A Formal Model and Composition Language for Context-Aware Service Protocols

Javier Cubo, Carlos Canal, Ernesto Pimentel, Gwen Salaün
University of Málaga, Spain

Presented by John Plaice
The University of New South Wales

CASTA, August 24, 2009

Introduction

A model is developed for context-aware distributed objects in which the context is a set of shared global variables, the semantics uses synchronous rendez-vous, and the dependencies between shared actions can be rendered explicit.

A semi-automatic mechanism is defined for creating these dependencies.

These is a case study for going on the road again.

Values, Types, Operators

- Let $\mathbf{Type}(\ni t)$ be a set of types. For a given type $t \in \mathbf{Type}$, we write $\mathbf{Val}_t$ for the set of possible values for t. We write $\mathbf{Val} = \bigcup_{t \in \mathbf{Type}} \mathbf{Val}_t$.
- Let $\mathbf{Op}(\ni f)$ be a set of operators. An *operator signature* Σ over $\mathbf{Op}$ is a mapping $\Sigma \in \mathbf{Op} \rightarrow \mathbf{Type}^+ \times \mathbf{Type}$ defining the types for each of the operators allowed in expressions.
- Let $x \in \mathbf{X}$ be a set of variables. Then the set of valid expressions of signature Σ over $\mathbf{X}$ is written $\Sigma(\mathbf{X})$.

Contexts

- A *context attribute* $A \in \mathbf{A}$ is a string. Examples are `language`, `temperature`, etc.
- A *context signature* T is a mapping $T : \mathbf{A} \rightarrow \mathbf{Type} \times \mathbf{Bool} \times \mathbf{Bool}$, with $T(A) = (t_A, s_A, p_A)$, where
  - t_A is the type of A;
  - s_A determines if A is static (true) or dynamic;
  - p_A determines if A is public (true) or private.
- A *context* C of type T is a mapping $C : \mathbf{A} \rightarrow \mathbf{Val}$ with $C(A) \in \mathbf{Val}_{t_A}$, as above.

Context-Aware Transition Systems

- A *context-aware label* ℓ is one of:
  - τ (internal action);
  - $?(B, a, (x_1, \ldots, x_n))$ (reception of message);
  - $!(B, a, (E_1, \ldots, E_n))$ (emission of message);

  where B means Boolean expression and a is a message name.
- An *atomic context-aware protocol* P is a 6-tuple $(p, L, S, s_I, S_F, \phi)$ where:
  - p is the name of the protocol;
  - L is a set of transition labels, as above;
  - S is a set of states;
  - s_I is the initial state;
  - S_F is the set of correct final states;
  - $\phi : S \times L \to S$ is a transition function.

Composing Context-Aware Protocols

Context-aware protocols can be built up through expressions:

$$\begin{array}{rcl} P & ::= & P_{\text{atomic}} \\ & | & P.P \\ & | & P + P \\ & | & P \parallel_D P \end{array}$$

where D is a *data dependency* of the form $(p_1, \ell_1) < (p_2, \ell_2)$, implying that label ℓ_1 in protocol p_1 must be executed before label ℓ_2 in protocol p_2.

Operational semantics of multiple protocols

$$\frac{\langle s_i, \mathcal{E}_i\rangle \xrightarrow{a!E} \langle s_i', \mathcal{E}_i\rangle \quad \langle s_j, \mathcal{E}_j\rangle \xrightarrow{a?x} \langle s_j', \mathcal{E}_j\rangle \qquad i, j \in 1..n \quad i \neq j \quad \mathcal{E}_j' = \mathcal{E}_j \dagger \{x \mapsto [\![E]\!]\mathcal{E}_j\}}{\{\ldots, \langle s_i, \mathcal{E}_i\rangle, \ldots \langle s_j, \mathcal{E}_j\rangle, \ldots\} \xrightarrow{a!E} \{\ldots, \langle s_i', \mathcal{E}_i\rangle, \ldots \langle s_j', \mathcal{E}_j'\rangle, \ldots\}}$$

$$\frac{\langle s_i, \mathcal{E}_i\rangle \xrightarrow{\tau} \langle s_i', \mathcal{E}_i\rangle \quad i \in 1..n}{\{\ldots, \langle s_i, \mathcal{E}_i\rangle, \ldots\} \xrightarrow{\tau} \{\ldots, \langle s_i', \mathcal{E}_i\rangle, \ldots\}}$$

Operational semantics of composition language

$$\frac{\langle s_1, \mathcal{E}_1 \rangle \xrightarrow{\ell_1} \langle s_1', \mathcal{E}_1 \rangle \quad ((p_1, \ell_1) < (p_2, \ell_2)) \in D \qquad \neg in\_a\_loop(s_1, \ell_1, \phi_1) \quad D' = remove((p_1, \ell_1), D)}{\langle s_1, \mathcal{E}_1 \rangle \parallel_D \langle s_2, \mathcal{E}_2 \rangle \xrightarrow{\ell_1} \langle s_1', \mathcal{E}_1 \rangle \parallel_{D'} \langle s_2, \mathcal{E}_2 \rangle}$$

$$\frac{\langle s_1, \mathcal{E}_1 \rangle \xrightarrow{\ell_1} \langle s_1', \mathcal{E}_1 \rangle \quad ((p_1, \ell_1) < (p_2, \ell_2)) \in D \qquad in\_a\_loop(s_1, \ell_1, \phi_1)}{\langle s_1, \mathcal{E}_1 \rangle \parallel_D \langle s_2, \mathcal{E}_2 \rangle \xrightarrow{\ell_1} \langle s_1', \mathcal{E}_1 \rangle \parallel_D \langle s_2, \mathcal{E}_2 \rangle}$$

Detecting dependencies

- Algorithm 1: Find possible dependencies (semantic matching);
- User: Manually selects among the results of Algorithm 1;
- Algorithm 2: Transitive closure of the user's choices.

Concluding remarks

- An example is developed with users in cars being driven down the road and interacting with services provided through their mobile devices.
- Users must decide what dependencies are relevant, may end up providing inconsistent or incomplete dependency sets.
- Future work involves handling dynamic composition specifications.