Separation of Concerns and Consistent Integration in Requirements Modelling

Xin Chen, Zhiming Liu and Vladimir Mencl
UNU-IIST, Macao
http://www.iist.unu.edu/∼lzm

1. Introduction
2. Models of Requirements
3. Consistency
4. Integration
4. Conclusions

Introduction

Development of a dependable software system is a *complex* task.

- Separation of concerns
  - The key for dealing with complexity
  - The main motivation for multi-view modelling techniques, such as UML
- Formal modelling and analysis
  - One of the main means to ensure high dependability

Issues in *Formal* and *Multi-view* modelling

- Consistency of different views (specified in different notations)
- Integration of methods and tools for reasoning and analysis via a *common semantic model*

Multi-view Model of Requirements

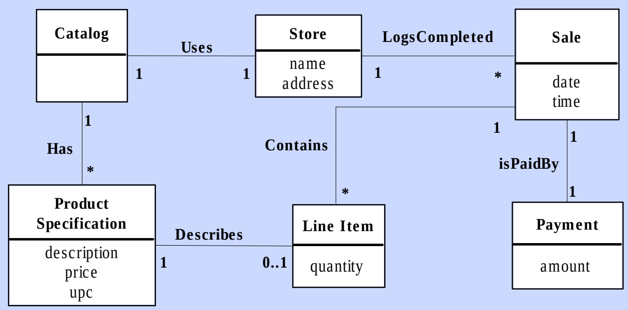
$\mathcal{M} = \langle \Gamma, \Delta, \Omega, \Phi, \Theta \rangle$

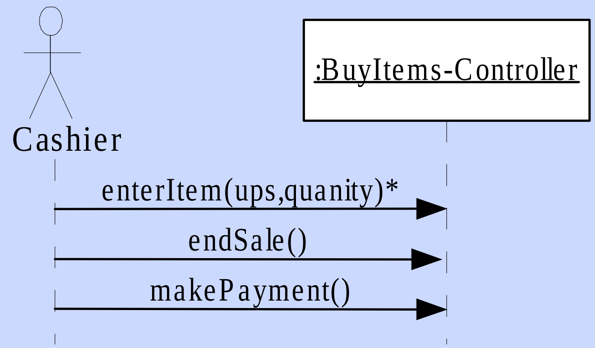
- Γ: *class diagram*, including a *use-case controller class* for each use case.
- Δ: a family of *sets* of *system sequence diagrams*, one set for each use case controller.
- Ω: a set of *state diagrams*, one for each use case controller.
- Φ: a *specification mapping* that assigns each *use-case operation* $m(T_1\ \mathit{in}; T_2\ \mathit{out})$ with a pair of pre- and post-conditions

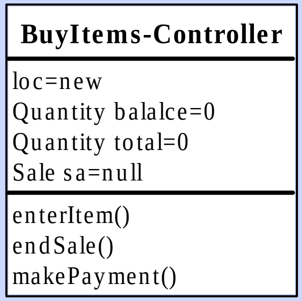
$$m(T_1\ \mathit{in}; T_2\ \mathit{out})\{\mathit{pre}_m \vdash \mathit{Post}_m\}$$

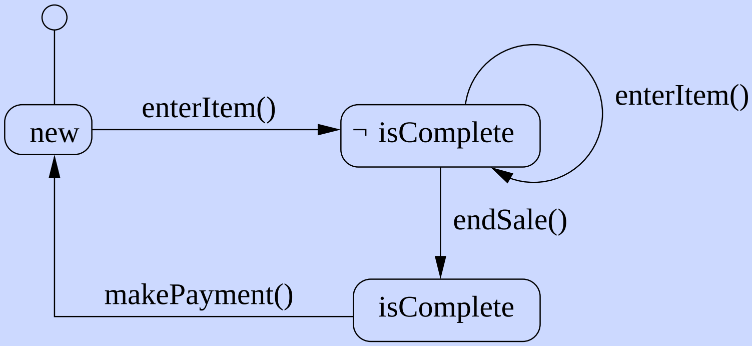
- Θ: a system *invariant*.

Example: Model of POS Requirements









Consistency and Integration??

Formalities

- *Design*: $p(in\alpha) \vdash R(in\alpha, out\alpha) \stackrel{def}{=} (ok \wedge p) \Rightarrow (ok' \wedge R)$
- *Assignment:* $x := e \stackrel{def}{=}$ *true* $\vdash (x' = val(e)) \wedge (y' = y)$
- Designs are closed under programming constructors
- *Guard:* $g_\top \stackrel{def}{=}$ *skip* $\lhd\, g \,\rhd$ *false*
- *Guarded Design:* $g_\top; D$, behaves like D when g holds, and deadlocks otherwise
- *Parallelism* and *non-deterministic choice*

$$
\begin{array}{lcl}
(p_1 \vdash R_1) \parallel (p_2 \vdash R_2) & \stackrel{def}{=} & (p_1 \wedge p_2 \vdash R_1 \wedge R_2) \\
(p_1 \vdash R_1) \sqcap (p_2 \vdash R_2) & \stackrel{def}{=} & (p_1 \vee p_2) \vdash ((p_1 \wedge R_1) \vee (p_2 \wedge R_2))
\end{array}
$$

- *Weakest Precondition:* $\mathbf{wp}(p \vdash R, q) \stackrel{def}{=} p \wedge \neg(R; \neg q)$
- The calculus of designs is extended to OOP: rCOS in TCS 365(1-2)

Semantic Models of Different Views

- The class diagram:
  - *Types* – both classes and primitive data types.
  - *State space* of the system
- *Sequence diagrams: prefix closed* set of finite traces of $?m(v;u)$ and $!c.n(v;y)$.

  Parameters and return events can be omitted, when guards do not depend on input parameters and all invocations will complete and return.

  Set union will be used when there is more than one sequence diagram for a use case.

Example

$$\begin{aligned} \mathit{Tr}(\mathit{BuyItems}) \quad &\stackrel{def}{=} \quad ?\mathit{enterItem}()^* + ?\mathit{enterItem}()^+ \cdot ?\mathit{endSale}() \\ &+ \quad ?\mathit{enterItem}()^+ \cdot ?\mathit{endSale}() \cdot ?\mathit{makePayment}() \end{aligned}$$

Semantic Models of Different Views: Sate Diagram

A *state diagram:* $\mathcal{S}_C = (\Sigma, \sigma_0, \zeta)$ for a use case controller of C

- Σ: a set of *control states*
- σ_0: an initial state,
- $\zeta \subseteq \Sigma \times \textit{Label} \times \Sigma$: a transition relation
  - Each $t \in \zeta$ is labelled with a pair $\ell = \langle m(\textit{in}; \textit{out}), g\rangle$
  - The change of control state by a transition $t = \sigma \xrightarrow{\ell} \sigma'$ is specified as

$$\textit{cs}(t) \stackrel{def}{=} (\textit{state} = \sigma)_{\top}; (\textit{state}' = \sigma')$$

- For each $m()$ of C, define

$$m() \; \{\sqcap_{t \in \mathcal{E}(m())} cs(t) \;\}$$

  where $\mathcal{E}(m())$ is a set of transitions that has $m()$ as their triggering event.

Example

$$\begin{aligned} \mathit{enterItem}()\{ \quad & ((\mathit{New})_\top ; (\neg \mathit{IsComplete}' \wedge \neg \mathit{New}')) \\ & \sqcap (\neg \mathit{New} \wedge \neg \mathit{IsComplete})_\top \} \end{aligned}$$

where

$$\begin{aligned} New & \stackrel{def}{=} state = new \\ New' & \stackrel{def}{=} state' = new \end{aligned}$$

Integrating Functionality into Transitions

- For a transition $t = \sigma \xrightarrow{\ell} \sigma'$ with $\ell = \langle m(), g \rangle$, the *integrated specification* of both control state change and functionality specification $\Phi(m)$ is defined as

$$Spec(t) \stackrel{def}{=} (g \wedge \mathit{state} = \sigma)_{\top}; \Phi(m) \parallel (\mathit{state}' = \sigma')$$

- Let $t_i = \sigma_i \xrightarrow{\ell_i} \sigma_i'$, with $t_i = \langle m(), g_i \rangle$, $i \in 1..k$ be the transitions of $\mathcal{M}_C$ triggered by event $m()$.

  The *integrated specification* of method $m()$ of class C is defined as

$$Spec(C :: m()) \stackrel{def}{=} \{\mathit{Spec}(t_1) \sqcap \ldots \sqcap \mathit{Spec}(t_k)\}$$

- $\mathit{body}(C :: m()) \stackrel{def}{=} \mathit{Spec}(t_1) \sqcap \ldots \sqcap \mathit{Spec}(t_k)$

  $\mathcal{G}(C :: m()) \stackrel{def}{=} \exists i \cdot (1 \le i \le k \wedge g_i \wedge (\mathit{state} = \sigma_i))$

Example: $\Phi(\mathit{enterItem}(\mathit{upc} : \mathit{UPC}, \mathit{qty} : \mathit{Quantity}))$

$$\begin{array}{lcl}
\mathit{known}(\mathit{upc}) & \stackrel{def}{=} & \exists \mathit{sp} \in \Pi(\mathit{ProductSpecification}) \cdot (sp.upc = upc) \\
\mathit{newLine}(\mathit{li}) & \stackrel{def}{=} & \exists \mathit{li} \in \Pi(\mathit{LineItem}) \cdot \mathit{LineItem.New}(\mathit{li}); (\mathit{li.quantity}' = \mathit{qty}) \\
\mathit{addLtoSale}(x, li) & \stackrel{def}{=} & \exists \mathit{y} \in \Pi(\mathit{Contains}) \cdot \mathit{Contains.New}(y); \\
 & & (y.sale' = x) \wedge (y.\mathit{line}' = li) \\
\mathit{addTotal} & \stackrel{def}{=} & \mathit{total}' = \mathit{total} + \mathit{qty} \times \mathit{sp.price} \\
\mathit{enterItem@new} & \stackrel{def}{=} & \mathit{known}(\mathit{upc}) \Rightarrow (\mathit{Sale}.\mathit{New}(sa); \\
 & & \bigvee_{\mathit{sp.upc}=\mathit{upc}} (\mathit{NewLine}(\mathit{li}); \mathit{addLtoSale}(sa, li)) \wedge \mathit{addTotal})) \\
—@\neg \mathit{isComplete} & \stackrel{def}{=} & \mathit{known}(\mathit{upc}) \Rightarrow (\mathit{NewLine}(\mathit{li}); \mathit{addLtoSale}(sa, li)) \wedge \mathit{addTotal}) \\
\Phi(\mathit{enterItem}()) & \stackrel{def}{=} & \mathit{enterItem@}(new) \vee \mathit{enterItem@}\neg \mathit{isComplete}
\end{array}$$

Static Consistency

- All types, classes and attributes that are used in the definitions of $\Phi(m())$ are defined in class diagram Γ.
- All the specification statements are *well-defined* in the context of Γ.
- The system invariant Θ has to be satisfied by the functionality specifications $\Phi(n())$ of all methods.

Dynamic Consistency

- A model $\mathcal{M}$ is *consistent* if all traces of each use case are *completely realisable* by the state diagram and the functional specification of the use case controller class.
- Consistency ensures that deadlock will not occur if each actor follows its *interaction protocol* specified by the traces of its use-case sequence diagrams.
- For a an initial value v_0 of the controller class, any trace $\mathit{tr} = m_1() \ldots m_k() \in \mathit{Tr}(C)$ of a use case class C, and any prefix $m_1() \ldots m_j()$ of tr,

$$\mathbf{wp}(v' = v_0 \wedge state' = s_0; \mathit{body}(m_0()); \ldots; \mathit{body}(m_j()), \mathcal{G}(m_{j+1}())) = \mathit{true}$$

- We can easily check that the model for POS is consistent

Use case decomposition

A use case U_1 may *include* a use case U_2, such as using the "ref" keyword in UML

1. Ensure the traces of U_1 include the traces of U_2 as sub-traces

   The state diagram U_1 should also include the state diagram of U_2. Statecharts can be used for this.

2. The actors directly interact with controller UC_1, and UC_1 *delegates* the request to UC_2.

   The consistency checking then requires the *composition* of state diagrams of UC_1 and UC_2 together with the consideration of *internal interactions* — More like a design model

Conclusion

- Discussed the issues of separation of concerns of different aspects of a software model, the consistency and integration.
- The approach allows to treat different properties with different techniques and tools: static functionality analysis and refinement, compatibility among interaction protocols, etc.
- Also, some model, such as a trace model, are easier to interpret, while another, such as operational state transition model, may be easier for verification.
- Future work: Consistency refinement, compositionality, automatic checking, separation of concerns and integration in component based modelling (Component rCOS by He Jifeng, Xiaoshan Li and Zhiming Liu)

4th International Colloquium on Theoretical Aspects of Computing 22-28 September 2007, Macao SAR, China

Associated Events

- *School on Domain Modelling and the Duration Calculus*, 17-21 September 2007, Shanghai, China
- *Festschrift Symposium* dedicated to the 70th birthdays of Dines Bjrner and Zhou Chaochen, 24-25 September 2007, Macao
- *Workshops*, 22-23 September, 2007, Macao

Deadline of Submission: 20 April 2007

Proceedings: Springer LNCS

http://www.iist.unu.edu/ictac07