Formal Component Models for Context-Awareness

Deploy FP7
Mats Neovius and Kaisa Sere

Outline of this talk

- Motivation
- Informal definitions
- Problem
  - Modelling context-aware system
  - Treating the contexts as components
  - Proposed framework for solving the problem
- Action Systems at a glimpse
- Context in action systems
- Example: fictionary driverless car
- Conclusion & further work

Motivation

- In order to provide rigour on SW, formal methods suffice
- However, in order to provide rigour on a system, environment must be considered
  - Why is this of interest
    - for example, if the car breaks down the problem is not malfunctioned SW or erroneous treating of sensor reading, the main problem is - a broken car.
  - In a system, the environment is a decisive factor, we need to introduce it "formally"
    - Consider irrationality and force the designer to treat and include which and how for instance sensor readings are acquired and calculated with

Informal definitions

- Context (when talking about context-aware)
  - *Context is any information that can be used to characterise the situation of entities at some given moment. An entity is a person, place, object, virtual object or state that is considered relevant to the interaction between a user and an application, including the user and the application themselves.* [Dey, others]
- Component
  - *"A software component is a unit of composition with contractually specified interfaces and explicit context dependences only. A software component can be deployed independently and is subject to composition by third parties"* [Szyperski]

Problem – modelling a context-aware system

- How is an elementary context utilised in the context-aware SW

  (Elementary context denotes for instance a sensor reading)

  – Need to specify the ”utiliser-interface”
- How is context processed to provide increased information
  – Composition of elementary contexts
  – Need to specify the interface between the processing entities

Problem – Treating contexts as components

- Avoid a monolithic constructions for the sake of independent deployment and reuse
- Information builds bottom-up
  - – Elementary contexts form the basis for context-aware SW
    - Composition of contexts

Accelerate ok (given speeding is prohibited)?

acquire : velocity $\wedge$ speed_limit

improve : velocity > speed_limit $\rightarrow$ speeding := true

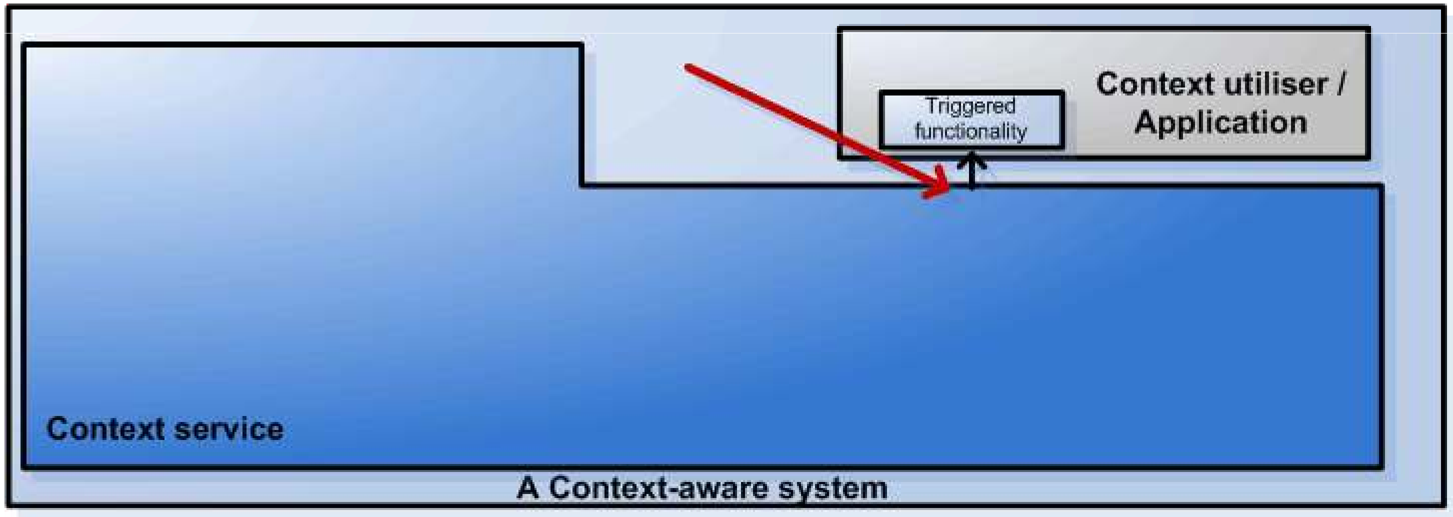
provide : speeding

Problem – solution

- Treat context as a service to its utiliser
  - Utiliser able to acquire information whenever desired (triggered by something, maybe a context)
    - Such as when resolving whether or not to accelerate (previous slide)
  - We consider each so called elementary context as a component in its own right

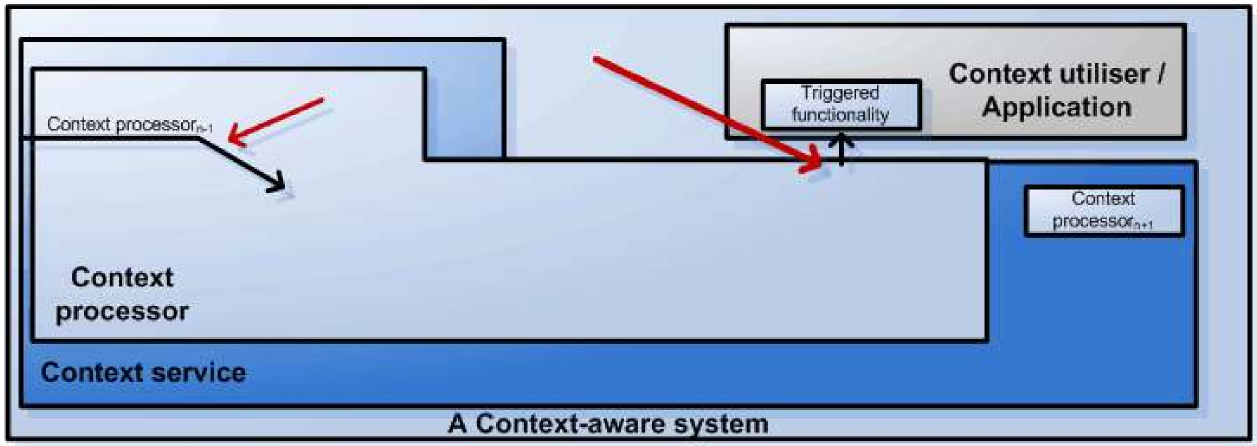
Problem – solution view

- Context utiliser / application
  - Relies on some context-service(s)
    - Proactive (or reactive)



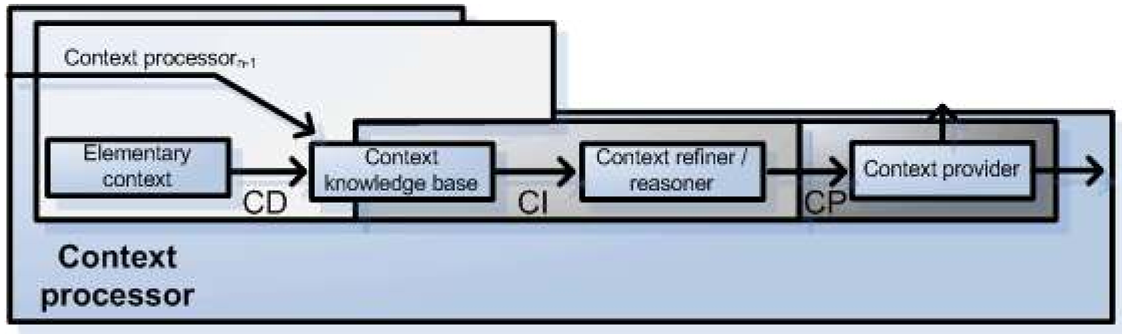
Problem – solution view

- Context utiliser / application
  - Relies on some context-services
    - Proactive or reactive
- Context service
  - Relies on a set of context-processors
    - Provides an answer to some question

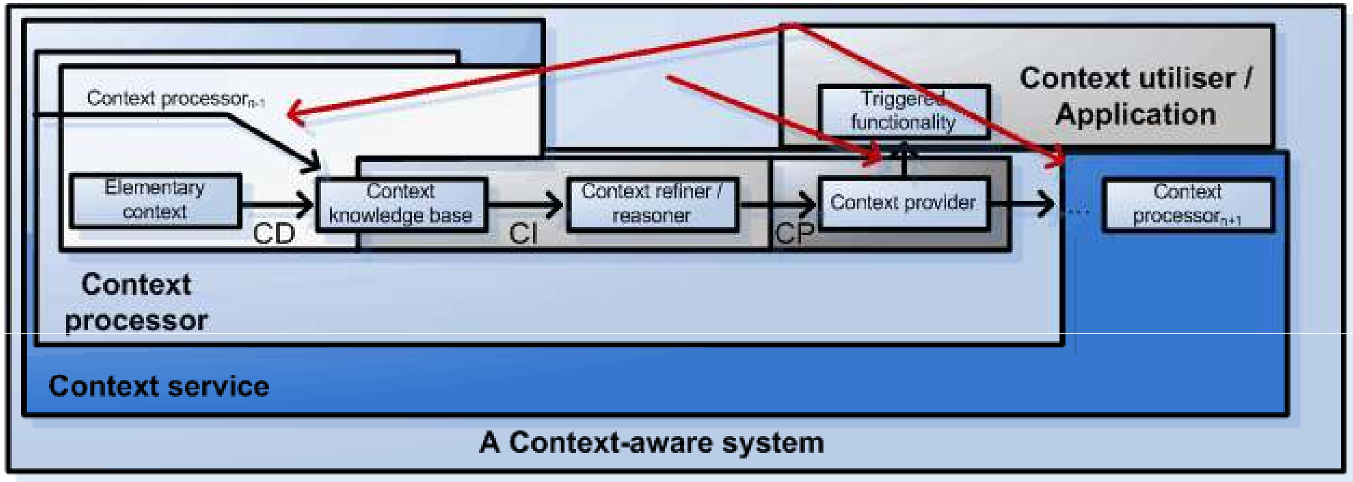


Problem – solution view

- Context processor
  - A triad, CD || CI || CP
    - CD – acquire metrics
    - CI – apply algorithm
    - CP – provide a metric whose information have been increased



Problem – solution view, all together



Inspired by:

Henricksen and Indulska, ”Modelling and Using Imperfect Context Information”, CoMoRea 2004.

Shehzad, Ngo, Pham, Lee ”Formal modeling in Context aware systems” CAMUS framework and Contextual informtaion hierarchy

Action systems at a glimpse

- State based formalism, based on Dijkstra's guarded command language
  - A predicate (guard) enables a statement (body) $g \rightarrow S$
    - Atomicity
    - Non-determinism
    - Parallelism

$\mathcal{A}$ = | [
import *imp_list*;
export *exp_list* := e_0;
var *var_list* := v_0;
proc *list of pr.name* (*par.name.list*) = <{*impl*}>;
do $g_1 \rightarrow S_1$ [] … [] $g_n \rightarrow S_n$ od
] |

Specialisations of action systems

- Context is non-changeable
  - New declarations of
    - read_only : published by some other AS
    - publish : read_only in other AS
    - context : elementary context(s)

$\mathcal{B}$ = | [
context *c*
import *j*;
export $f := f_0$;
read_only *r;*
publish $p := p_0$;
var $w := w_0$;
proc ;
do $g_1 \rightarrow S_1$ [] … [] $g_n \rightarrow S_n$ od
] |

Specialisations to action systems cont

- Introducing context in the utiliser - @
  - Def.@: $g_1 \rightarrow S_1$ @T= $g_1 \wedge g_T \rightarrow S_1; S_T$ where g_T and S_T is defined in action system $\mathcal{C}_{interface}$
  - Action system $\mathcal{C}_{interface}$ function as the interface for the context service (relying on context processors)

$\mathcal{B}_{utiliser}$ = |[
import *j*;
export $f := f_0$;
var $w := w_0$;
proc ;
do $g_1 \rightarrow S_1$@T [] … [] $g_n \rightarrow S_n$
… od
]| @$\mathcal{C}_{interface}$

Specialisations to action systems cont

- Introducing context in the utiliser - @
  - Def.@:$g_1 \rightarrow S_1$ @T= $g_1 \wedge g_T \rightarrow S_1; S_T$ where g_T and S_T is defined in action system $\mathcal{C}_{interface}$

$\mathcal{C}_{interface}$ = |[
read_only r_r;
import i_r;
export e_r;
var v_r;
proc *Proc_name* (*par.name.list*)
= {<impl>};
do …
T: $g_T \rightarrow S_T$
od |]

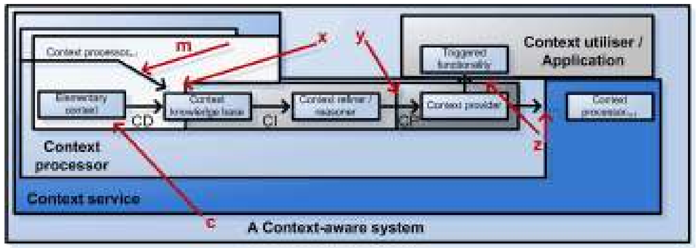
$\mathcal{B}_{utiliser}$ = |[
import j;
export $f := f_0$;
var $w := w_0$;
proc ;
do $g_1 \rightarrow S_1$@T [] … [] $g_n \rightarrow S_n$
… od
]| @$\mathcal{C}_{interface}$

A context processor – making information

$\mathcal{CD}$ = | [context *c*;
import ϕ;
read_only *m* ;
export *t*;
publish *x*;
var *a*;
proc ;
do
$g \rightarrow S\ (x := E)$
[] $\neg g \rightarrow T$
[] β
od
] |

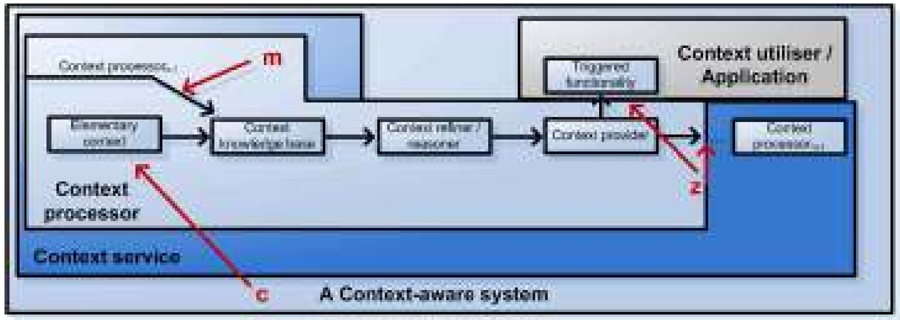
$\mathcal{CI}$ = | [context ε;
import λ;
read_only *x*;
export *q*;
publish *y*;
var *b*;
proc *nonce*(x)=f_2(x):x';
do
$\omega \rightarrow y := f_1(nonce(x))\lambda$
[] $\neg\omega \rightarrow V$
[] γ
od
] |

$\mathcal{CP}$ = | [context ε;
import μ;
read_only *y*;
export *s*;
publish *z*;
var *h*;
proc ;
do
$\psi \rightarrow z := f(y)$
[] $\neg\psi \rightarrow U$
[] δ
od
] |



A context processor- composed

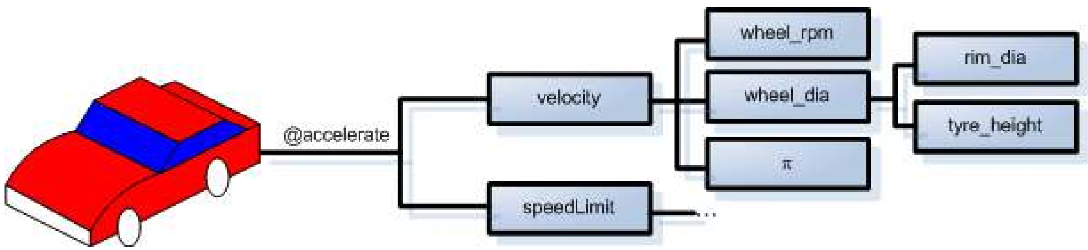
- Composition possible assuming
  - The local variables are disjoint



$\mathcal{C}_{\mathcal{CD}||\mathcal{CI}||\mathcal{CP}}$ = |[
context *c* ∪ ε;
import (ϕ ∪ λ ∪ μ \ e);
read_only *m*;
export *e* := t ∪ q ∪ s;
publish *z*;
var *v* := a ∪ b ∪ h ∪ x ∪ y;
proc *nonce*(x)=<{impl}>;
…
do
β [] γ [] δ [] T [] V [] U [] B_1 [] B_2 [] … [] B_n;
od
]|

Example: non-speeding car

- A smart car that does not allow acceleration if speeding
  - Demands comparison of velocity vs. speedLimit
    - Seemingly easy but what is needed to calculate velocity (assuming speed limit is clear cut)
      - velocity = f(wheel_height, wheel_rpm, π)



Example: the context utiliser

- $\mathcal{D}$ is the utiliser $\mathcal{Q}$ is the context-interface
  - $g_2 \rightarrow S_2$ calls on action labled Acc in $\mathcal{Q}$
  - Action Acc calls on resolving procedure speeding
  - According to definition of @, $g_2 \rightarrow S_2$ = $g_2 \wedge g_{Acc} \rightarrow S_2; S_{Acc}$

$\mathcal{D}$ = |[import *j*;
export $f := f_0$;
read_only ;
var $w := w_0$;
proc ;
do
A_1 [] $g_2 \rightarrow S_2$@*Acc* [] … [] A_n
od]| @$\mathcal{Q}$

$\mathcal{Q}$ = |[
proc *Speeding* (vel:Int, spLi:Int) =
    vel ≤ spLi → return := true
read_only velocity, speedLimit;
var vel ≔ velocity, spLi ≔ speedLimit;
do …
Acc : *Speeding*(vel, spLi) = true → skip
od]|

Example: making velocity

- Context processor $\mathcal{E}$ publishes *velocity* and calculates it given that
  - wheel_rpm, rim_dia and tyre_height is provided by read_only or by sensed elementary contexts (π is a constant)
    - wheel_rpm, rim_dia, tyre_height $\in \{k \cup c\}$

$\mathcal{E}_{\mathcal{CD}||\mathcal{CI}||\mathcal{CP}}$ = |[…
context c;
publish velocity;
read_only k;
var v;
…
Proc *CalcVel*(rpm:Int, rim:Int, tyre:Int)
= v := rpm* π * (rim + tyre)
do
CalcVel(wheel_rpm, rim_dia,
tyre_height) → velocity ≔ v;
…
od
]|

Åbo Akademi

Conclusions

- Conclusions
  - Each elementary context is treated as a component
    - Specified interface
    - Independently deployed
    - Composed by third parties
  - We have given a formal syntax for this and showed the relevance and implication of this
    - With an example
    - Motivated with words – remember the broken car

Future work

- Future work in this case
  - How does this relate to refinement
- Future work on a more general level
  - Can we treat composed components in separation and what can we tell about them
  - Can we formally model only a subset of a system and what can we say about it in that case