become part of a context. Our design is more general and formal than the one suggested in [3].



(a) The Architecture of ABS

(b) Behavior Model of Context Adapter of ABS

Fig. 2. The Architecture of ABS and Formal Behavior Model of Context Adapter of ABS

ABS has two types of devices, wheels and braking units. For simplicity, we only consider one wheel (D_{11}) and one brake unit (D_{21}) in the architecture. Environmental

contexts such as the road friction and the current wheel speed are sensed by the Sensing mechanism S. They are converted into symbolic form by the transformer and passed as atomic contexts to T_2. Applying the operations given by the context calculus toolkit, the atomic contexts are aggregated to a simple context c and passed as a parameter to T_3 [6]. The architecture of ABS is shown in Figure 2[a]. The formal behavior model of context adaptation is shown in Figure 2[b].

5 Conclusion

The main contribution of this paper is a formal approach for developing context-aware systems. The significant aspects of the architecture are: (1) *modularity:* each component is fully encapsulated, and their relative independence promotes easy refinements and maximal reuse; (2) *codesign:* the sensing components in CC require hardware implementation, and the transformer T requires a software implementation, and the tight coupling between them can be optimized through appropriate hardware/software partitioning; (3) *context toolkit:* component T_2 implements context operators and provides precedence rules, and their implementation is quite independent of the nature of contexts (supplied by CC) and adaptation (at T_3); and (4) *knowledge-base:* T_3 may be given the knowledge-base that is specific to an application, and hence the adaptation/controller can be reused for different applications by simply rebooting a new knowledge-base for every new application.

Sensory devices produce signals that are converted to symbolic forms. Necessarily, this process is error-prone, and consequently not all context information will be accurately quantified. In order to guarantee that the design satisfies a desired property, the design methodology should be verification-driven. This aspect of verification-driven design and implementation of context-aware systems is an integral part of our ongoing research.

References

1. F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli. *Hardware-Software Co-design of Embedded Systems – The POLIS approach.* Kluwer Academic Publishers, 1997.
2. A.K. Dey, D. Salber, G.D. Abowd. *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications*, anchor article of a special issue on Human Computer Interaction, Vol.16, (2001).
3. Cheverst, K., N. Davies, K. Mitchell and C. Efstratiou. *Using Context as a Crystal Ball: Rewards and Pitfalls.* Personal Technologies Journal, Vol. 3 No5, pp. 8-11, 2001.
4. F. Plasil, S. Visnovsky. *Behavior Protocols for Software Components.* IEEE Transactions on Software Engineering archive, 28(11), pp.1056 - 1076,2002.
5. K. Wan, V.S. Alagar, J. Paquet. *Real Time Reactive Programming Enriched with Context.* ICTAC2004, Guiyang, China, September 2004, Lecture Notes in Computer Science,3407, Springer-Verlag.
6. K. Wan. *Lucx: An Intensional Programming Language Enriched With Contexts*, Ph.d thesis(under preparation), Department of Computer Science, Concordia University, Montreal, Canada, 2005.