ActionGUI semantics

David Basin¹, Manuel Clavel²,³, Marina Egea⁴, Miguel A. García de Dios², and Carolina Dania²,³

¹ ETH Zürich, Switzerland
basin@inf.ethz.ch
² IMDEA Software Institute, Madrid, Spain
[manuel.clavel,miguelangel.garcia,carolina.dania]@imdea.org
³ Universidad Complutense, Madrid, Spain
⁴ Atos Research & Innovation Dept., Madrid, Spain
marina.egea@atosresearch.eu

Abstract. In this technical report we provide the formal account of our ActionGUI methodology, including the semantics of the modeling languages that we use, the definition of our many-models-to-models transformation, and the proof of its correctness.

1 ComponentUML

In this section we first define ComponentUML data models. Then, given a ComponentUML data model D, we define D-object models. Finally, we define the semantics of a ComponentUML data model D as the set Sem(D) of all the D-object models.

Notation. Let TP = \{Integer, Real, String, Boolean\} be the set of ComponentUML primitive data types. In what follows, we denote by \(\llbracket t\rrbracket_{TP}\) the standard carrier set of \(t\), for each primitive data type \(t\) ∈ TP, e.g., \(\llbracket\text{Integer}\rrbracket_{TP} = \mathbb{Z}\).

Let \(A \subseteq \llbracket\text{String}\rrbracket_{TP}\) be the set of all finite strings that only contain letters of the English alphabet.

1.1 ComponentUML data models

Definition. A ComponentUML data model is a tuple \((C, AT, AS, ASO)\) such that:

- \(C \subseteq A\) is a set of class identifiers.
- \(AT\) is a set of triples \((at, c, t)\), also represented as \(at(c, t)\), where \(at\) ∈ \(A\) is an attribute identifier, \(c \in C\), \(t \in C \cup TP\), and \(c\) and \(t\) are, respectively, the class and the type of the attribute \(at\).
- \(AS\) is a set of triples \((as, c, c')\), also denoted as \(as(c, c')\), where \(as\) ∈ \(A\) is an association-end identifier, \(c, c' \in C\), and \(c\) and \(c'\) are, respectively, the source and the target classes of \(as\).
- \(ASO\) is a symmetric relation, \(ASO \subseteq AS \times AS\), where \((as, c, c')\), \(as'(c, c')\) ∈ \(ASO\) represents that \(as'\) is the association-end opposite to \(as\), and vice versa, and \(c, c' \in C\).
Invariants.

- There is no class whose identifier also belongs to TP.
- Attributes and associations-ends of the same class always have different identifiers.
- Every association-end is related with exactly another association-end. That is, for every tuple \((as, c, c')\) in \(AS\), there exists exactly one other tuple \((as', c', c)\) in \(AS\) such that \((as, c, c') (as', c', c)\) in \(ASO\).

1.2 ComponentUML object models

Definition. Let \(D\) be a ComponentUML data model \(\langle C, AT, AS, ASO \rangle\). Then, a \(D\)-object model is a tuple \(\langle O, VA, LK \rangle\), such that:

- \(O\) is a set of pairs \(\langle o, c \rangle\), where \(o \in A\) is an object identifier and \(c \in C\). Each pair \(\langle o, c \rangle\), also represented as \(o c\), denotes that the object \(o\) is of the class \(c\).
- \(VA\) is a set of triples \(\langle o c, at(c, t), va \rangle\), where \(at(c, t) \in AT\), \(o c \in O\), \(t \in TP\), and \(va \in \{T\}\) is a value of type \(t\). Each triple \(\langle o c, at(c, t), va \rangle\) denotes that \(va\) is the value of the attribute \(at\) of the object \(o\).
- \(LK\) is a set of triples \(\langle o c, as(c, c'), o_c' \rangle\), where \(as(c, c') \in AS\), and \(o c, o_c' \in O\). Each tuple \(\langle o c, as(c, c'), o_c' \rangle\) denotes that the object \(o'\) is among the objects that are linked to the object \(o\) through the association-end \(as\).

Invariants.

- There are no two different values for the same attribute of the same object. (However, it is not necessary that every attribute of an object has a value.)
- For every association-end \(as(c, c')\) in \(AS\), such that \(as(c, c'), as'(c, c')\) in \(ASO\), if there is a link \(\langle o_c, as(c, c'), o_c' \rangle\) in \(LK\) between two objects \(o_c\) and \(o_c'\) through this association-end, then there is also a link \(\langle o_c', as'(c, c'), o_c \rangle\) in \(LK\) between these two objects through the opposite association-end.

1.3 Semantics of ComponentUML data models

Definition. Let \(D\) be a ComponentUML data model. The semantics of \(D\), denoted by \(Sem(D)\), is the set of all the ComponentUML \(D\)-object models.

2 SecureUML

In this section we first define SecureUML data actions and SecureUML authorization constraints, both relative to a given ComponentUML data model. Then, we define SecureUML security models, also relative to a given ComponentUML model. Next, given a SecureUML security model \(S\), we define \(S\)-authorized actions. Finally, we define the semantics of a SecureUML security model as the set \(Sem(S)\) of all the \(S\)-authorized actions and, based on this definition, we define the notion of a consistent SecureUML security model.
Notation. Let \( D \) be a data model, and let \( I \) be a \( D \)-object model, \( I = \langle O, VA, LK \rangle \). In what follows, we will use the following notation:

- We denote by \( \text{Typ}(D) \) the set of all the OCL types, given the classes declared in \( D \). These types are defined in the OCL standard [1].
- We denote by \( \text{Expr}(D) \) the set of all the OCL expressions that have \( D \) as their contextual model. These expressions are defined in the OCL standard [1].
- Note that, by definition, they do not contain free variables.
- We denote by \( \text{Expr}(D, X) \) the set of all the OCL expressions that have \( D \) as their contextual model but that may also contain variables in \( X \) as free variables. Moreover, for every \( expr \in \text{Expr}(D, X) \), we denote by \( \text{FVar}(expr) \subseteq X \) the set of all the free variables contained in \( expr \).
- We denote by \( \text{Expr}(D_I) \) the set of all the OCL expressions that have \( D \) as their contextual model and may also contain as constants the objects \( o_c \in O \).
- Let \( expr \) be an OCL expression in \( \text{Expr}(D_I) \). Then, we denote by \( \llbracket expr \rrbracket^I \) the evaluation of the expression \( expr \) in the object model \( I \), as defined in the OCL standard [1]. Note that the evaluation of an OCL expression always return a \emph{literal expression} in \( \text{Expr}(D_I) \), which can not be further reduced and which does not contain any variables.
- We denote by \( \text{Typ}(D) \) the set of all the OCL types, given the classes declared in \( D \). These types are defined in the OCL standard [1].

2.1 SecureUML data actions

Definition. Let \( D \) be a data model \( \langle C, AT, AS, ASO \rangle \). Then, we denote by \( \text{Act}(D) \) the set of all the \( (\text{atomic}) \) data actions that can be executed on \( D \)-object models. \( \text{Act}(D) \) is defined as follows: for every class \( c \in C \), every attribute \( at_{(c,t)} \in AT \), and every association-end \( as_{(c,c')} \in AS \),

\[
\text{Create}(c), \text{Delete}(c) \in \text{Act}(D),
\]

\[
\text{Read}(at_{(c,t)}), \text{Update}(at_{(c,t)}) \in \text{Act}(D),
\]

\[
\text{Read}(as_{(c,c')}), \text{Create}(as_{(c,c')}), \text{Delete}(as_{(c,c')}) \in \text{Act}(D).
\]

The notation \( \text{action}(\text{resource}) \) reflects that data actions are always upon resources.
2.2 SecureUML authorization constraints

Definition. Let \( D \) be a data model \( \langle C, AT, AS, ASO \rangle \). Let \( u \in C \) be a class that represents the users. Let \( act \in Act(D) \) be a data action. Then, we denote by \( \text{AuthExpr}(D, u, act) \) the set of all the authorization constraints that can be imposed on users of type \( u \) for executing the data action \( act \) with respect to the data model \( D \). Informally, an authorization constraint is an (extended) OCL expression that may contain distinguished keywords (logically interpreted as free variables) that refer to the user attempting to execute the action (\( \text{caller} \)), to the data upon which the action is to be executed (\( \text{self} \)), or to the data that the action takes as its arguments (\( \text{value} \) and \( \text{target} \)).

More formally, \( \text{AuthExpr}(D, u, act) \) is defined as follows: for every class \( c \in C \), every attribute \( at_{(c,t)} \in AT \), and every association-end \( as_{(c,c')} \in AS \):

\[
\begin{align*}
\text{AuthExpr}(D, u, \text{Create}(c)) &= \text{Expr}(D, \{\text{caller}_u\}). \\
\text{AuthExpr}(D, u, \text{Delete}(c)) &= \text{Expr}(D, \{\text{self}_c, \text{caller}_u\}). \\
\text{AuthExpr}(D, u, \text{Read}(at_{(c,t)})) &= \text{Expr}(D, \{\text{self}_c, \text{caller}_u\}). \\
\text{AuthExpr}(D, u, \text{Update}(at_{(c,t)})) &= \text{Expr}(D, \{\text{self}_c, \text{value}_u, \text{caller}_u\}). \\
\text{AuthExpr}(D, u, \text{Read}(as_{(c,c')})) &= \text{Expr}(D, \{\text{self}_c, \text{caller}_u\}). \\
\text{AuthExpr}(D, u, \text{Create}(as_{(c,c')})) &= \text{Expr}(D, \{\text{self}_c, \text{target}_{c'}, \text{caller}_u\}). \\
\text{AuthExpr}(D, u, \text{Delete}(as_{(c,c')})) &= \text{Expr}(D, \{\text{self}_c, \text{target}_{c'}, \text{caller}_u\}).
\end{align*}
\]

2.3 SecureUML security models

Definition. Let \( D \) be a data model \( \langle C, AT, AS, ASO \rangle \). Then, a SecureUML \( D \)-security model \( S \) is a tuple \( \langle D, R, RH, u, P \rangle \) such that:

\(- \ R \subset A \) is a set of role identifiers.
\(- \ RH \subset R \times R \) is a partial order representing the role hierarchy.
\(- \ u \in C \) is a class that represents the users.
\(- \ P \) is a set of triples \( \langle r, act, expr \rangle \) representing permissions: namely, that the role \( r \in R \) is granted permission for the action \( act \in Act(D) \) provided the constraint \( expr \in \text{AuthExpr}(D, u, act) \) is satisfied.

Definition. Let \( D \) be a data model \( \langle C, AT, AS, ASO \rangle \). Let \( S \) be a \( D \)-security model \( \langle D, R, RH, u, P \rangle \). Then, \( \text{AuthPerm}(S, r, act) \) is the disjunction of all the authorization constraints controlling the access for users in the role \( r \) to execute the action \( act \), according to \( S \). \( \text{AuthPerm}(S, r, act) \) is defined as follows: Let \( Q = \{ expr | \exists r' \in R. \langle r', act, expr \rangle \in P \land (r, r') \in RH \} \). Then,

\[
\text{AuthPerm}(S, r, act) = \begin{cases} 
expr_1 \lor \ldots \lor expr_n, & \text{if } Q = \{ expr_1, \ldots, expr_n \}, \\
\text{false}, & \text{if } Q = \emptyset.
\end{cases}
\]

Note that, by definition, \( \text{AuthPerm}(S, r, act) \in \text{Expr}(D, X) \), where \( X \) is the set containing \( \text{caller}_u \) plus the appropriate instances of \( \text{self}, \text{target}, \) and \( \text{value} \), depending on the type of the action \( act \).
2.4 SecureUML authorized actions

Definition. Let $D$ be a data model $\langle C, AT, AS, ASO \rangle$. Let $S$ be a $D$-security model $\langle D, R, RH, u, P \rangle$. Let $I$ be a $D$-object model $\langle O, VA, LK \rangle$. Let $o_u \in O$ be a user, $r \in R$ be a role, and $act \in \text{Act}(D)$ be a $D$-data action. Moreover, let $\theta$ be a $(\text{FVar}(\text{AuthPerm}(S, r, act)), I)$-substitution.

Then, $\langle I, o_u, r, act, \theta \rangle$ is an $S$-authorized action if and only if

$$[[\text{AuthPerm}(S, r, act)(\theta \oplus \{\text{caller} \mapsto o_u\})]]^I = \text{true}.$$ 

Note that, given our definition of AuthPerm,

– No permission is granted for executing an action, unless it is explicitly declared.
– All permissions are inherited along the role hierarchy.

2.5 Semantics of SecureUML security models

Definition. Let $D$ be a data model and let $S$ be a $D$-security model. Then, the semantics of $S$, given by $\text{Sem}(S)$, is the set of all the $S$-authorized actions.

3 GUIML

In this section we first define GUIML layout models, which simply model graphical user interfaces without considering their behaviors. Then, we define GUIML statements, which specify sequences of actions that are possibly conditional and iterated. Next, we define GUIML behavioral models, which are GUIML layout models but also with associated behavior, i.e., with statements associated to each of the widget events. Finally, we define a set of inference rules that will provide the (operational) semantics of GUIML events as the set $\text{Sem}(G, ev)$ of all the transitions defined by these rules.

Notation. In what follows, let $ET$ be the set of GUIML event types,

$$ET = \{\text{onClick, onCreate}\}.$$ 

Also, let $WT$ be the set of GUIML widget types,

$$WT = \{\text{Window, Table, Combo-box, Button, Text field, Label, Boolean check}\}.$$ 

3.1 GUIML layout models

Definition. A GUIML layout model $H$ is a tuple $\langle W, WC, X, EV \rangle$ such that:

– $W$ is a set of pairs $\langle w, wt \rangle$, also represented as $w_{wt}$, where $w \in A$ is a widget identifier, and $wt \in WT$ is the widget’s type.
$WC \subset W \times W$ is a relation representing the widget containment.

$X$ is a set of pairs $\langle (x, t), (w, wt) \rangle$, called widget variables, also represented as $\langle x_t, w_{wt} \rangle$, where $x$ is a variable identifier, $t \in \text{Typ}(D)$ is the variable’s type, and $(w, wt) \in W$ is the widget that owns this variable.

$EV$ is a set of pairs $\langle ev, (w, wt) \rangle$, also represented as $\langle ev_{wt} \rangle$, where $ev \in \text{ET}$ is an event type and $(w, wt) \in W$ is the widget that supports this event type.

**Invariants.**

- The containment relation $WC$ defines a set of rooted trees. Moreover, at the root of every tree in $WC$ there is a widget of type Window and, conversely, every widget in $W$ of type Window is the root of a tree in $WC$.
- There are no two variables owned by the same widget with the same identifier.
- If two widgets are directly contained in the same widget, then they have different identifiers.

**Notation.** Let $H$ be a GUIML layout model $\langle W, WC, X, EV \rangle$. In what follows we will use the following notation:

- We denote by $WC^+$ the transitive closure of the containment relation defined in $WC$.
- Let $w_{wt} \in W$ be a window in $W$, $wt \neq \text{Window}$. Then, we denote by $\text{Win}(H, w_{wt})$ the window that contains $w_{wt}$ in $W$, i.e., $(w_{wt}, \text{Win}(H, w_{wt})) \in WC^+$.
- Let $w_{wt} \in W$ be a window in $W$. Then, we denote by $\text{Var}(H, w_{wt})$ the set of variables in $X$ that are owned by $w_{wt}$, i.e., $\text{Var}(H, w_{wt}) = \{ \langle x_t, w'_{wt} \rangle \mid \langle x_t, w'_{wt} \rangle \in X \wedge w_{wt} = w'_{wt} \}$.
- Let $w_{wt} \in W$ be a window in $W$. Then, we denote by $\text{Var}^\sharp(H, w_{wt})$ the set of the variables in $X$ that are visible from $w_{wt}$. $\text{Var}^\sharp(H, w_{wt})$ is defined as follows:

$$\text{Var}^\sharp(H, w_{wt}) = \text{Var}(H, w_{wt}) \cup \{ \langle x_t, w'_{wt} \rangle \mid \langle x_t, w'_{wt} \rangle \in \text{Var}(H, w'_{wt}) \wedge (w'_{wt}, \text{Win}(H, w_{wt})) \in WC^+ \}.$$  

Note that, by definition, if two widgets are contained in the same window, then their sets of visible variables are identical. Also, the set of visible variables of a widget is the same than the set of visible variables of the widget’s containing window.

### 3.2 GUIML statements

**Definition.** Let $D$ be a data model $\langle C, AT, AS, ASO \rangle$. Let $H$ be a GUIML layout model $\langle W, WC, X, EV \rangle$. Let $w_{wt}$ be a window in $H$, i.e., $w_{\text{Window}} \in W$. Then, we denote by $\text{Stm}(D, H, w)$ the set of all the statements that can be written in the context of the window $w$. This set is inductively defined as follows:
Base case (data actions): The building block for statements are the data actions along with the GUI actions. The GUIML data actions are the SecureUML data actions introduced before, except that they now take additional arguments that, depending on the action’s type, either specify, using OCL (extended with widget variables), the object $self$ upon which the action is to be executed, or the $value$ and $target$ of this action, or indicate the widget $variable$ where the action’s outcome is to be stored. To reflect this difference between the GUIML data actions and their corresponding SecureUML data actions, we use the notation $action(resource)[arguments]$ for GUIML data actions. Thus, if $action(resource)[arguments]$ is a GUIML data action, then $action(resource)$ is its corresponding SecureML data action.

- For every entity create action $Create(c) \in \text{Act}(D)$ and every variable of type $c$ in $\text{Var}^d(H,w_{wt})$, then
  
  $Create(c)[variable] \in \text{Stm}(D,H,w)$.  

- For every entity delete action $Delete(c) \in \text{Act}(D)$ and every expression $self$ of type $c$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, then
  
  $Delete(c)[self] \in \text{Stm}(D,H,w)$.  

- For every attribute read action $Read(at(c,t)) \in \text{Act}(D)$, every expression $self$ of type $c$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, and every widget variable of type $t$ in $\text{Var}^d(H,w_{wt})$, then
  
  $Read(at(c,t))[self, variable] \in \text{Stm}(D,H,w)$.  

- For every attribute update action $Update(at(c,t)) \in \text{Act}(D)$, every expression $self$ of type $c$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, and every expression $value$ of type $t$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, then
  
  $Update(at(c,t))[self, value] \in \text{Stm}(D,H,w)$.  

- For every association-end read action $Read(as(c,c')) \in \text{Act}(D)$, every expression $self$ of type $c$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, and every variable of type $\text{Set}(c')$ in $\text{Var}^d(H,w_{wt})$, then
  
  $Read(as(c,c'))[self, variable] \in \text{Stm}(D,H,w)$.  

- For every association-end create action $Create(as(c,c')) \in \text{Act}(D)$, every expression $self$ of type $c$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, and every expression $target$ of type $c'$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, then
  
  $Create(as(c,c'))[self, target] \in \text{Stm}(D,H,w)$.  

- For every association-end delete action $Delete(as(c,c')) \in \text{Act}(D)$, every expression $self$ of type $c$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, and every expression $target$ of type $c'$ in $\text{Expr}(D,\text{Var}^d(H,w_{wt}))$, then
  
  $Delete(as(c,c'))[self, target] \in \text{Stm}(D,H,w)$.  

Base case (GUI actions):

- For every type \( t \in \text{Typ}(D) \), every variable of type \( t \) in \( \text{Var}^\#(H, w_{wt}) \) and every expression value of type \( t \) in \( \text{Expr}(D, \text{Var}^\#(H, w_{wt})) \), then
  \[
  \text{Set}[\text{variable}, \text{value}] \in \text{Stm}(D, H, w).
  \]

- For every window \( \langle w', \text{Window} \rangle \in W \), every list of variables \( \text{variable}_1, \ldots, \text{variable}_n \), such that, for \( 1 \leq i \leq n \), \( \text{variable}_i \) is of type \( t_i \) in \( \text{Var}(H, w'_{\text{Window}}) \), and every list of expressions \( \text{value}_1, \ldots, \text{value}_n \), such that, for \( 1 \leq i \leq n \), \( \text{value}_i \) is of type \( t_i \) in \( \text{Expr}(D, \text{Var}^\#(H, w_{wt})) \), then
  \[
  \text{Open}[w', (\text{variable}, \text{value})] \in \text{Stm}(D, H, w).
  \]

- Finally,
  \[
  \text{Back}, \text{Fail}, \text{Skip} \in \text{Stm}(D, H, w).
  \]

Inductive case

- For every expression \( \text{cond} \) of type \( \text{Boolean} \) in \( \text{Expr}(D, \text{Var}^\#(H, w_{wt})) \), and every statements \( \text{stm}_1, \text{stm}_2 \in \text{Stm}(D, H, w) \), then
  \[
  \text{if,then,else}[\text{cond}, \text{stm}_1, \text{stm}_2] \in \text{Stm}(D, H, w).
  \]

- For every expression \( \text{source} \) of type \( \text{Sequence}(t) \) in \( \text{Expr}(D, \text{Var}^\#(H, w_{wt})) \), every widget variable \( \text{variable} \) of type \( t \) in \( \text{Var}^\#(H, w_{wt}) \), and every statement \( \text{body} \in \text{Stm}(D, H, w) \), then
  \[
  \text{iterator}[\text{source}, \text{variable}, \text{body}] \in \text{Stm}(D, H, w).
  \]

- For all statements \( \text{stm}, \text{stm}' \in \text{Stm}(D, H, w) \), then
  \[
  \text{stm} ; \text{stm}' \in \text{Stm}(D, H, w).
  \]

3.3 Behavioral GUIML models

Definition. Let \( D \) be a data model \( \langle C, AT, AS, ASO \rangle \). Let \( H \) be a GUIML layout model \( \langle W, WC, X, EV \rangle \). Then, a GUIML behavioral model \( G \) is a tuple \( \langle D, H, \text{EST} \rangle \) such that:

- \( \text{EST} \) is a set of pairs \( \langle \text{ev}, w_{wt} \rangle, \text{stm} \rangle \), where
  - \( \langle \text{ev}, w_{wt} \rangle \in \text{EV} \) is an event.
  - \( \text{stm} \in \text{Stm}(D, H, \text{Win}(H, w_{wt})) \) is the statement associated to this event.

Invariants.

- Every event is associated with exactly one statement.
- In every sequence of statement associated to an event, the GUI actions \( \text{Open} \) and \( \text{Back} \) can only appear (if at all) at the last position.\footnote{When this last position is occupied by an if-then-else, then \( \text{Open} \) and \( \text{Back} \) can only appear (if at all) at the last position of its then- or else-branches (and recursively in the case of nested if-then-elses). The situation is similar for iterator statements.}
3.4 Operational semantics for events

**Notation.** Let $D$ be a data model $\langle C, AT, AS, ASO \rangle$. Let $I$ be a $D$-object model $\langle O, VA, LK \rangle$. In what follows we will use the following notation:

- Let $o_c \in O$ be an object. Then, $(VA \setminus o_c)$ denotes the set that results from deleting from $VA$ every triple that contains $o_c$. That is, $(VA \setminus o_c) = \{ \langle o_c', at(o_c', v_a) \rangle \mid \langle o_c', at(o_c', v_a) \rangle \in VA \land o_c' \neq o_c \}$.

- Let $o_c \in O$ be an object. Then, $(LK \setminus o_c)$ denotes the set that results from deleting from $LK$ every triple that contains $o_c$. That is, $(LK \setminus o_c) = \{ \langle o_c', as(o_c', c), o_c'' \rangle \mid \langle o_c', as(o_c', c), o_c'' \rangle \in LK \land c, c' \neq o_c \land o_c'' \neq o_c \}$.

- Let $at(o_c, t) \in AT$ be an attribute. Let $o_c \in O$ be an object and let $va \in \Gamma^m_{TP}$ be a value of type $t$. Then $VA \oplus \{o_c, at(o_c, t), va\}$ denotes the set that results from overriding (i.e., updating) in $VA$ the value of the attribute $at$ of the object $o_c$ with $va$. That is, $(VA \oplus \{o_c, at(o_c, t), va\}) = \{ \langle o_c, at(o_c, t), va\rangle \cup \{ \langle o_c', at(o_c', t), va\rangle \mid \langle o_c', at(o_c', t), va\rangle \in VA \land o_c', t \neq o_c \land at(o_c', t) \neq at(o_c, t) \}$.

**Definition.** Let $D$ be a data model $\langle C, AT, AS, ASO \rangle$. Let $H$ be a GUIML layout model $\langle W, WC, X, EV \rangle$. Let $G$ be a GUIML behavioral model $\langle D, H, EST \rangle$. Let $ev \in EV$ be an event in $G$ with $\langle ev, stm \rangle \in EST$. Then, $\text{Sem}(G, ev)$ is the set of all the transitions

$$\langle stm, I, \theta \rangle \longrightarrow^* \langle \text{Skip}, I', \theta' \rangle$$

where $\longrightarrow^*$ is the transitive closure of the small-step transition relation $\longrightarrow$ defined by the following inference rules. For every $D$-object model $I = \langle O, VA, LK \rangle$ and every $(X, I)$-substitution we have:

**Base case (data actions)**

$$o_c \notin O$$

$$\langle \text{Create}(c)[\text{variable}], I, \theta \rangle \longrightarrow \langle \text{Skip}, \{O \cup \{o_c\}, VA, LK\}, \theta \oplus \{\text{variable} \mapsto o_c\} \rangle.$$

$$\| (\text{self}){\theta} \|^I = o$$

$$\langle \text{Delete}(c)[\text{self}], I, \theta \rangle \longrightarrow \langle \text{Skip}, \{\{O \setminus o_c\}, (VA \setminus o_c), (LK \setminus o_c)\}, \theta \rangle.$$

$$\| (\text{self}.at){\theta} \|^I = va$$

$$\langle \text{Read}(at(o_c, t))[\text{self}, \text{variable}], I, \theta \rangle \longrightarrow \langle \text{Skip}, I, \theta \oplus \{\text{variable} \mapsto va\} \rangle.$$

$$\| (\text{self}){\theta} \|^I = o, \| (\text{value}){\theta} \|^I = va$$

$$\langle \text{Update}(at(o_c, t))[\text{self}, \text{value}], I, \theta \rangle \longrightarrow \langle \text{Skip}, \{O, (VA \oplus \{o, at, va\}), LK\}, \theta \rangle.$$

$$\| (\text{self}.as){\theta} \|^I = \{o_1, \ldots, o_n\}$$

$$\langle \text{Read}(as(o_c, t'))[\text{self}, \text{variable}], I, \theta \rangle \longrightarrow \langle \text{Skip}, I, \theta \oplus \{\text{variable} \mapsto \{o_1, \ldots, o_n\}\} \rangle.$$
\[ \overline{(self)}^{I} = o, \overline{(target)}^{I} = o', (as_{(c,c')}, as'_{(c',c)}) \in ASO \]

\[ \langle \text{Create}(as_{(c,c')}) | (self, target), I, \theta \rangle \rightarrow \langle \text{Skip}, (O, VA, (LK \cup \{(o, as, o'), (o', as', o')\})), \theta \rangle \]

\[ \overline{(self)}^{I} = o, \overline{(target)}^{I} = o', (as_{(c,c')}, as'_{(c',c)}) \in ASO \]

\[ \langle \text{Delete}(as_{(c,c')}) | (self, target), I, \theta \rangle \rightarrow \langle \text{Skip}, (O, VA, (LK \setminus \{(o, as, o'), (o', as', o')\})), \theta \rangle \]

**Base case (GUI actions)**

\[ \overline{(value)}^{I} = va \]

\[ \langle \text{Set}[\text{variable}, value], I, \theta \rangle \rightarrow \langle \text{Skip}, I, \theta \oplus \{\text{variable} \mapsto \text{va}\} \rangle \]

\[ \langle \text{Open}[\langle w, \text{Window} \rangle, (\text{variable}, \text{value})], I, \theta \rangle \rightarrow \langle \text{Skip}, I, \theta \rangle \]

\[ \langle \text{Back}, I, \theta \rangle \rightarrow \langle \text{Skip}, I, \theta \rangle \]

**Inductive case**

\[ \overline{(cond)}^{I} = \text{true} \]

\[ \langle \text{if_then_else}[\text{cond}, \text{stm}_1, \text{stm}_2], I, \theta \rangle \rightarrow \langle \text{stm}_1, I, \theta \rangle \]

\[ \overline{(cond)}^{I} = \text{false} \]

\[ \langle \text{if_then_else}[\text{cond}, \text{stm}_1, \text{stm}_2], I, \theta \rangle \rightarrow \langle \text{stm}_2, I, \theta \rangle \]

\[ \overline{(source)}^{I} = [v_1, \ldots, v_n] \]

\[ \langle \text{Iterator}[\text{source}, \text{variable}, \text{body}], I, \theta \rangle \rightarrow \langle (\text{Set}(\text{variable}, v_1); \text{body}; \ldots; \text{Set}(\text{variable}, v_n); \text{body}), I, \theta \rangle \]

\[ \langle \text{stm}_1, I, \theta \rangle \rightarrow \langle \text{stm}'_1, I', \theta' \rangle \]

\[ \langle (\text{stm}_1; \text{stm}_2), I, \theta \rangle \rightarrow \langle (\text{stm}'_1; \text{stm}_2), I', \theta' \rangle \]

\[ \langle (\text{Skip}; \text{stm}_2), I, \theta \rangle \rightarrow \langle \text{stm}_2, I, \theta \rangle \]
4 Security-aware GUIML

In this section we first characterize security-awareness of GUIML behavior models in terms of a transition relation defined by a security-aware version of the inference rules that define the (non security-aware) operational semantics of GUIML events. Then, we define a model transformation that, given a GUIML model $G$ and a SecureUML model $S$, generates a new GUIML model that is security aware with respect to $S$. Finally, we formalize and prove the correctness of our model transformation.

4.1 Operational semantics for security-aware events

Informally, security-aware events are those events where the execution of the associated actions are conditional on the satisfaction of the corresponding authorization constraints. However, which constraint these are depends, of course, on the role of the actual user who triggers this event. Thus, in order to be able to refer to the user’s role (when specifying the aforementioned conditions within the statement associated to the event), we will explicitly require that:

- The underlying data model $D$ includes a class `Role`, with an attribute `name` of type `String`.
- Every window in the GUIML model owns two distinguished variables, `caller` (of the same type than the users) and `role` (of type `Role`), whose intended values are, respectively, the actual user and its role.

Moreover, when discussing security-awareness with respect to a security model $S$ we will be interested only in $D$-object models whose objects of type `Role` are conformant with the roles declared in $S$, in the following sense: Let $D$ be a data model $⟨C, AT, AS, ASO⟩$, such that `Role` $∈ C$ and $⟨\text{name}, \text{Role}, \text{String}⟩ ∈ AT$. Let $S$ be a $D$-security model $⟨D, R, RH, u, P⟩$. Let $I = ⟨O, VA, LK⟩$ be a $D$-object model. Then, we say that $I$ is $R$-conformant if and only if

$$R = \{va | ⟨\text{role}_O, \text{name}\{\text{Role}, \text{String}\}, va⟩ ∈ VA\}.$$

**Definition.** Let $D$ be a data model $⟨C, AT, AS, ASO⟩$, such that `Role` $∈ C$ and $⟨\text{name}, \text{Role}, \text{String}⟩ ∈ AT$. Let $S$ be a $D$-security model $⟨D, R, RH, u, P⟩$. Let $H$ be a GUIML layout model $⟨W, WC, X, EV⟩$ such that, for every $w_{\text{Window}} ∈ W$, $⟨\text{role}_{\text{Role}}, w_{\text{Window}}⟩ ∈ X$ and $⟨\text{caller}_u, w_{\text{Window}}⟩ ∈ X$. Let $G$ be a GUIML behavioral model $⟨D, H, EST⟩$. Let $ev ∈ EV$ be an event in $G$ whose associated statement is $stm$, i.e., $⟨ev, stm⟩ ∈ EST$. Then, the security-aware operational semantics for the event $ev$ is given by the set of all the transitions

$$⟨stm, I, θ⟩ \xrightarrow{*_{sec}} ⟨\text{Skip}, I', θ'⟩$$

such that $I$ is $R$-conformant and $\xrightarrow{*_{sec}}$ is the transitive closure of the small-step transition relation $\xrightarrow{\text{sec}}$ defined by the security-aware versions of the inference rules that define the operational semantics of GUIML events. Formally, for every
GUIML data action $\text{act}[	ext{arg}]$, the security-aware version of the corresponding inference rule includes the following additional condition:

$$[(\text{AuthPerm}(S, [(\text{role}_\text{Win}(H, ev).\text{name})\theta])^I, \text{act})(\text{Subst}(\text{act}[	ext{arg}])))\theta]^I = \text{true},$$

where $\text{Subst}(\text{act}[	ext{arg}])$ is the substitution defined below, which depends on the type of the action $\text{act}$.

- $\text{Subst}(\text{Create}(c)[\text{variable}]) = \{\text{caller}_u \mapsto (\text{caller}_u, \text{w}_\text{Window})\}$
- $\text{Subst}(\text{Delete}(c)[\text{self}]) = \{\text{caller}_u \mapsto (\text{caller}_u, \text{w}_\text{Window}), \text{self}_c \mapsto \text{self}\}$
- $\text{Subst}(\text{Read}(\text{at}(c,t))[\text{self}, \text{variable}]) = \{\text{caller}_u \mapsto (\text{caller}_u, \text{w}_\text{Window}), \text{self}_c \mapsto \text{self}\}$
- $\text{Subst}(\text{Update}(\text{at}(c,t))[\text{self}, \text{value}]) = \{\text{caller}_u \mapsto (\text{caller}_u, \text{w}_\text{Window}), \text{self}_c \mapsto \text{self}, \text{value}_t \mapsto \text{value}\}$
- $\text{Subst}(\text{Read}(\text{as}(c,c'))[\text{self}, \text{variable}]) = \{\text{caller}_u \mapsto (\text{caller}_u, \text{w}_\text{Window}), \text{self}_c \mapsto \text{object}\}$
- $\text{Subst}(\text{Create}(\text{as}(c,c'))[\text{self}, \text{target}]) = \{\text{caller}_u \mapsto (\text{caller}_u, \text{w}_\text{Window}), \text{self}_c \mapsto \text{self}, \text{target}_c \mapsto \text{target}\}$
- $\text{Subst}(\text{Delete}(\text{as}(c,c'))[\text{self}, \text{target}]) = \{\text{caller}_u \mapsto (\text{caller}_u, \text{w}_\text{Window}), \text{self}_c \mapsto \text{self}, \text{target}_c \mapsto \text{target}\}$

The inference rules for GUI actions are not modified in their security-aware versions. The inference rules for if-then-else statements, iterator statements, or sequences of statements also remain unmodified.

### 4.2 Security-aware model transformation

**Definition.** Let $D$ be a data model $\langle C, AT, AS, ASO \rangle$, such that Role $\in C$ and $\langle \text{name}, \text{Role}, \text{String} \rangle \in AT$. Let $S$ be a $D$-security model $\langle D, R, RH, u, P \rangle$. Let $H$ be a GUIML layout model $\langle W, WC, X, EV \rangle$ such that, for every $\text{w}_\text{Window} \in W$, $\langle \text{role}_\text{Role}, \text{w}_\text{Window} \rangle \in X$ and $\langle \text{caller}_u, \text{w}_\text{Window} \rangle \in X$. Let $G$ be a GUIML behavioral model $\langle D, H, EST \rangle$. Then, Sec$(G, S)$ is the $S$-security-aware version of $G$, defined as follows:

$$\text{Sec}(G, S) = \langle D, H, \{\langle ev, \text{Sec}(stm, S) \rangle \mid \langle ev, stm \rangle \in EST \} \rangle.$$
Base case (data actions): Let \( R = \{r_1, \ldots, r_n\} \). Then,

\[
\text{Sec}(\text{act}\{\text{arg}\}, S) = \\
\text{If} \ \text{then} \ \text{else} [r_1 = \text{role}_{\text{Win}(H, ev)}\text{.name}, \\
\text{If} \ \text{then} \ \text{else} \text{AuthPerm}(S, r_1, \text{act})(\text{Subst}(\text{act}\{\text{arg}\})), \\
\text{act}\{\text{arg}\}, \\
\text{Fail}], \\
\ldots \\
\text{If} \ \text{then} \ \text{else} [r_n = \text{role}_{\text{Win}(H, ev)}\text{.name}, \\
\text{If} \ \text{then} \ \text{else} \text{AuthPerm}(S, r_n, \text{act})(\text{Subst}(\text{act}\{\text{arg}\})), \\
\text{act}\{\text{arg}\}, \\
\text{Fail}], \\
\text{Fail} \ldots ].
\]

Here \( \text{Subst}(\text{act}\{\text{arg}\}) \) is the substitution defined above, where \( w_{\text{Window}} \) is in this case the window that contains the widget that supports the event \( ev \).

Base case (GUI actions):

\[
\text{Sec}(\text{Set}\{\text{variable}, \text{value}\}, S) = \text{Set}\{\text{variable}, \text{value}\}. \\
\text{Sec}(\text{Back}, S) = \text{Back}. \\
\text{Sec}(\text{Skip}, S) = \text{Skip}. \\
\text{Sec}(\text{Open}\{w_{\text{Window}}, (\text{variable}, \text{value})\}, S) = \text{Open}(w_{\text{Window}}, (\text{variable}, \text{value})).
\]

Inductive cases.

\[
\text{Sec}(\text{if} \ \text{then} \ \text{else}\{\text{cond}, \text{stm}_1, \text{stm}_2\}, S) = \\
\text{if} \ \text{then} \ \text{else}\{\text{cond}, \text{Sec}(%(\text{stm}_1, S), \text{Sec}(\text{stm}_2, S))\}. \\
\text{Sec}(\text{iterator}\{\text{source}, \text{variable}, \text{body}\}, S) = \text{iterator}\{\text{source}, \text{variable}, \text{Sec}(\text{body}, S)\}. \\
\text{Sec}(\text{stm}_1; \text{stm}_2, S) = (\text{Sec}(\text{stm}_1, S); \text{Sec}(\text{stm}_2, S)).
\]

4.3 Correctness

The following theorem basically states that the evaluation of a transformed statement following the non-security-aware operational semantics for events returns the same result than its evaluation using the security-aware version of this semantics and, therefore, that the transformed statement respects the authorization constraints formalized in the underlying security model.

**Theorem.** Let \( D \) be a data model \( \langle C, AT, AS, ASO \rangle \), such that \( \text{Role} \in C \) and \( \langle \text{name}, \text{Role}, \text{String} \rangle \in AT \). Let \( S \) be a \( D \)-security model \( \langle D, R, RH, u, P \rangle \). Let \( H \) be a GUIML layout model \( \langle W, WC, X, EV \rangle \) such that for every \( w_{\text{Window}} \in W \),
\langle \text{role}_{\text{Role}}, w_{\text{Window}} \rangle \in X \text{ and } \langle \text{caller}_u, w_{\text{Window}} \rangle \in X. \text{ Let } G \text{ be a GUIML behavioral model } \langle D, H, \text{EST} \rangle. \text{ Let } w_{\text{Window}} \in W \text{ be a window and let } stm \in \text{Stm}(D, H, w). \text{ Then, for every } R\text{-conformant } D\text{-object data model } I, \text{ and every } (X, I)\text{-substitution } \theta,

\langle \text{Sec}(stm, S), I, \theta \rangle \longrightarrow^* \langle \text{Skip}, I', \theta' \rangle. \iff
\langle stm, I, \theta \rangle \longrightarrow^*_{\text{sec}} \langle \text{Skip}, I', \theta' \rangle.

Proof. By induction on \textit{stm}. 

Acknowledgements

This work is partially supported by the EU FP7-ICT Project “NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems” (256980) by the Spanish Ministry of Science and Innovation Project “DESAFIOS-10” (TIN2009-14599-C03-01), and by Comunidad de Madrid Program “PROMETIDOS-CM” (S2009TIC-1465).

References