Situation Specification and Realisation in Rule-Based Context-Aware Applications

TTT presentation 8 May 2008

Luis Ferreira Pires

Background
- Collaboration on architectural support for context-aware applications
- PhD thesis of Patricia Dockhorn Costa
- 15+ scientific papers
- Involvement of several people
- Patricia Dockhorn Costa
- João Paulo Andrade Almeida
- Luís Ferreira Pires
- Marten van Sinderen
- Embedded in the AWARENESS and A-MUSE projects

Main contributions
- Architectural patterns for context-aware services infrastructures
- Context modelling based on an ontological foundation
- Situation modelling
  - Situation realization using a rule-based approach
  - Domain-specific language for specifying context-aware reactivity

Contents
- Background
- Contributions
- Motivation
- Context
- Context model
- Context situations
- Situation realisation
- Conclusions

Motivation
- Context-aware applications are capable of reacting intelligently upon changes in their users’ context
- Too expensive to build each application (for each user) from scratch
  → need for a ‘platform’ with reusable building blocks and facilities to accelerate application development and deployment
- We applied the Event-Control-Action (ECA) architectural pattern to design a context-aware services platform
  - Lower-level patterns: Context Sources and Managers Hierarchy pattern, Actions pattern

Motivation (cont’d)
- ECA pattern has been filled in with components to handle context changes (Events), components that perform the application logic (Control), and components to handle service calls (Actions)
- We developed a language to describe the application logic (ECA-DL) meant for application designers
- We investigated the implementation of the controller component using rule engines (particularly Jess)
Context-aware applications

User interaction

Context information

Context-aware application

Context
- ‘the set of possibly interrelated conditions in which an entity exists’

CA application development

In order to support context-aware applications one needs amongst others (meta)models that represent
- context types and their relationships
- the ‘imperfection’ of context information (Quality of context)

Presentation concentrates on first topic
- conceptual model for context

Goal: common understanding (unambiguous representation)

Context modelling goals

We aim at providing basic conceptual foundations for context modelling, which allow designers to represent
(i) relevant elements of a context-aware application’s universe of discourse and
(ii) particular state-of-affairs of interest

We consider results from foundational ontologies to support our conceptual context modeling approaches

We focus on situation specification and realisation

Foundational ontologies

Context
- the set of possibly interrelated conditions in which an entity exists
Context situations

- Context models presented so far give a static view of context
- Context-aware applications should be able to react on Situations → specific configurations of entities and context conditions
- We have applied the ontological notion of Situation for that → particular state-of-affairs

Context situation specification

- Standard UML 2.0 class diagrams using OCL 2.0 for constraints
- OCL constraints define the conditions under which context situations of a certain type exist
- An invariant defines a predicate that must hold for all instances of the target class
- Mature technology for a model-driven approach

Context situation types

- Intrinsic Situation
- Relational Situation
- Formal Relation Situation
- Situation of Situations
- Combined Situations
Situation Fever

\[
\text{Context } \text{SituationFever inv:} \\
\quad \text{temp} \rightarrow \text{person hasTemperature AND} \\
\quad \text{person hasTemperature.value > 38}
\]

Situations for specification

\[
\begin{align*}
\text{IntrinsicContext} & : \text{Temperature} \\
\text{Person} & : \text{Temperature} \\
\text{SituationFever} & : 0..1 \\
\end{align*}
\]

Situation Connected

\[
\begin{align*}
\text{Context } \text{SituationConnected inv:} \\
\quad \text{not device hasConnection.oclIsUndefined()}
\end{align*}
\]

Situations for specification

\[
\begin{align*}
\text{RelationalContext} & : \text{Connection} \\
\text{Device} & : \text{Connection} \\
\text{Network} & : \text{Connection} \\
\text{SituationConnected} & : 0..1 \\
\end{align*}
\]

Situation Switch

\[
\begin{align*}
\text{Context } \text{SituationSwitch inv:} \\
\quad \text{wlan.device = bluetooth.device AND} \\
\quad \text{wlan.device hasConnection.network.oclsIsTypeOf(WLAN)) AND} \\
\quad \text{bluetooth.device hasConnection.network.oclsIsTypeOf (Bluetooth)) AND} \\
\quad \text{bluetooth.initTime - wlan.finalTime < 1)}
\end{align*}
\]

Foundational Context Concepts

(UML profile)

Artifacts for specification

- Context profile: <<Entity>>, <<IntrinsicContext>>, ..., <<RelationalContext>>, ...<IntrinsicSituation>>, ... <FormalRelationSituation>, <<RelationalSituation>>, ..., <<RelationalSituation>>, ...

Products of specification

- Context Models: Person, Temperature, GeoLocation, GeoLocationCoordinate, Device, etc.
- Situation Models: SituationFever, SituationConnected, SituationPresentation, etc.

Situation realisation

Rule-based approach

- Fits nicely the nature of situation detection
- Rules (OCL invariants) are repeatedly applied to a collection of facts (context information)

Jess

- Shadow facts
- Main components: working memory and rule-base

Situation Realization (overview)
### Situation Lifecycle

- **Detect Situation**: Situation fact exists
- **Invariant holds**: Situation fact does not exist and invariant holds
- **Create Situation Fact**
- **Deactivate Situation Fact**

#### Situation Detection

**Creation Rule**

- \( (\text{situation type invariant}) \) and \( (\text{situation exists}) \)
- => deactivate (situation)
- \([\text{RaiseEvent}()]\)

**Deactivation Rule**

- \( (\text{not (situation type invariant)}) \) and \( (\text{situation exists}) \)
- => deactivate (situation)
- \([\text{RaiseEvent}()]\)

### Context models to Java

- **Octopus** ([www.klasse.nl/octopus/index.html](http://www.klasse.nl/octopus/index.html))
- Generates Java code from UML classes, and statically checks OCL constraints
- UML classes to Java classes
- Associations to class attributes
  - **One-to-one** (one attribute in each class)
  - **One-to-many** (one of the attributes is a collection)
  - **Many-to-many** (both attributes are collections)
- Subsets association

### Situation models to Java + Jess

- **Table**

<table>
<thead>
<tr>
<th>Context Models to Java + Jess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octopus</td>
</tr>
<tr>
<td>Generates Java code from UML classes, and statically checks OCL constraints</td>
</tr>
<tr>
<td>UML classes to Java classes</td>
</tr>
<tr>
<td>Associations to class attributes</td>
</tr>
<tr>
<td>One-to-one (one attribute in each class)</td>
</tr>
<tr>
<td>One-to-many (one of the attributes is a collection)</td>
</tr>
<tr>
<td>Many-to-many (both attributes are collections)</td>
</tr>
</tbody>
</table>

### Distribution

- **Service-oriented approach**
  - Components encapsulate Jess engines, and situation information is exchanged by means of the component services
- **DJess**
  - Separate engines virtually share working memory
  - Rule engines running on different nodes can apply rules on shared facts

### Conclusions

- **Context models** help understanding context concepts and how they relate to each other
- Context models are **static**
- Situations allow one to define state-of-affairs of concern for context-aware applications
- Behaviours can be defined in terms of how the system evolves from situation to situation!
- Situations can be used to define conditions that trigger a rule system, e.g., in ECA rules
- Situations can be composed of situations themselves → modularization of the situation models, improving organization and reuse of situation specifications
Conclusions

Rule-based situation realisation allows attentive situation detection as opposed to query-based solutions.

Model-driven approach:
- Specification elements are systematically mapped to realization elements.
- UML (including OCL) is mature technology for model-driven developments.