Journal of Mobile Multimedia, Vol. 4, No.3&4 (2008) 185-199
© Rinton Press

ON FOUNDATION OF ENGINEERING CONTEXT-SENSITIVE APPLICATIONS

LU YAN
School of Computer Science, University of Hertfordshire
College Lane, Hatfield, Hertfordshire AL10 9AB, UK
lu.yan@ieee.org

Received April 20, 2008
Revised July 10, 2008

The communication environment surrounding our daily experience is more and more characterized by mobile devices that can exchange multimedia information and provide access to various services of complex nature. The trend is now clear that future consumer computing experience will be based on multiple pervasive communication devices and services, where navigability, context-sensitivity, adaptability and ubiquity are key characteristics. Several issues have been studied, models and methodologies proposed, and tools and systems implemented. However, when we look at the foundation and what we are missing in research, some of the most relevant issues probably are a formal model of context-sensitive and a notion of synthesizing reliable complex systems from vast numbers of unreliable components. In this paper, we discuss a formal foundation and software engineering techniques for mobile context-aware and context-dependent service derivation and application development, emphasizing the relationships between context and system.

Key words: Context, specification, foundation

1 Introduction

With more than two billions terminals in commercial operation world-wide, wireless and mobile technologies have facilitated in the first wave of pervasive communication systems and applications. This trend shows several aspects consistent in the evolution of computing including the increasing miniaturization of the computing units and an increasing emphasis of the role of communication between them. Significant research work has been done over recent years on these systems at several levels, from the lowest physical level to the highest information processing level. However, the latter is less developed than the research at the lower levels. For instance, we think that the most relevant issue for the future perspective of true ubiquitous computing, *context-sensitive* has not received justified attention in the research community.

The term *context* has been extensively studied since the early 1990s; it was mainly associated with the concept of location, but it is much richer than this; some works have categorized context into different aspects, such as computational, user, physical, spatial and temporal context [1, 2, 3, 4, 5, 6]. However, a precise definition of context is yet still missing. In this paper, we interpret context as *a setting in which an event occurs*, and this definition, we believe, is suitable for the system software research.

As the previous work [7], we have described a formal approach to context-aware mobile computing: we offer the context-aware action systems framework, which provides a systematic method for managing and processing context information, defined on a subset of the classical action systems [8]. Based on the essential notions and properties of this formalism, we applied this formalism on deriving context-aware services for mobile applications [9], and implemented a smart context-aware kindergarten scenario where kids are supervised unobtrusively with wireless sensor networks [10].

Issues that have been considered are both theoretical and practical: modelling the system requirement rigorously with formal approaches, deriving the software architecture from formal models, stepwise refinement of the specification, code generation, and verification vs. simulation. While all these research issues have been individually studied in an extensive way, their interaction within the final implementation raises new challenges, which constitutes the focus of this paper.

The remainder of the paper is organized as follows: after a short introduction to related work in Section 2, a design framework for wireless sensor networks is presented in Section 3. In Section 4 we describe a formal model of context-awareness and context-dependency, and show the relationship between the model and software architectures. We discuss a case study on applying this model to software development process in Section 5, and then conclude the paper in Section 6.

2 Related Work

Several related works have noticed the importance of seeking a foundation of context-aware computing [22]. Roman *et al.* presented a formal treatment of context-awareness via extending the mobile UNITY with context handling part into context UNITY [23]. The context UNITY formalism is similar to our context-aware action systems formalism, but approaching from an agent-like view in modelling context-sensitive.

Henricksen *et al.* showed a conceptual framework and software infrastructure that together address known software engineering challenges in context-aware computing applications [24]. The context model is built on semantic level with the CML language [25], which can be categorized as an extension of the Object-Role Modeling in software engineering process.

UML approach to context models was presented by Hinze *et al.*, where UML diagrams are combined with discrete event systems to facilitate the development of mobile context-aware systems [26]. Due to the limitation of UML, which lacks a rigorous mathematic foundation, this approach can be deemed as a semi-formal one. The similar UML-like approach can be found [27], where a simulation-based paradigm was presented.

Service computing view on context-aware mobile applications was discussed by Aleksy *et al*. They proposed a set of architectural components and principles which allow context-sensitive, mobile business applications to be assembled in highly flexible and reuse-oriented way based on the principles of SOA [32]. This approach is mainly focused on software architectural considerations, and context is usually coded into XML segments [33].

Besides general aspect of context, fragment aspects of context, such as ontology [28], rational [29], middleware [30], trust [31] were also considered.

3 Wireless Sensor Networks

Wireless sensor networks provide perfect platforms to study context-aware systems upon. Wireless sensor networks have been an area of active research since the early 1990s [11], accelerated by the advancement of wireless networking and the development of sensors. Only recently, wireless sensor networks have moved from academic research concepts to commercially available products, increasing production quantities.

Although significant research work has been undertaken, most of the research is still very application specific, with security and environmental applications dominating [12]. However, it is likely that more generic and comprehensive approach is required, where true system level problems in wireless sensor networks and their applications can be studied. With such a perspective, we developed a design framework in Figure 1 for wireless sensor networks [13].

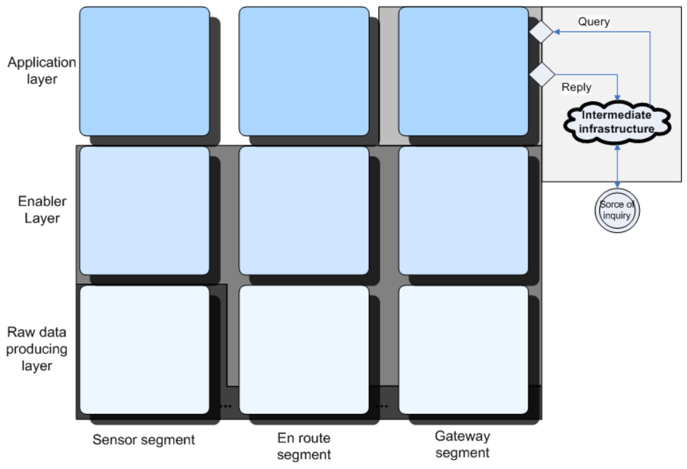


Figure 1 The sensornet system framework.

In this framework, we have distinguished between *context-provider* and *context-utilizer*; the former is the reactive part which detects the surroundings and acquires the context, and latter is the proactive part which interprets and responds to the context. The interaction between the context-provider and context-utilizer constitute a complete context-sensitive system.

Because the possibly bi-directional communication and the impossibility of restricting context to be a sensor reading, all nodes can potentially act as context-providers as well as context-utilizers. The roles are dependent on whether the data is propagating (an inquiry) or composing (a reply).

4 Formalizing Context-sensitive

We start by giving a brief overview of the action system formalism and then present how we model context-awareness and context-dependency within this formalism. By mapping the formal model back to the software architecture of wireless sensor networks, we show some realistic implementations of this model on system software research.

4.1 Action systems

The action systems formalism is based on Dijkstra's language of guarded commands [14]. This language includes assignment, sequential composition, conditional choice, and iteration. An action is a guarded command, i.e. a construct of the form $g \to S$, where *g* is a predicate, the *guard*, and *S* is a program statement, the *body*. An action is said to be enabled when its guard is evaluated to true. If an action does not change the program state it is called a *stuttering* action. The body *S* of an action is defined as follows:

$$S ::= \text{abort} \mid \text{skip} \mid x := e \mid \{x := x' \mid R\} \mid \text{if } g \text{ then } S_1 \text{ else } S_2 \text{ fi} \mid S_1; S_2$$

where *x* is a list of attributes; *e* is a corresponding list of expressions, *x'* is a list of variables standing for unknown values, and *R* is a relation specified in terms of *x* and *x'*. Intuitively, *skip* is a stuttering action, $x := e$ is a multiple assignment, $\text{if } g \text{ then } S_1 \text{ else } S_2 \text{ fi}$ is the conditional composition of two statements, and $S_1; S_2$ is the sequential composition of two statements. The action *abort* always fails and is used to model disallowed behaviours. Given a relation $R(x, x')$ and a list of attributes *x*, we denote by $\{x := x' \mid R\}$ the *non-deterministic* assignment of some value $x' \in R.x$ to *x* (the effect is the same as abort, if $R.x = \varnothing$).

The semantics of the actions language has been defined in terms of weakest preconditions in a standard way [14]. Thus, for any predicate *p*, we define:

$$\begin{aligned}
&wp(\text{abort}, p) = \text{false} \\
&wp(\text{skip}, p) = p \\
&wp(x := e, p) = p[x := e] \\
&wp(\{x := x' \mid R\}, p) = \forall x' \in R.x \cdot p[x := x'] \\
&wp(S_1; S_2, p) = wp(S_1, wp(S_2, p)) \\
&wp(\text{if } g \text{ then } S_1 \text{ else } S_2 \text{ fi}, p) = \text{if } g \text{ then } wp(S_1, p) \text{ else } wp(S_2, p) \text{ fi}
\end{aligned}$$

where $p[x := e]$ stands for the result of substituting all the free occurrences of the attributes *x* in the predicate *p*.

Thus, an action system is a construct of the form:

$$\begin{aligned}
\text{A } = |[\ &\text{import } i; \\
&\text{export } e := e_0; \\
&\text{var} \quad v := v_0; \\
&\text{do} \quad A_1 [] A_2 [] ... [] A_n \quad \text{od} \\
]|\ &
\end{aligned}$$

where the *import* section describes the imported variables *i* that are not declared, but used in *A*. The variables *i* are declared in other action systems, and thus they model the communication between action systems. The *export* section describes the exported variables *e* declared by *A*. They can be used within *A* and also within other action systems that import them. Initially, they get the values e_0. If the initialization is missing, arbitrary values from the type sets of *e* are assigned as initial values. The *var* section describes the local variables of action system *A*. They can be used only within *A*. Initially they are assigned values v_0, or, if the initialization is missing, some arbitrary values from their type set. Technically, all the used variables in import and export sections are global variables, and only variables defined in *var* section are local ones. The $do \ldots od$ section describes the computation involved in *A*. Within the loop, $A_1 \ldots A_n$ are actions of *A*.

The behaviour of the action system *A* is as follows: the execution starts by initialization of all variables, and then repeatedly, an enabled action from $A_1 \ldots A_n$ is nondeterministically selected and executed. If two actions are independent, i.e., they do not have any variables in common, they can be executed in parallel [15]. Their parallel execution is then equivalent to executing the actions one after the other, in either order.

An action system is not usually regarded in isolation, but as a part of a more complex system. A large action system can be constructed from smaller ones using composition. Consider two action systems *A* and *B* below:

```
A = |[ import  $i$;                          B = |[ import  $j$;
          export  $e := e_0$;                         export  $f := f_0$;
          var     $v := v_0$;                         var     $w := w_0$;
          do      $A_1[]A_2[]...[]A_n$   od           do      $B_1[]B_2[]...[]B_m$   od
      ]|                                          ]|
```

where $v \cap w = \varnothing$. We define the parallel composition of *A* and *B*, written $A \square B$, to be the following action system *C*:

```
A || B = |[ import  $k$;
               export  $h := h_0$;
               var     $u := u_0$;
               do      $A_1[]A_2[]...[]A_n[]B_1[]B_2[]...[]B_m$
               od
           ]|
```

where $k = (i \cup j) \setminus h, h = e \cup f$ and $u = v \cup w$. The initial values of the variables and the actions in $A \square B$ consist of the initial variables and actions of the original action systems. The binary parallel composition operator $\square$ is associative and commutative and thus extends naturally to the parallel composition of a finite set of action systems. The behaviour of a parallel composition of action systems is dependent on how the individual action systems interact with each other. The parallel composition operator can also be used in a reverse direction to decompose one action system into a number of those.

The underlying basis for stepwise development of action systems is the refinement calculus [16]. In the refinement calculus, program statements are identified with their weakest precondition predicate transformers. Our treatment of the action system refinement is based on the theory presented there.

4.2 Context model

With this formalism, we start modelling the context-sensitive systems by specifying the context-provider and context-utilizer roles as described in section 3. First we consider a context-dependent system, modelled by the action system *CD*:

```
CD = |[ import ...
        export ...
        var    ...
        do     $g \rightarrow S [] \neg g \rightarrow T [] \beta$
        od
     ]|
```

where g is the context guard and S is a statement dependent on the context g: $g \rightarrow S$ models the system behaviour with provided context, and $\neg g \rightarrow T$ models the system behaviour without provided context; β stands for the other actions of *CD*. The context guard g is a predicate on the local and context variable(s) x. The context variables constitute in a subset of the import and export variables. The value of g is maintained by some other action system *CH*, called context-handler. Consequently, the context variable x is an imported variable to *CD* and an exported variable in *CH*.

Hence, we need to introduce the context handler, maintaining g in Figure 2. The context handler is a context-provider and can potentially be a context-utilizer, depending on the service. If it were not a context-provider, there would not be anything requiring handling of the context. Thus, the handler is an independent, but necessary part of the system. The context handler is modelled by action system *CH*:

```
CH = |[ import ...
        export ...
        var    ...
        do     $b \rightarrow x := x' \mid x' \in \{g, \neg g\}$
               $\neg b \rightarrow \mathrm{V}$
        od
     ]|
```

where b is a predicate; and $b \rightarrow x := x' \mid x' \in \{g, \neg g\}$ nondeterministically updates the global context variable x. The nondeterministic update is later refined to a realistic intelligent algorithms. Hence, it models the context provided to *CD*.

Now, the parallel composition of action systems *CD* and *CH*, i.e. $CD \| CH$ is a complete context-aware model, and it models interactions between the context-provider and context-utilizer. The implication of this model in the software architecture design can be explained in Figure 1, where the grey-shaded areas illustrate the main responsibility for the nodes belonging to them. The dark grey area constitutes the sensing nodes, the grey the en route nodes and the light-grey the gateway node. Moreover, it should be interpreted so that each item is considered belonging primarily to *layer* and secondarily to *segment*.

One merit of our model is that we intentionally separate the origin of the context from the whole context-aware system. This separation has one important consequence: the context is the result *after* processing within the context-provider; i.e. the action system *CH* differentiates between *data* and *relevant data* and context is therefore always *refined* raw data.

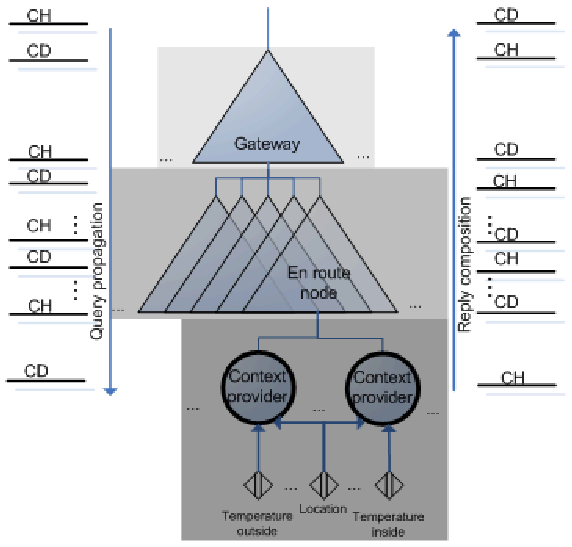


Figure 2 Data propagation and composition.

As the realistic implication, the above idea contributes to a further classification of sensor nodes in wireless sensor networks as Figure 2. In this service oriented view, all sensors do not necessarily provide data needed for replying a query, nor does all function as en route nodes. Consequently, if possible the en route nodes decide based on the context whether their underlying sensing nodes can provide relevant information and thereby, forward or not. The en route nodes can also, if implemented, compose data for providing relevance and because energy efficiency reasons. In the end, the context information is fused in the gateway node from the en route nodes to provide relevant and accurate answers for the propagated query.

4.3 Context refinement

In this section we discuss how the refinement principles can be used together with a parallel composition rule in our model. We show how to refine an abstract specification towards a detailed one, as well as the realistic implications of these refinements in system software design.

First, we consider one simple refinement scenario:

$$CD||CH\ \hat{o}_{R}\ CD'||CH$$

where *CD’* is the refinement result of *CD*. The realistic implication of this scenario is upgrading the sensor application without touching the sensing part. This kind of refinement could mean: suppose we have a supervisory software *CD* running on top of the wireless sensor network infrastructure, now we update the existing software to a later version with more features *CD’*.

Since this category of refinement only concerns individual action systems, there should not be any change in the aggregated behaviour of the whole system. Thus, we give the refinement rules as follows. Consider two actions systems *CD* and *CD'*:

```
CD = |[ import i;                      CD' = |[ import i;
        export  e := e_0;                      export  e := e_0;
        var     a := a_0;                      var     a' := a_0';
        do      A_1[]A_2[]...[]A_n              do      A_1'[]A_2'[]...[]A_n'[]X_1[]X_2[]...[]X_m
        od                                     od
     ]|                                     ]|
```

where the local variables *a* in *CD* are replaced with new local variables *a'* in *CD'*. The actions A_i in *CD* are replaced with A_i' in *CD'*, and auxiliary actions X_j are added into *CD'*.

R is mapping a relation between the new local variable *a'* and the old variable *a*. Consequently, we can say that the action system *CD* is refined by the action system *CD'*, if there exists an abstraction relation $R(a, a')$ such that the following conditions hold:

1. Initialization: $R(a_0, a_0')$
2. Main actions: A_i ô $_R$ A_i', for $i = 1, \ldots, n$
3. Auxiliary actions: skip ô $_R$ X_j, for $j = 1, \ldots, m$
4. Continuation condition: $R \wedge gCD \Rightarrow gCD'$
5. Internal convergence: $R \Rightarrow wp(\text{do } X_1[]X_2[]\ldots[]X_m \text{ od, true})$

where the first condition says that the abstraction is established by the initializations. The second condition requires that each action A_i is refined by the corresponding action A_i' using $R(a, a')$. The third condition states that the auxiliary actions X_j behave like *skip* with respect to the global variables $i \cup e$ while preserving $R(a, a')$. The fourth condition requires that an action in *CD'* is enabled whenever an action in *CD* is enabled and $R(a, a')$ holds. The last condition stipulates that the execution of the auxiliary actions taken separately cannot continue forever whenever $R(a, a')$ holds.

The second simple refinement scenario considers the context-provider itself:

$$\text{CD}||\text{CH} \text{ ô }_{\text{R}} \text{ CD}||\text{CH'}$$

where *CH'* is the refinement result of *CH*. The realistic implication of this scenario is improving the context processing unit without touching the upper layer sensor applications. This kind of refinement could be exemplified by for example: suppose we have a supervisory software running on top of the wireless sensor network infrastructure, now we improve the wireless sensor network infrastructure to provide more relevant and precise context information.

This category of refinement also concerns individual action systems and there is no change in the aggregated behaviour of the whole system. Therefore, we can use the refinement rules described above as well.

Here we consider one common refinement example on refining the context handling algorithm. In our initial model, the context handling algorithm is rudimentally expressed as $b \to x := x' \mid x' \in \{g, \neg g\}$. Later we further refined this algorithm into a realistic intelligent algorithm such as [21]. Usually this kind of refinement only refines local actions.

The last refinement scenario is a complex one, where the context-provider and context-utilizer co-refines together:

$$\mathrm{CD}\|\mathrm{CH}\ \hat{o}_{\mathrm{R}}\ \mathrm{CD'}\|\mathrm{CH'}$$

where *CD’* is the refinement result of *CD*, and *CH’* is the refinement result of *CH*. The realistic implication of this scenario is refining the sensing part and application part simultaneously, interacting with each other. This kind of refinement could be exemplified as: suppose we have a supervisory software running on top of the wireless sensor network infrastructure, now we redesign the whole system, touching both the existing upper layer software and lower layer wireless sensor network infrastructure.

Obviously, this category of refinement is complex, because it concerns not only the individual behaviour of each action system but also the aggregated behaviour of the whole system. We can use the compositional refinement rules [19], to refine this kind of scenario. However, due to the complexity of this kind of refinement, we do not list down the complete refinement rules (more on compositional refinement can be found [19]). We instead present an intuitive illustration for understanding this kind of refinement in Figure 3, where an arrow represents a refinement step and a line represents an abstraction relation.

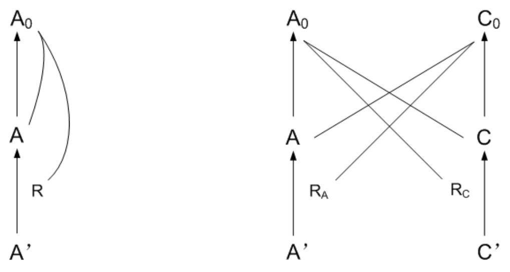


Figure 3 Individual refinement process vs. compositional refinement process.

We further show an example on introducing new context to the whole system via compositional refinement: suppose we have the original system modelled as $CD\square CH$, where *CD* and *CH* are defined in the previous section. In this original setting, we have only *g* as our context. Now we would like to extend the context part by introducing a new context to the whole system. In reality, this scenario implies the case as utilizing additional data in the system which usually compels to redesigning of the system.

Using the compositional refinement, we can approach the problem as follows. First we consider the *CD'*, which is the refinement result of *CD*. Let this new extra context be *d*. Assume *d* is a subset of $\neg g$, i.e. $d \subseteq \neg g$. Applying the individual refinement rules, we can refine the original action $\neg g \rightarrow T$ in into two new actions:

$$d \rightarrow R \; [] \; (\neg g \setminus d) \rightarrow T' \wedge \neg b \setminus d \rightarrow V'$$

where *R* and *T'* are refined statements satisfying:

$$T \; \hat{o}_R \; R \text{ and } T \; \hat{o}_R \; T'$$

Then the new context is evaluated in *CH'*, which is the refinement result of *CH*. Now $CD' \square CH'$ is the refinement result of $CD \square CH$.

Actually this is an effective way of stepwise adding new features to the system, when simultaneously touching both the sensing part and the application part is inevitable. If we limit the context to system failure, this approach is similar to the work done [20] in the fault tolerant direction to provide means to handle certain faults.

5 Case Study

We have implemented a smart kindergarten (nursery school) scenario as a case study for the proposed context-role categorization approach. The core concept of this application is illustrated in Figure 4, as a smart surveillance system for a kindergarten. The system consists of stationary base stations, mobile sensor nodes which are attached to the children, and the supervisory application. The children are allowed to move freely in a predefined area (playground), and the supervisor is able to get the location information of all nodes (visually). When a child leaves the predefined area, the alertness level of the system increases, and the supervisor is informed. Higher alertness level implies intensified communication. Moreover, intensified location reporting, by the distinct node, is conducted when vibration is detected (the child can be assumed to move).

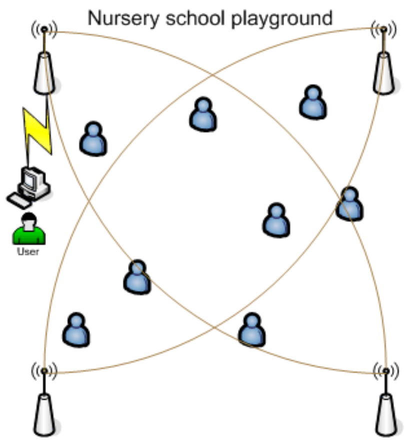


Figure 4 Smart kindergarten case study.

This scenario is a typical context-sensitive example consisting of a context-provider and a context-utilizer. The system behaviour, the context-utilizer, is critically dependent on different contexts provided by the context-provider, i.e. for supervision and localization. Moreover, in this particular example the base stations function as context-providers, the beacon, as well as context-utilizers, calculating the position and raising the alertness level.

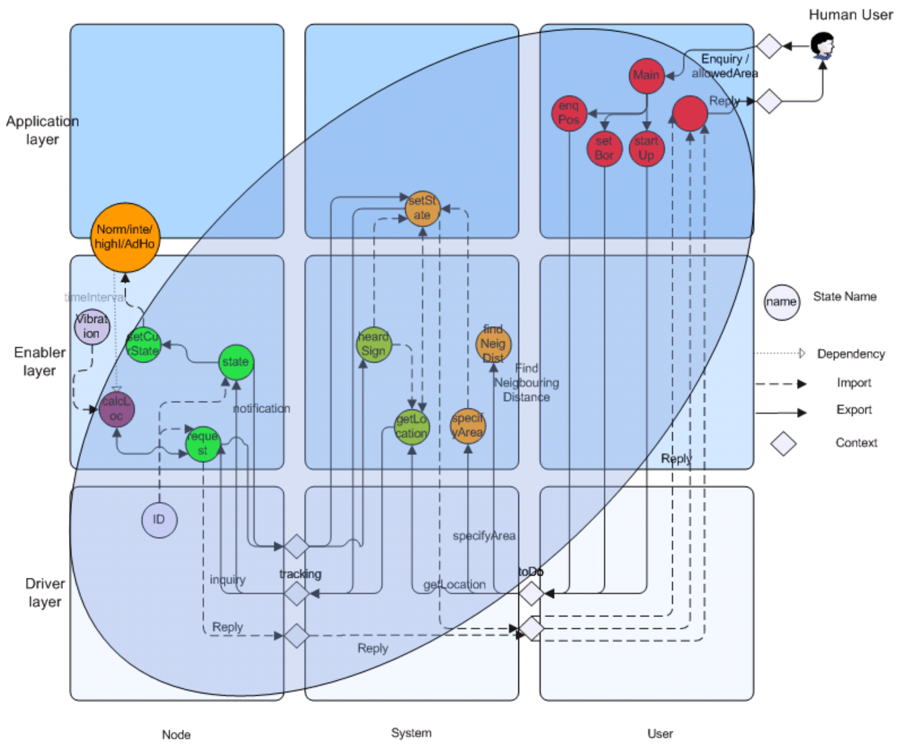


Figure 5 Refined software architecture of the system.

Using the proposed context model in Section 4 and formalism [7], we implemented a variant of ROCRSSI [21] for the localizing service. Here we show a final model of the system in Figure 5, which is the stepwise developed result of Figure 1. This model follows the Lyra process [17] and works as the basis of the kindergarten application. The conclusion drawn was that the system is hierarchically pushing/pulling context information.

We then take a fraction of the model and show the specification. For instance, some specification of the tracking part:

```
AgetLocation |[
 context  getPos;
 import   recordHeardSignals, timeInterval;
 export   square, inquiry, checkState, tracking;
 var      nowTime;
 do
       nowTime - recordHeardSignals.time > timeInterval
       -> tracking := inquiry.calcLocation
    [] square = recordHeardSignals.intersection
       -> checkState := setState
 od ]|
```

```
AcalcLocation |[
 import currentState, timeInterval;
 export currentLocation, recordHeardSignals;
 var     nowTime, thresholdTime;
 do
      nowTime < thresholdTime
       -> recordHeardSignals;
    [] currentLocation := recordHeardSignals;
 od ]|
```

where the context can be viewed as the guard, in this case the variables *getPos*, *checkState* and *tracking*. The imported variables are the services for which this fraction is a context-utilizer for. The exported variables constitute in the context-provider role of the fraction and hence, *checkState* and *tracking* can appear in the guards of systems importing them.

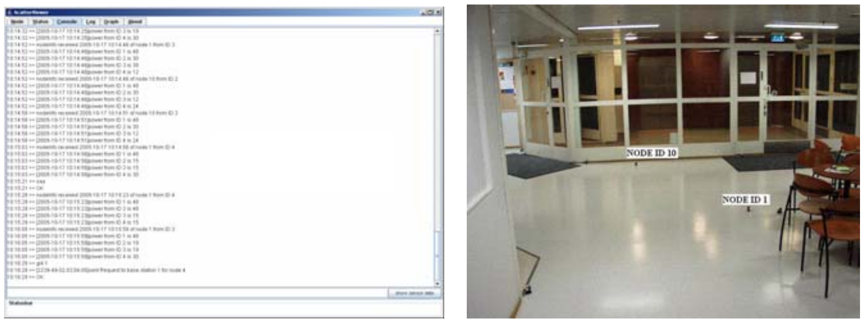
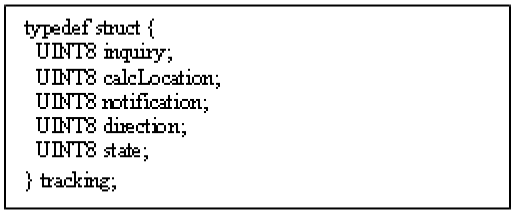
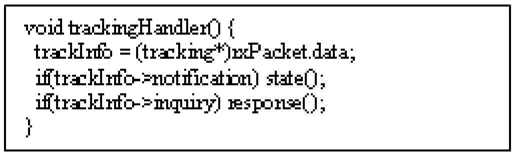


Figure 6 Implemented prototype of the system.

We don't elaborate the full system specification here. A complete specification of the kindergarten application and some of its implementation details are available [10]. Most refinement steps were subsequently conducted with Event-B and the Rodin platform [18]. A segment of final code generated from the above specification is listed below, and some snapshots of the running prototype system are shown in Figure 6.



```
typedef struct {
  UINT8 inquiry;
  UINT8 calcLocation;
  UINT8 notification;
  UINT8 direction;
  UINT8 state;
} tracking;
```



```
void trackingHandler() {
  trackInfo = (tracking*)rxPacket.data;
  if(trackInfo->notification) state();
  if(trackInfo->inquiry) response();
}
```

6 Concluding Remarks

In this paper, we discussed a formal foundation and software engineering techniques for context-sensitive service derivation and application development, emphasizing the relationships between context and system. As stated in the abstract, there is a definite lack of formal support for modelling realistic context-awareness in mobile computing applications. The context-sensitive action systems presented in this paper provides mechanisms for modelling complex and interwoven sets of context-information by extending traditional software engineering models with new constructs and capabilities. We believe this formal model is a giant leap forward in the direction of making formal methods applicable in the area of mobile computing.

References

1. A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. *Proc. CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, The Hague, The Netherlands, 2000.
2. Mika Raento, Antti Oulasvirta, Renaud Petit, Hannu Toivonen. ContextPhone - A prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing*, 4 (2): 51-59, 2005.
3. Special issue on Context-Aware Computing. *IEEE Pervasive Computing*, 2002.
4. H. Chen, T. Finin, and A. Joshi. An ontology for contextaware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197-207, 2004.
5. A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6): 893-901, 1999.
6. G. Chen and D. Kotz. A survey of context-aware mobile computing. *Technical Report TR2000-381*, Dartmouth College, Department of Computer Science, 2000.
7. L. Yan and K. Sere. A Formalism for Context-Aware Mobile Computing. *Proc. Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, 2004.
8. R.J.Back and K. Sere. From Action Systems to Modular Systems. *Software - Concepts and Tools*. (1996) 17: 26-39.

9. Mats Neovius and Christoffer Beck. From requirements via context-aware formalisation to implementation. *Proc. the 17th Nordic Workshop on Programming Theory*, Copenhagen, Denmark, 2005.
10. Christoffer Beck. An application and evaluation of Sensor Networks. *Master thesis*, Åbo Akademi, Finland, 2005.
11. S. Sitharama Iyengar and Richard R. Brooks. *Distributed Sensor Networks*. Chapman & Hall/CRC, 2004.
12. E. Yoneki and J. Bacon. A survey of Wireless Sensor Network technologies: research trends and middleware's role. *Technical Report UCAM-CL-TR-646*, University of Cambridge.
13. M. Neovius and L. Yan. A Design Framework for Wireless Sensor Networks. *Proc. of IFIP 1st International Conference on Ad-Hoc Networking*, Santiago De Chile, Chile. 2006.
14. E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.
15. R.J. Back and K. Sere. Stepwise Refinement of Action Systems. *Structured Programming*, 12(1): 17-30, 1991.
16. R.-J. Back, J. Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science, Springer-Verlag, 1998.
17. J. Honkola, S. Leppanen, P. Rinne-Rahkola, M. Soderlund, M. Turunen, K. Varpaaniemi. A Case Study: Applying Lyra in Modeling S60 Camera Functionality. *Proc. 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS)*, Tucson, USA, 2007.
18. J.-R. Abrial. A System Development Process with Event-B and the Rodin Platform. *Proc. 9th International Conference on Formal Engineering Methods (ICFEM)*, FL, USA, 2007.
19. R. J. Back and J. Wright. Compositional action system refinement. *Formal Aspect of Computing*,15(2-3): 103-117, 2003.
20. J. Xu, B. Randell, A. Romanovsky, R.J. Stroud, A.F. Zorzo, E. Canver, F. Henke. Rigorous development of an embedded fault-tolerant system based oncoordinated atomic actions. *IEEE Transactions on Computers*, vol. 51, issue 2, pp. 164-179, 2002.
21. C. Liu, K. Wu, and T. He. Sensor localization with Ring Overlapping based on Comparison of Received Signal Strength Indicator. *Proc. IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, Oct. 2004.
22. P. Dourish. *Where The Action Is: The Foundations of Embodied Interaction*. MIT Press, 2001.
23. G.-C. Roman, C. Julien, and J. Payton. A Formal Treatment of Context-Awareness. *Proc. 7th International Conference Fundamental Approaches to Software Engineering (FASE)*, Lecture Notes in Computer Science 2984, Springer 2004.
24. K. Henricksen and J. Indulska. A Software Engineering Framework for Context-Aware Pervasive Computing. *Proc. 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2004.
25. K. Henricksen. A framework for context-aware pervasive computing applications. *PhD thesis*, University of Queensland, Sept. 2003.
26. A. Hinze, P. Malik, and R. Malik. Interaction design for a mobile context-aware system using discrete event modelling. *Proc. Twenty-nineth Australian Computer Science Conference (ACSC)*, Hobart, Australia, 2006.
27. P. Guo and R. Heckel. Modeling and Simulation of Context-Aware Mobile Systems. *Proc. 19th IEEE International Conference on Automated Software Engineering (ASE)*, 2004.
28. A. Pappas, Stephen Hailes, and Raffaele Giaffreda. A design model for context-aware services based on primitive contexts. *Proc. UbiComp*, 2004.
29. Y. Roussos and Y. Stavrakas. Towards a Context-Aware Relational Model. *Technical Report TR-2005-1*, National Technical University of Athens, 2005.

30. E. Katsiri. Middleware support for context-awareness in distributed sensor-driven systems. *Ph.D. Thesis*, University of Cambridge, Feb. 2005.
31. M. Carbone, M. Nielsen, and V. Sassone. A Formal Model for Trust in Dynamic Networks. *BRICS Report RS-03-4*, 2003.
32. T. Butter, M. Aleksy, P. Bostan, M. Schader. Context-aware User Interface Framework for Mobile Applications. *Proc. 27th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Toronto, Canada, June 2007.
33. M. Aleksy, C. Atkinson, P. Bostan, T. Butter, M. Schader. Interaction Styles for Service Discovery in Mobile Business Applications. *Proc. 17th International Conference on Database and Expert Systems Applications (DEXA)*, Krakow, Poland, Sep. 2006.