AN INTEGRATED APPROACH TO DESIGN PATTERNS FORMALIZATION

Toufik Taibi, UAEU, UAE

Table of Contents

Introduction/Motivation

- A design pattern is a description of a set of proven solutions to a set of recurring problems within a context.
- Reusing patterns improves quality and productivity of software designs.
- Patterns are mostly described using informal means (text and graphical notations) that lack well-defined semantics.

Introduction/Motivation

- As the number of patterns is growing, informal specifications were found ambiguous and sometimes misleading in understanding and properly applying patterns.
- We present BPSL that accurately describe patterns in order to allow rigorous reasoning about them their instances and their composition.

Introduction/Motivation

- We focus on the solution element of the pattern and not on others aspects such as intent, problem, etc…
- The main focus of this work is to show how BPSL can be used to formally specify patterns at 3 levels of abstraction: patterns, pattern composition and instances of patterns.

Balanced Pattern Specification Language (BPSL)

- BPSL uses First Order Logic (FOL) and Temporal Logic of Actions (TLA) as formal basis to specify the structural and behavioral aspect of patterns respectively.
- The structural part of BPSL is called S_{BPSL} where the "S" stands for structural.
- The behavioral part of BPSL is called B_{BPSL}, where the "B" stands for behavioral.

S_{BPSL}

- S_{BPSL} is a very simple first-order language having the following characteristics:
- Each formula is a sentence (closed well-formed formula) as S_{BPSL} does not contain free variables.
- S_{BPSL} does not support function symbols, restricts predicates to have two arguments and requires only the usage of the existential quantifier ($\exists$).
- Thus, its semantics completely derives from FOL

S_{BPSL}

- Variables and constants of S_{BPSL} are many-sorted.
- Variable and constant symbols represent classes, typed variables and methods.
- The sets of classes (or references to classes), typed variables and methods are designated *C*, *V* and *M* respectively.
- Typed variables represent variables of any predefined or user-defined types except elements of set *C*.

S_{BPSL}

- Binary predicate symbols represent permanent relations between variables.
- S_{BPSL} defines a set of *primary* permanent relations based on which other permanent relations can be built (Table 1)
- The term "permanent" is used to differentiate these relations with "temporal" relations defined later.

S_{BPSL}

- Primary permanent relations represent the smallest set on top of which any other permanent relation can be built (for example forwarding).
- $Forwarding(m_1,m_2) \Leftrightarrow Invocation(m_1,m_2) \wedge Argument(a_1,m_1) \wedge \ldots \wedge Argument(a_n,m_1) \wedge Argument(a_1,m_2) \wedge \ldots \wedge Argument(a_n,m_2)$, where $m_1, m_2 \in M$ and $\{a_1, \ldots, a_n\} \subset C \cup V$

Table 1, Primary Permanent Relations

Name	Domain	Intent
Defined-in	$M \times C$	Indicates that a method is defined in a certain class.
	$V \times C$	Indicates that a typed variable is defined as an attribute in a certain class.
Reference-to-one (-many)	$C \times C$	Indicates that one class defines a member whose type is a reference to one (many) instance(s) of the second class.
Inheritance	$C \times C$	Indicates that the first class inherits from the second.
Creation	$M \times C$	Indicates that a method contains an instruction that creates a new instance of a class.
	$C \times C$	Indicates that one of the methods of a class contains an instruction that creates a new instance of another class.
Invocation	$M \times M$	Indicates that the first method invokes the second method.
	$C \times M$	Indicates that a method of a class invokes a specific method of another class.
	$M \times C$	Indicates that a specific method of a class invokes a method of another class.
	$C \times C$	Indicates that a method of a class invokes a method of another class.
Argument	$C \times M$	Indicates that a reference to a class is an argument of a method.
	$V \times M$	Indicates that a typed variable is an argument of a method.
Return-type	$C \times M$	Indicates that a method returns a reference to a class.

A Summary of TLA Concepts

- In TLA, a semantic is given by assigning a semantic meaning $[\![F]\!]$ to each syntactic object *F*.
- The semantics of TLA is defined in terms of *states*, where a state is an assignment of values to variables.
- A state *s* assigns a values *s(x)* to a variable *x*.
- The collection of all possible states is denoted ***St***.
- A state is a function from the set of variables ***Var*** to the set of values ***Val***.
- Thus, $s[\![x]\!]$ denotes *s*(*x*). The meaning $[\![x]\!]$ of variable *x* is a mapping from states to values.

A Summary of TLA Concepts

- A state function is a non-Boolean expression built from variables and constant symbols. The meaning ⟦f⟧ of a state function f is a mapping from the collection St of states to the collection Val of values.
- A postfix functional notation is used letting s⟦f⟧ denote the values that ⟦f⟧ assigns to state s. s⟦f⟧ ≜ f(∀ 'v':s⟦v⟧/v)
- f(∀ 'v':s⟦v⟧/v) denotes the value obtained from f by replacing v by s⟦v⟧ for all variables v. The symbol ≜ means equal by definition.
- A state predicate (or predicate) is a Boolean expression built from variable and constant symbols. ⟦P⟧ is a mapping from states to Booleans, so s⟦P⟧ equals true or false for every state s. A state s satisfies a predicate P iff s⟦P⟧ equals true.

A Summary of TLA Concepts

- An action is a Boolean-valued expression formed from variables, primed variables and constant symbols.
- An action represents a relation between old states and new states, where the unprimed variables refer to the old state and the prime variables refer to the new state.
- The meaning ⟦A⟧ of an action A is a function that assigns a Boolean s⟦A⟧t to a pair of states s,t.
- s⟦A⟧t is obtained from A by replacing each unprimed variable v by s⟦v⟧ and each primed variable v' by t⟦v⟧ as follows: s⟦A⟧t ≜ A(∀ 'v':s⟦v⟧/v,t⟦v⟧/v').

A Summary of TLA Concepts

- A pair of successive states is called a **step**.
- The pair of states *s,t* is called an "*A* step" iff $s[\![A]\!]t$ equals true.
- A predicate *P* can also be viewed as an action that does not contain primed variables.
- Thus $s[\![P]\!]t$ is a Boolean which equals $s[\![P]\!]$ for any states *s* and *t*.
- A pair of states *s,t* is a *P* step iff it satisfies *P*.
- For any state function or predicate *F*, we define *F'* to be the expression obtained by replacing each variable *v* in *F* by the primed variable *v'*.
- $F' \triangleq F(\forall \text{'}v\text{'} : v'/v)$
- *If P is a predicate symbol then P' is an action and* $s[\![P']\!]t$ *equals* $t[\![P]\!]$ *for any state s and t.*

A Summary of TLA Concepts

- An action *A* is said to be valid, written as $\models A$, iff every step is an *A* step. Formally $\models A \triangleq \forall s,t \in \mathbf{St}: s[\![A]\!]t$.
- A special case of this is $\models P \triangleq \forall s \in \mathbf{St}: s[\![P]\!]$.
- A valid action is true regardless of the values substituted for primed and unprimed variables.
- For any action *A*, *Enabled A* is a predicate that is true for a state iff it is possible to take an *A* step starting in that state.
- Semantically, *Enabled A* is defined by $s[\![Enabled\ A]\!] \triangleq \exists t \in \mathbf{St}: s[\![A]\!]t$ for any state *s.*
- A temporal formula is built from elementary formulas using Boolean operators (basically $\wedge$ and $\neg$ as the others are derived from these two) and the unary operator $\Box$ (read always).

A Summary of TLA Concepts

- TLA is a special case of simple temporal logic. The semantics of temporal logic is based on behaviors, where a behavior is an infinite sequence of states.
- The meaning of a temporal formula is defined in terms of the elementary formulas it contains.
- A temporal formula is interpreted as an assertion about behaviors.
- Formally, the meaning $[\![F]\!]$ of a formula F, is a Boolean-valued function on behaviors.
- Let $\sigma[\![F]\!]$ denote the Boolean value that formula F assigns to behavior σ, and we say that σ satisfies F iff $\sigma[\![F]\!]$ equals true.
- The definition of $\sigma[\![F \wedge G]\!]$ and $\sigma[\![\neg F]\!]$ are given below:
- $\sigma[\![F \wedge G]\!] \triangleq \sigma[\![F]\!] \wedge \sigma[\![G]\!]$ and $\sigma[\![\neg F]\!] \triangleq \neg\sigma[\![F]\!]$

A Summary of TLA Concepts

- Let $\langle s_0, s_1, s_2, \ldots \rangle$ denote the behavior whose first state is s_0, second state is s_1, and so on.
- $\langle s_0, s_1, s_2, \ldots \rangle [\![\Box F]\!] \triangleq \forall\, n \in \textbf{\textit{Nat}}: \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [\![F]\!]$
- ***Nat*** is the set of natural numbers. $\Box F$ asserts that F is *always* true. For any temporal formula F, let $\Diamond F$ be defined by $\Diamond F \triangleq \neg \Box \neg F$.
- The above formula asserts that it is not the case that F is always false. Thus, $\Diamond F$ asserts that F is *eventually* true.
- $\langle s_0, s_1, s_2, \ldots \rangle [\![\Diamond F]\!] \triangleq \exists\, n \in \textbf{\textit{Nat}}: \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [\![F]\!]$
- Therefore, a behavior satisfies $\Diamond F$ iff F is true at some time during the behavior.

A Summary of TLA Concepts

- $\langle s_0,s_1,s_2,\ldots\rangle[\![\Box\Diamond F]\!] \triangleq \forall\ n\in \mathbf{Nat}: \exists\ \ m\in \mathbf{Nat}\ \langle s_{n+m},s_{n+m+1},s_{n+m+2},\ldots\rangle[\![F]\!]$

- A behavior satisfies $\Box\Diamond F$ iff F is true at infinitely many times during the behavior i.e. that F is true infinitely often.
- The formula $\Diamond\Box F$ asserts that eventually F is always true. Thus, a behavior satisfies $\Diamond\Box F$ iff there is some time such that F is true from that time on.
- $\langle s_0,s_1,s_2,\ldots\rangle[\![\Diamond\Box F]\!] \triangleq \exists\ \ m\in \mathbf{Nat}: \forall\ n\in \mathbf{Nat}: \langle s_{n+m},s_{n+m+1},s_{n+m+2},\ldots\rangle[\![F]\!]$

- A temporal formula F is said to be valid, written $\models F$ iff it is satisfied by all possible behaviors.
- $\models F \triangleq \forall\ \sigma\in \mathbf{St}^{\infty}: \sigma[\![F]\!]$
- Where $\mathbf{St}^{\infty}$ denotes the collection of all behaviors (infinite sequences of elements of **St**).

A Summary of TLA Concepts

- A *stuttering* step on an action *A* under the state function *f* occurs when either the action *A* occurs or the variables in *f* are unchanged.
- For any action *A* and state function *f*, we let $[A]_f \triangleq A \vee (f' = f)$.
- Let *A* be any action and *f* any state function. Then $\neg A$ is also an action, so $\neg\Box[\neg A]_f$ is a TLA formula. Applying the definitions gives:
- $\neg\Box[\neg A]_f \equiv \Diamond\neg[\neg A]_f \equiv \Diamond\neg(\neg A \vee (f' = f)) \equiv \Diamond(A \wedge (f' \neq f))$
- We define action $\langle A \rangle f$ by $\langle A \rangle f \triangleq A \wedge (f' \neq f)$
- The above calculation shows that $\Diamond\langle A \rangle_f \equiv \neg\Box[\neg A]_f$, so it is a TLA formula.

A Summary of TLA Concepts

- Reasoning about fairness is an important aspect when modeling concurrency.
- Fairness is concerned with progress properties, facts that ensure that no process is consistently neglected
- In TLA, two types of fairness properties are declared
- An action is said to satisfy the weak fairness condition if at all times either it is eventually executed or it eventually becomes disabled.
- An action is said to satisfy the strong fairness condition if at all times either it will eventually be executed or eventually it will be disabled at all later states.
- The definitions for these two conditions are given as:
- $WF_f(A) \triangleq (\Box\Diamond\langle A\rangle_f) \vee (\Box\Diamond\neg Enabled\ \langle A\rangle_f)$
- $SF_f(A) \triangleq (\Box\Diamond\langle A\rangle_f) \vee (\Diamond\Box\neg Enabled\ \langle A\rangle_f)$

B_{BPSL}

- B_{BPSL} completely derive from TLA
- TLA formulas can be written as $\Phi \triangleq Init_{\Phi} \wedge \Box[N]_{u} \wedge F$, where:
- $Init_{\Phi}$ is a predicate specifying the initial values of variables.
- *N* is the system's next-state relation (disjunction of actions)
- u is an n-tuple of variables
- *F* is the conjunction of formulas of the form $SF_f(A)$ and/or $WF_f(A)$, where A is an action representing a subset of the set of actions.

Temporal Relations

- B_{BPSL} uses a special type of predicates called temporal relations. A temporal relation can be defined as follows:
- *TR(C_1<cardinality>,C_2<cardinality>)*, where *TR* is the name of the temporal relation, C_1 and C_2 are classes involved by this relation, and *cardinality* represents the number of instances (objects) of each class that participate in the relation.
- Cardinality can be represented as either a closed interval <*n..m*>, where *n* and *m* represent any two positive integers or <*> to depict any possible number of instances.

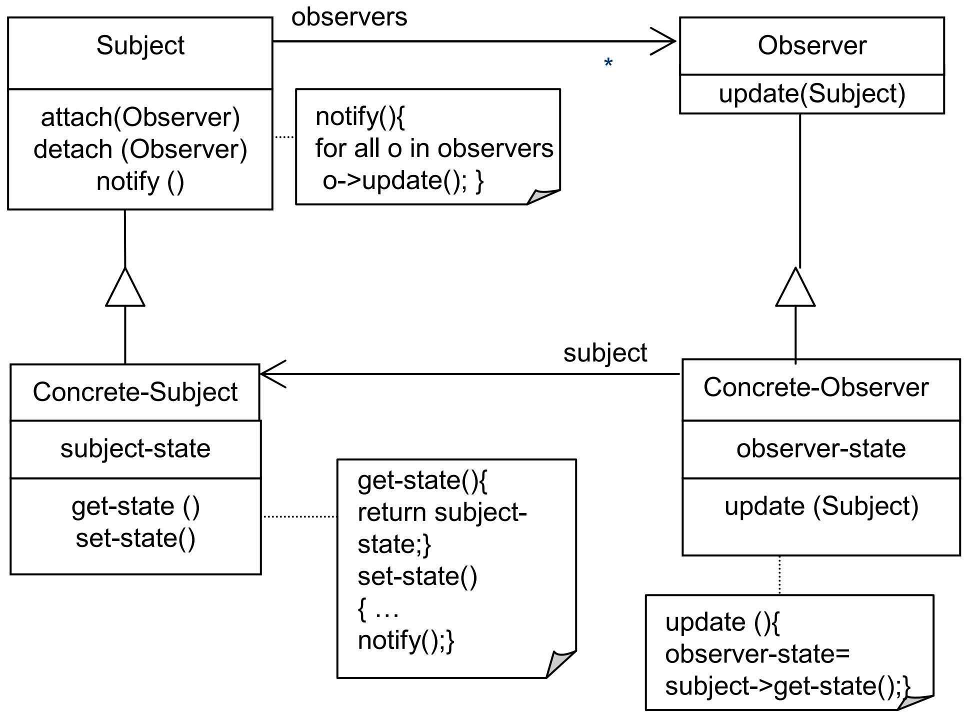
Temporal Relations

- When used in actions, temporal relations can take different forms each of which `
- $TR(o_1,o_2)$ depicts that an object o_1 of a class C_1 is currently linked through *TR* with an object o_2 of a class C_2.
- $\neg TR(o_1,o_2)$ depicts that objects o_1 and o_2 are no longer linked through *TR*.
- $TR(o_1,C_2)$ depicts that object o_1 is linked with all objects of the class C_2.
- $\neg TR(o_1,C_2)$ depicts that object o_1 is not linked through *TR* with any object of class C_2.
- $\neg TR(C_1,C_2)$ depicts that no object of class C_1 is linked through *TR* with any object of class C_2.

BPSL Formula (Table 2)

$\exists\ x_1,\ldots,x_{q1},y_1,\ldots,y_{q2} \subset C \cup V \cup M \quad \wedge_i PR_i\ (x_j,y_k)$ $\{1<=i<=q,\ 1<=j<q_1;1<=k<=q_2\}$	
$TR_1(z_1<cz_1>,t_1<ct_1>),\ldots,TR_m(z_m<cz_m>,t_m<ct_m>) \in TR;$ $\{z_1,\ldots,z_m,t_1,\ldots,t_m \in C\}$ $u_1,\ldots,u_n \subset (\text{Member of } C) \cup V;$	
$Init_\Phi \triangleq P$	$\{P \text{ is the initial predicate}\}$
$N \triangleq A1 \vee \ldots \vee A_r$	$\{A_1 \ldots A_r \text{ are actions}\}$
$u \triangleq \langle u_i,\ldots,u_j \rangle$	$\{1<=i<=n \text{ and } 1<=j<=n\}$
$\Phi \triangleq Init_\Phi \wedge \Box[N]_u \wedge WF_u(A)$	$\{A \equiv A_{i1} \vee \ldots \vee A_{i2}, 1<=i1<=i2<=r\}$

Case Study 1:Observer Pattern



BPSL Specification of Observer Pattern (Table 3)

$\Phi_1 \equiv \exists$ subject, concrete-subject, observer, concrete-observer $\in$ C;
subject-state, observer-state $\in$ V;
attach, detach, notify, get-state, set-state, update $\in$ M:
Defined-in(subject-state, concrete-subject)$\wedge$ Defined-in (observer-state, concrete-observer) $\wedge$Defined-in(attach, subject) $\wedge$Defined-in (detach ,subject) $\wedge$Defined-in (notify, subject) $\wedge$Defined-in(set-state, concrete-subject) $\wedge$Defined-in (get state, concrete-subject) $\wedge$ Defined-in (update, observer) $\wedge$ Reference-to-one(concrete-observer, concrete-subject) $\wedge$ Reference-to-many(subject, observer) $\wedge$ Inheritance(concrete-subject, subject) $\wedge$ Inheritance(concrete-observer, observer) $\wedge$ Invocation(set-state, notify) $\wedge$ Invocation(notify, update) $\wedge$ Invocation(update, get-state) $\wedge$ Argument(observer, attach) $\wedge$ Argument(observer, detach) $\wedge$ Argument(subject, update)

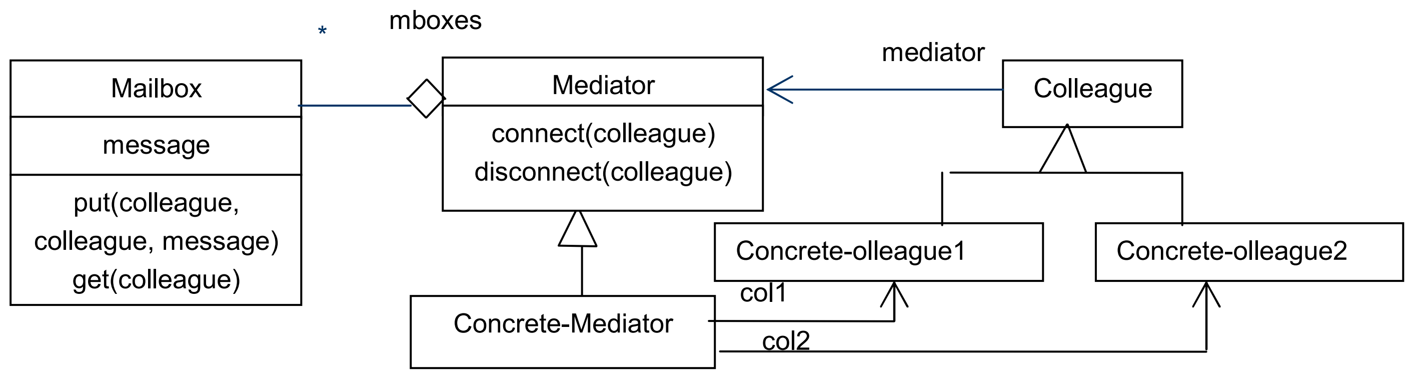
Attached(concrete-subject<0..1>,concrete-observer<*>),Updated(concrete-subject<0..1>,concrete-observer<*>)$\in$ TR;
s $\in$ concrete-subject; o $\in$ concrete-observer; d $\in$ V;

$\text{Init}_{\Psi 1} \triangleq \neg$Attached(s, concrete-observer)
Attach $\triangleq \neg$Attached(s,o)$\wedge$ Attached'(s,o)
Notify $\triangleq$ Attached(s,o)$\wedge$ ((s.subject-state)' =d) $\wedge\neg$Updated'(s,concrete-observer)
Update$\triangleq$ Attached (s,o) $\wedge$ $\neg$Updated(s,o) $\wedge$ ((o.observer-state)' = (s.subject-state)) $\wedge$ Updated'(s,o)
Detach $\triangleq$ Attached(s,o) $\wedge$ $\neg$Attached'(s,o)
M $\triangleq$ Attach $\vee$ Notify $\vee$ Update$\vee$ Detach
u $\triangleq$ $\langle$s,o$\rangle$
$\Psi_1 \triangleq \text{Init}_{\Psi 1} \wedge \Box\, [M]_u \wedge WF_u$ (Update)

Why Temporal Relations

- Temporal relations can be replaced by Boolean expressions made of variables (primed and unprimed) and constant symbols.
- However, this will involve implementation details rather that staying at design level, which is the level of abstraction of patterns.
- In the action *Attach*, instead of using the temporal relation *Attached*, we could use a list handled by a subject (we call this set *List*) and containing the object references of the observers attached to it.
- As such, action *Attach* could be written as: $Attach \triangleq o \notin List \wedge List' = List \cup \{o\}$

Case Study 2:Mediator Pattern



BPSL Specification of Mediator Pattern (Table 4)

$\Phi_2 \equiv \exists$ mediator, concrete-mediator, colleague, concrete-colleague1,concrete-colleague2, mailbox $\in$ C; message $\in$ V; connect, disconnect, put, get $\in$ M:
Defined-in (connect ,mediator) $\wedge$ Defined-in (disconnect, mediator) $\wedge$ Defined-in(message, mailbox) $\wedge$ Defined-in (put ,mailbox) $\wedge$ Defined-in(get, mailbox) $\wedge$ Reference-to-one(colleague, mediator) $\wedge$ Reference-to-one(concrete-mediator, concrete-colleague1) $\wedge$ Reference-to-one(concrete-mediator, concrete-colleague2) $\wedge$ Reference-to-many(mediator, mailbox) $\wedge$ Inheritance(concrete-mediator, mediator) $\wedge$ Inheritance(concrete-colleague1, colleague) $\wedge$ Inheritance(concrete-colleague2, colleague) $\wedge$ Argument(colleague, connect) $\wedge$ Argument(colleague, disconnect) $\wedge$ Argument(colleague, put) $\wedge$ Argument (message, put) $\wedge$ Argument(colleague, get)

Owned (mailbox<0..1>, colleague<0..1>),Connected (colleague<*>,concrete-mediator <0..1>), Accessed (mailbox<*>, colleague<*>) $\in$ TR;
m $\in$ concrete-mediator; $c_1 \in$ concrete-colleague1; $c_2 \in$ concrete-colleague2;mb $\in$ mailbox;d $\in$ V;

$\mathrm{Init}_{\Psi 2} \triangleq \neg$Connected(colleague, m) $\wedge \neg$ Owned(mailbox, colleague)
Connect $\triangleq \neg$Connected(c_1,m) $\wedge$ Connected'(c_1,m)
Acquire $\triangleq$ Connected(c_1,m) $\wedge \neg$ Owned(mb, colleague) $\wedge$ Owned'(mb,c_1)
Release $\triangleq$ Owned(mb,c_1) $\wedge \neg$Accessed(mb, colleague) $\wedge$ $\neg$Owned'(mb,c_1)
Put $\triangleq$ Owned(mb,c_1) $\wedge$Connected(c_2,m) $\wedge$ (mb.message)'=d $\wedge$ Accessed'(mb,c_2)
Get $\triangleq$ Accessed (mb,c_1) $\wedge$ (d'=mb.message) $\wedge$ $\neg$Accessed'(mb,c_1)
Disconnect $\triangleq$ Connected(c_1,m) $\wedge\neg$Owned(mailbox,c_1) $\wedge$ $\neg$Connected'(c_1,m) $\wedge\neg$Accessed'(mailbox,c_1)
N $\triangleq$ Connect $\vee$ Acquire $\vee$ Release $\vee$ Put $\vee$ Get $\vee$ Disconnect
v $\triangleq$ $\langle$mb$\rangle$
$\Psi_2 \triangleq \mathrm{Init}_{\Psi 2} \wedge \Box\, [N]_v \wedge WF_v$ (Get)

Pattern Composition

The correctness of pattern composition is satisfied by both of the following fundamental rules:

- The composition does not make a pattern looses any of its properties.
- The composition does not add new properties to any pattern.

Pattern Composition

- Name mapping is applied during composition of patterns.
- Name mapping associates properties defined in the patterns to be composed with properties defined in the composition.
- For the structural aspect properties represent terms (variables and constants) and predicates, while for the behavioral aspect they represent predicates (temporal relations), variables and actions.

Pattern Composition

- Let P_1 and P_2 denote sets containing properties of the patterns to be composed and let Q denotes a set containing properties of the composition of P_1 with P_2.
- Name mappings are defined as:

$C_1: P_1 \rightarrow Q$

$C_2: P_2 \rightarrow Q$

- The correctness rules define above (informally), can be formalized as follows (*f* represents any property):

(1) $f \in (P_1 \cap P_2) \Rightarrow C_1(f) = C_2(f)$

$f \in ((P_1 \cup P_2) \backslash (P_1 \cap P_2)) \Rightarrow (C_1(f) \in Q) \vee (C_2(f) \in Q)$

(2) $f \notin (P_1 \cup P_2) \Rightarrow (C_1(f) \notin Q) \wedge (C_2(f) \notin Q)$

- Note that (2) means that Q can contain new properties not related to P_1 and P_2.

Pattern Composition

- In FOL, a *substitution* list *Theta* $=\{v_1/t_1,..,v_n/t_n\}$ means to replace all occurrences of variable symbol v_i by terms t_i.
- Substitutions are made from left to right order in the list. For example *subst({x/Pasta,y/John},eats(y,x)) = eats(John, Pasta)*.
- In S_{BPSL}, we restrict the terms t_i to variable and constant symbols only, i.e., function symbols are not supported, as they are not used in BPSL.
- Substitutions can also be applied to TLA actions in a similar way as described above. If A1 is an action involving variables $x_1,\ldots,x_n$, then A_2 an action involving variables $y_1,\ldots,y_n$ can be defined based on A_1 as follows: $A_2 \triangleq$ $subst(\{x_1/y_1,\ldots,x_n/y_n\}, A_1)$.

Pattern Composition

Following are declarations needed for the composition formulas given below:

- P_1 and P_2 are the patterns to be composed and P is the result of the composition.
- $P_i(p,q)$ is a temporal relation of P_1, $Q_j(s,t)$ is a temporal relations of P_2 and $R_k(w,z)$ is a temporal relation of P. The cardinalities have been omitted and $\{p,q,s,t,w,z\} \subset C$.
- Φ is the S_{BPSL} formula of P, Φ_1 is the S_{BPSL} formula of P_1, Φ_2 is the S_{BPSL} formula of P_2 and Φ_3 is an S_{BPSL} formula representing the extra variables, constants and permanent relations of the composition itself.
- Ψ is the B_{BPSL} of P, Ψ_1 is the B_{BPSL} formula of P_1, Ψ_2 is the B_{BPSL} formula of P_2.
- $u_1 \ldots u_m, v_1, \ldots, v_m \in C \cup V \cup M$
- $x_1 \ldots x_n, y_1 \ldots y_n \in (Member\ of\ C) \cup V$

Pattern Composition (Table 5)

$\Phi \equiv subst(\{u_1/v_1, \ldots, u_m/v_m\}, \Phi_1 \wedge \Phi_2) \wedge \Phi_3$	{structural aspect}
$R_k(w,z) \equiv subst(\{p/w, q/z\}, P_i(p,q))$ or	
$R_k(w,z) \equiv subst(\{s/w, t/z\}, Q_j(s,t))$	{temporal relations}
$\Psi \equiv subst(\{x_1/y_1, \ldots, x_n/y_n\}, \Psi_1 \wedge \Psi_2)$	{behavioral aspect}

- The temporal relations of the pattern composition represent the union of the temporal relations of the patterns to be composed.
- Moreover, the substitutions made in formula $\Phi_1 \wedge \Phi_2$ also take effect in the temporal relations of the pattern composition.
- We have shown substitutions in temporal relation here just for clarity.
- In the formula of the pattern composition, we do not explicitly show the substitutions in temporal relations.

Pattern Composition

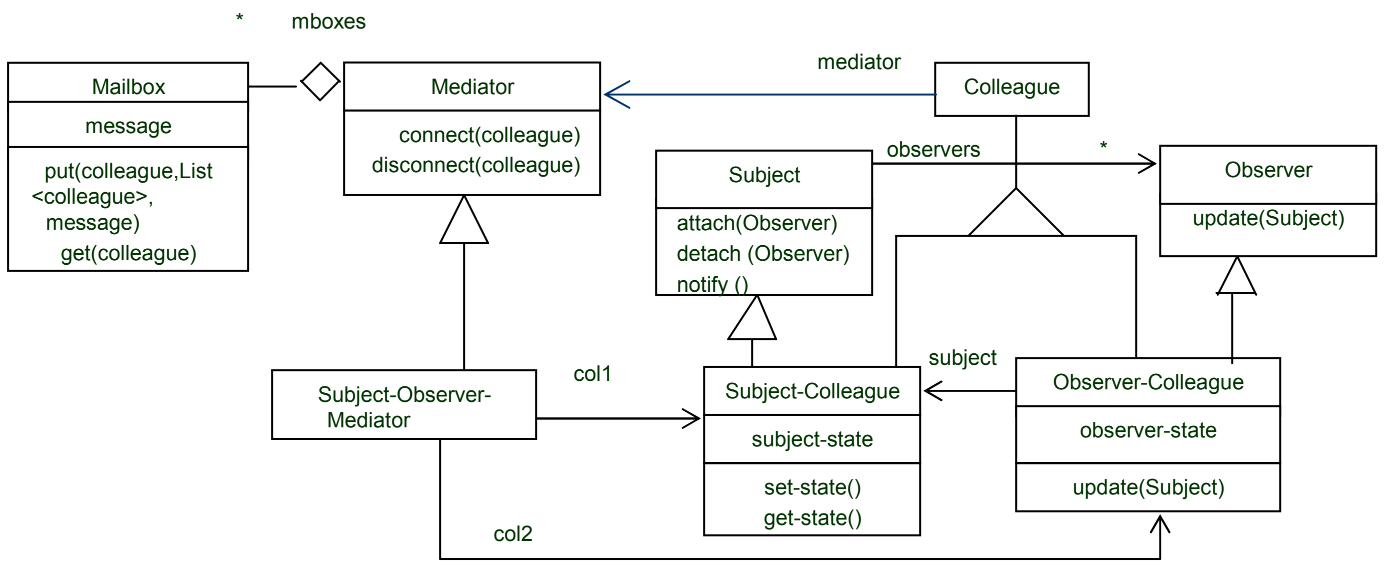
$Init_{\Psi 1} \triangleq P$ {initial predicate}
$M \triangleq A1 \vee \ldots \vee A_{m1}$ {actions}
$u \triangleq \langle u_1, \ldots, u_{m2} \rangle$ {variables}
$\Psi_1 \triangleq Init_{\Psi 1} \wedge \Box[M]u \wedge WF_u(A)$ $\{A \equiv A_{i1} \vee \ldots \vee A_{j1}, 1<=i1<=j1<=m1\}$

$Init_{\Psi 2} \triangleq Q$ {initial predicate}
$N \triangleq B1 \vee \ldots \vee B_{n1}$ {actions}
$v \triangleq \langle v_1, \ldots, v_{n2} \rangle$ {variables}
$\Psi \triangleq Init_{\Psi 2} \wedge \Box[N]_v \wedge WF_v(B)$ $\{B \equiv B_{i2} \vee \ldots \vee B_{j2}, 1<=i2<=j2<=n1\}$

$Init_{\Psi} \triangleq subst(\{\ldots\}, Init_{\Psi 1}) \wedge subst(\{\ldots\}, Init_{\Psi 2})$
$W \triangleq C_1 \vee \ldots \vee C_{m1+n1}$ $\{C_i \equiv subst(\{\ldots\}, A_j)$ or $C_i \equiv subst(\{\ldots\}, B_k)$, $1<=i<=m1+n1, 1<=j<=m1, 1<=k<=n1\}$
$w \triangleq u \cup v$
$\Psi \triangleq Init_{\Psi} \wedge \Box[W]w \wedge WF_w(A) \wedge WF_w(B)$

{…} represents any substitution list. It is to be noted that common permanent relation, initial predicates and actions will only appear once in the composition formulas as by simple logic $P \wedge P \equiv P$ and $P \vee P \equiv P$.

Case Study 3:Observer-Mediator Pattern Composition



BPSL Specification of Observer-Mediator Pattern Composition (Table 6)

$\Phi \equiv$ subst({concrete-mediator/subject-observer-mediator,concrete-colleague1/subject-colleague,concrete-colleague2/observer-colleague, concrete-observer/observer-colleague, concrete-subject/subject-colleague},$\Phi_1 \wedge \Phi_2$)

$\text{Init}_\Psi \triangleq$ subst({s/sc, concrete-observer/observer-colleague},$\text{Init}_{\Psi 1}$) $\wedge$ subst({m/som},$\text{Init}_{\Psi 2}$)

Observer-Attach $\triangleq$ subst({s/sc,o/oc}, Attach)

Subject-Notify $\triangleq$ subst({s/sc,o/oc,concrete-observer/observer-colleague},Notify)

Observer-Update $\triangleq$ subst({s/sc,o/oc},Update)

Observer-Detach $\triangleq$ subst({s/sc,o/oc}, Detach)

Colleague-Connect $\triangleq$ subst ({(c_1/sc)$\vee$(c_1/oc), m/som}, Connect)

Subject-Acquire $\triangleq$ subst({$c_{1/}$sc,m/som},Acquire)

Subject-Release $\triangleq$ subst({c_1/sc},Release)

Subject-Put $\triangleq$ subst({ c_1/sc, c_2/observer-colleague, m/som, d/sc.subject-state},Put)

Observer-Get $\triangleq$ subst({c_1/oc, d/oc.observer-state},Get)

Colleague-Disconnect $\triangleq$ subst({(c_1/sc)$\vee$(c_1/oc), m/som}, Disconnect)

W $\triangleq$ Observer-Attach $\vee$ Subject-Notify $\vee$Observer-Update $\vee$ Observer-Detach $\vee$ Colleague-Connect $\vee$ Subject-Acquire $\vee$ Subject-Release $\vee$ Subject-Putt $\vee$ Observer-Get $\vee$ Colleague-Disconnect

w $\triangleq$ $\langle$ sc,oc,mb$\rangle$

$\Psi \triangleq \text{Init}_\Psi \wedge \Box\, [W]_w \wedge \text{WF}_w$ (Observer-Update) $\wedge$ WF_w(Observer-Get)

Checking The Correctness of Pattern Composition

- The *Observer* pattern has 4 classes, the *Mediato*r pattern has 6 classes, while the *Observer-Mediator* pattern composition has 8 classes.
- This means that there are 2 common classes (which have been mapped to *Observer-Colleague* and *Subject-Colleague*).
- These mappings have been reflected in the substitutions shown in compartment 1 of Table 6 (*concrete-colleague1/observer-colleague, concrete-colleague2/subject-colleague,concrete-observer/observer-colleague,concrete-subject/subject-colleague*).

Checking The Correctness of Pattern Composition

- Moreover all terms and predicates belonging to either *Observer* or *Mediator* patterns have been mapped to themselves (remained unchanged) in the *Observer-Mediator* pattern composition, except the *Concrete-Mediator* class which has been renamed *Subject-Observer-Mediator*.
- Based on the above discussion, condition (1) holds. Since no new terms and/or predicates about either the *Observer* or the *Mediator* pattern have been added by the composition, rule (2) also holds.

Checking The Correctness of Pattern Composition

- As for the behavioral aspect specification, it can be seen that it follows exactly the composition formula shown in Table 5.
- The *Observer* and *Mediator* patterns have no common variables, temporal relations or actions. Temporal relations, initial predicates and actions of the composed pattern are based on those of the original pattern with some straightforward substitutions of variables.

Checking The Correctness of Pattern Composition

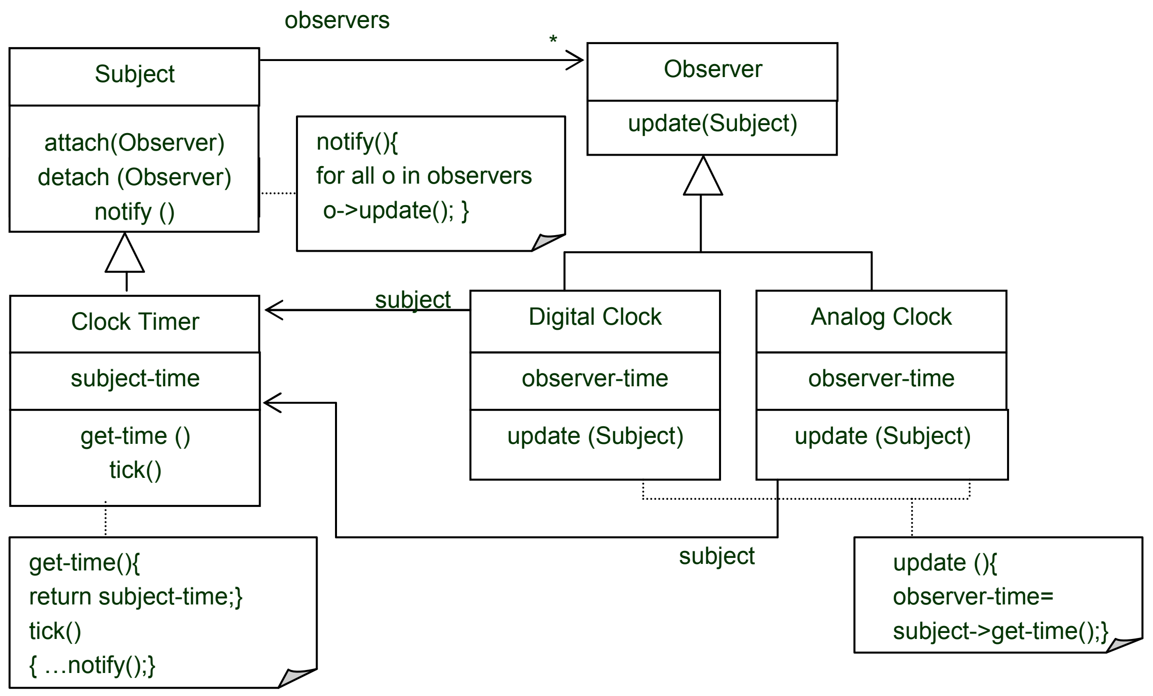
- Hence rule (1) holds. The changes required on actions of the *Observer-Mediator* pattern composition are as follows:
- Only a *Subject Colleague* can own a *mailbox.*
- A *Subject Colleague* can put messages for a group of *Observer Colleagues.*
- Since no new variables, predicates or actions about either the *Observer* or the *Mediator* pattern have been added by the composition, rule (2) also holds.

Formal Specification of Instances of Patterns

- BPSL specification of instances of patterns is done by applying substitutions on the specification of the original pattern. Following are declarations needed for pattern instantiation formulas given below:
- P_1 is a pattern and P is an instance of P_1.
- $Q_i(p,q)$ is a temporal relation of P_1, $R_j(s,t)$ is a temporal relations of P. The cardinalities have been omitted and $\{p,q,s,t\} \subset C$.
- Φ_1 is the S_{BPSL} formula of P_1, Φ is the S_{BPSL} formula of P.
- Ψ_1 is the BBPSL formula of P_1, Ψ is the $B_{BPS}L$ formula of P.
- Φ_2 is an S_{BPSL} formula representing extra required variables and permanent relations for P.
- $u_1 \ldots u_m, v1, \ldots, v_m \in C \cup V \cup M$
- $x_1 \ldots x_n, y_1 \ldots y_n \in (Member\ of\ C) \cup V$

$\Phi \equiv subst(\{u_1/v_1, \ldots, u_n/v_n\}, \Phi_1) \wedge \Phi_2$	*{structural aspect}*
$R_j(s,t) = subst(\{p/s, q/t\}, Q_i(p,q))$	*{temporal relations}*
$\Psi \equiv subst(\{x_1/y_1, \ldots, x_n/y_n\}, \Psi_1)$	*{behavioral aspect}*

Case Study 4: Instance of The Observer Pattern



BPSL Specification of an Instance of The Observer Pattern

$\Phi_2 \equiv \exists$ analog-clock $\in$ C: Inheritance(analog-clock, observer) Φ=subst({ concrete-subject/clock-timer ,concrete-observer/digital-clock, subject-state/subject-time,get-state/get-time,set-state/set-time,observer-state/observer-time}, Φ_1)$\wedge$ Φ_2
Attached(concrete-subject<0..1>,concrete-observer<*>)$\equiv$ subst(concrete-subject/clock-timer, concrete-observer/observer),Updated(concrete-subject<0..1>,concrete-observer<*>)$\equiv$ subst(concrete-subject/clock-timer, concrete-observer/observer)$\in$ TR;
Ψ=subst({concrete-subject/clock-timer, concrete-observer/observer, subject-state/subject-time, observer-state/observer-time}, Ψ_1)

Current Progress with Research Collaboration in Babel Group

- Define actions entirely using temporal relations
- Define temporal relations using variables
- Define instances of patterns, patterns and pattern composition entirely in TLA.

Thank You!!!