A Method for Formalizing, Analyzing, and Verifying Secure User Interfaces*

Bernhard Beckert and Gerd Beuster

Institute for Computer Science, University of Koblenz-Landau
beckert@uni-koblenz.de, gb@uni-koblenz.de

Abstract. We present a methodology for the formalization of human-computer interaction under security aspects. As part of the methodology, we give formal semantics for the well-known GOMS methodology for user modeling, and we provide a formal definition of an important aspect of human-computer interaction security. We show how formal GOMS models can be augmented with formal models of (1) the application and (2) the user's assumptions about the application. In combination, this allows the pervasive formal modeling of and reasoning about secure human-computer interaction. The method is illustrated by a simple eVoting example.

1 Introduction

1.1 Overview

We present a methodology for the pervasive formal specification and verification of user interfaces under security aspects. We define formal semantics for GOMS [10], a well-established user modeling methodology. We augment formal GOMS models with formal models of (1) the application and (2) formal models of the user's assumptions about the application. We adapt the common definitions of computer security to the field of human-computer interaction (HCI). For Integrity, an important aspect of HCI security, we provide a formal definition. In combination with a formal definition of human-computer interaction (HCI) security, this allows formal reasoning about the security of user interfaces. Our approach is illustrated by a simple eVoting example.

While formal methods are used extensively in many fields of computer security, they are rarely used in HCI—even for security critical systems. The reason is that HCI does not deal with the interaction of two machines but with the interaction of a machine and a human. While the behavior of a machine can be described precisely with formal methods, human behavior is more difficult to

* This work was partially funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the authors. See http://www.verisoft.de for more information about Verisoft.

describe in a precise way and it can be formalized to a limited extend only. This makes comprehensive modeling of all aspects of user behavior an unreachable goal. However, we argue that, nevertheless it is possible to formally describe human behavior under computer security aspects.

Our approach addresses real world security threats and shows how to counter them. It is directly applicable for the security evaluation of existing systems, as well as to the specification of new systems.

The user modeling methodology presented in this paper is based on the well-established GOMS methodology [10]. GOMS is extensively used for the modeling of user behavior. For our purposes, however, it has two weaknesses: A strict formal semantics is missing, and GOMS models the user behavior independently from the behavior of the system. Both of these short-comings are overcome in this paper.

The structure of this paper is as follows. In Section 2, we develop a formal semantics for GOMS models and illustrate it with an example. In Section 3, that example is completed by adding components representing the application and the user's assumptions about the application. In Section 4, the common definition of computer security is adapted for HCI-security, an a formal definition of Integrity, an important aspect of HCI-security, is developed. In Section 5, our approach is extended to hierarchical models. This allows the *pervasive* description of HCI security and to prove security for all aspects of a user interface—from the pixel level up to high-level functionality of the user interface. Finally, in Section 6, we summarize our work.

1.2 User Modeling Formalisms

User models are routinely used in computer system usability studies. Such user models usually draw on psychological models of the user. They model the user's tasks, goals, motivations, etc. While this is essential under a usability point of view, it makes a comprehensive formal modeling of the effects of user actions infeasible because complex psychological activities can be modeled to a limited extend only. From a usability point of view, this is not necessarily a severe drawback. To guarantee a certain level of usability, it suffices to give plausible evidence that the application's interface is usable, assuming certain goals and behaviors of the user. Security, however, requires a stricter notion of human-computer interaction. While a usability glitch in some dialog window may decrease the general usability of the application a bit, a security glitch can have more severe consequences. Even worse, a security glitch will encourage attackers to seek methods to actually exploit the glitch. The different view on the user and the different goals of usability and security, make it possible and advisable to apply formal methods to security aspects of user interfaces with user models adapted to the particular needs of security.

The computer security problem of proper visual representation of system state is addressed by Duke, Harrison, and others in a number of papers[1, 6–8]. Their focus is to define the relationship between the functional component and the representational component of applications. In [8], they present a theory

of how to describe representations of system state. Our approach is orthogonal to the approach of Duke et al. We present a formal method to reason about correspondonce of the application's state and the user's representation of the state *under the assumption* that the visualization is adequate.

Process oriented formalisms like the well known PIE model developed by Dix and Runciman [5] and its more recent variations (e.g. [4]) allow to describe the interaction of the system and a user formally, but they focus on describing the computer system's side of the interaction. In PIE, the behavior of a user interface is described by a sequence of commands (issued by the user) leading to a sequence of effects. While PIE and similar formalisms put an emphasis on describing the I/O behavior of a computer system and are suitable for automated reasoning (e.g., with model checkers), other approaches like Task Knowledge Structures (TKS) [9], (Extended) Task Action Grammar ((E)TAG) [2], and Goals Operators Methods Selection-rules (GOMS) [10] focus on providing cognitive models of the user. TKS provides an explicit representation of the cognitive model of the user. GOMS is more oriented towards psychological analysis of user behavior and timed measurement of user activity. TAGs allow a precise formal description of the user actions, the user's knowledge and the user's internal representation of the system (what the user thinks about the system).

We base our formalization on GOMS, because GOMS is a well established formalism, and—in the incarnation CMN-GOMS [10]—it allows to describe user models hierarchically. This is an important property for modeling a user interface under security aspects because of the large variety of errors in human-computer interaction. Some of these errors are on a very low level (for example, the user may push the mouse button twice instead of once), while others are on a very high level of abstraction (e.g., the user may misinterpret the meaning of an error message). A hierarchical modeling mechanism allows to model all kinds of errors within one formalism. GOMS models are semi-formal. We provide formal semantics for GOMS models. The formal GOMS model is augmented by formal models of the application and formal models of the user's assumptions about the application. With a formal definition of secure human-computer interaction, this allows to determine the security of a user interface by automated reasoning.

2 Formal Semantics for GOMS User Models

In this section we define formal semantics of GOMS models. In Section 2.1 the formal methods used throughout this paper are defined. Based on these formal methods, formal semantics for GOMS are defined in Section 2.2, and the example used throughout this paper is introduced. In Section 2.3, the formal semantics are extended by defining semantics of selection criteria. In combination with the formal model of the application (Section 3), and a formal definition of HCI security (Section 4), automated reasoning about the security of a HCI interaction model becomes possible.

2.1 Components

Our methodology for the formal description of and reasoning about GOMS makes use of Input Output Labeled Transition Systems (IOLTS) and Linear Temporal Logic (LTL). Below, we define these concepts and some related notions used throughout this paper.

Definition 1. *A* Labeled Transition System *(LTS) is a tuple* $L = (S, \Sigma, s_0, \rightarrow)$ *where* S *is a set of* states*,* $s_0 \in S$ *is an* initial state*,* Σ *is a set of* labels*, and* $\rightarrow \subseteq S \times \Sigma \times S$ *is a* transition relation*. We use the notation* $p \xrightarrow{\sigma} q$ *for* $(p, \sigma, q) \in \rightarrow$*.*

Definition 2. *An* Input Output Labeled Transition System *(IOLTS) is an LTS* $L = (S, \Sigma, s_0, \rightarrow)$ *with* $\Sigma = \Sigma? \cup \Sigma! \cup \Sigma I$*. We call* $\Sigma?$ *the* input alphabet*,* $\Sigma!$ *the* output alphabet*, and* ΣI *the* internal alphabet*.*

We use state transition diagrams to visualize IOLTS. An example is shown in Figure 1.

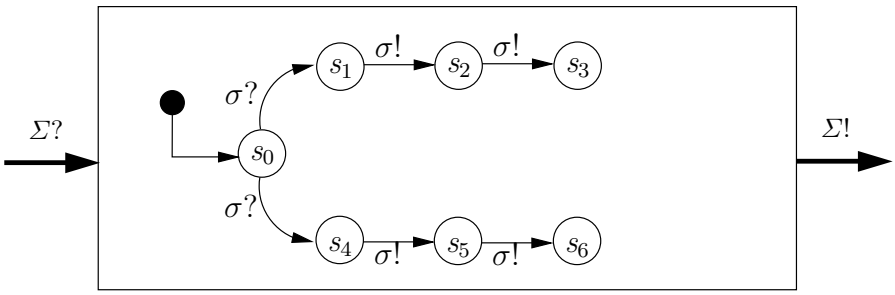

Fig. 1. State Transition Diagram representation of an IOLTS.

The combination of two IOLTSs L_a and L_b where the output alphabet of L_a is the input alphabet of L_b is called a *composition*:

Definition 3. *Let* $L_a = (S_a, \Sigma_a, s_{0\,a}, \rightarrow_a)$*,* $L_b = (S_b, \Sigma_b, s_{0\,b}, \rightarrow_b)$ *be two IOLTS with* $\Sigma!_a = \Sigma?_b$*. The* composition $(L_a \mid\mid L_b) = (S, \Sigma, s_0, \rightarrow)$ *of* L_a *and* L_b *is defined by:*

$$
\begin{aligned}
S &= S_a \times S_b \\
\Sigma? &= \Sigma?_a \\
\Sigma! &= \Sigma!_b \\
\Sigma I &= \Sigma I_a \cup \Sigma I_b \cup \Sigma!_a \\
s_0 &= (s_{0\,a}, s_{0\,b}) \\
\rightarrow &= \{((s_a, s_b), \sigma, (s'_a, s_b)) \mid s_a \xrightarrow{\sigma}_a s'_a \text{ with } \sigma \in \Sigma?_a \cup \Sigma I_a\} \cup \\
&\quad \{((s_a, s_b), \sigma, (s_a, s'_b)) \mid s_b \xrightarrow{\sigma}_b s'_b \text{ with } \sigma \in \Sigma!_b \cup \Sigma I_b\} \cup \\
&\quad \{((s_a, s_b), \sigma, (s'_a, s'_b)) \mid s_a \xrightarrow{\sigma}_a s'_a \text{ and } s_b \xrightarrow{\sigma}_b s'_b \text{ with } \sigma \in \Sigma!_a\}
\end{aligned}
$$

Often, components are combined by *mutual* composition. In mutual composition, the output of L_a serves as input for L_b, and the output of L_b serves as input of L_a (this is illustrated in Figure 2).

Definition 4. *Let* $L_a = (S_a, \Sigma_a, s_{0\,a}, \rightarrow_a)$ *and* $L_b = (S_b, \Sigma_b, s_{0\,b}, \rightarrow_b)$ *be IOLTS.*

We assume the input and output alphabets of L_a *and* L_b *to consist of internal and external subsets, where the internal input is denoted with* $\Sigma?I$*, the external input with* $\Sigma?I$*, the internal output with* $\Sigma!I$*, and the external output with* $\Sigma!E$*. And we demand that these subsets are chosen such that* $\Sigma!I_a = \Sigma?I_b$ *and* $\Sigma!I_b = \Sigma?I_a$*.*

Then, the mutual composition $(L_a \,||_m\, L_b) = (S, \Sigma, s_0, \rightarrow)$ *of* L_a *and* L_b *is defined by:*

$$
\begin{aligned}
S &= S_a \times S_b \\
\Sigma? &= \Sigma?E_a \cup \Sigma?E_b \\
\Sigma! &= \Sigma!E_a \cup \Sigma!E_b \\
\Sigma I &= \Sigma I_a \cup \Sigma I_b \cup \Sigma!I_a \cup \Sigma!I_b \\
s_0 &= (s_{0\,a}, s_{0\,b}) \\
\rightarrow &= \{(s_a, s_b), \sigma, (s'_a, s_b)) \mid s_a \xrightarrow{\sigma}_a s'_a \text{ with } \sigma \in \Sigma?E_a \cup \Sigma!E_a \cup \Sigma I_a\} \cup \\
&\quad \{(s_a, s_b), \sigma, (s_a, s'_b)) \mid s_b \xrightarrow{\sigma}_b s'_b \text{ with } \sigma \in \Sigma?E_b \cup \Sigma!E_b \cup \Sigma I_b\} \cup \\
&\quad \{(s_a, s_b), \sigma, (s'_a, s'_b)) \mid s_a \xrightarrow{\sigma}_a s'_a \text{ and } s_b \xrightarrow{\sigma}_b s'_b \text{ with} \\
&\qquad\qquad \sigma \in \Sigma!I_a \cup \Sigma!I_b\}
\end{aligned}
$$

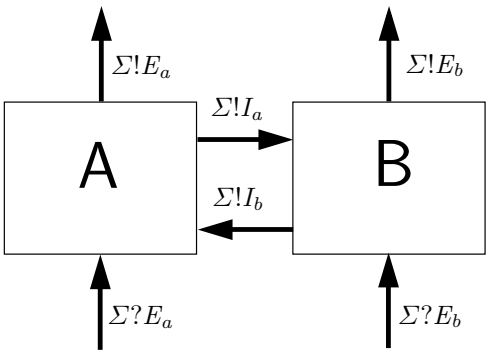

Fig. 2. Mutual composition of IOLTSs.

The input/output behavior of a component is described by *traces*, which are (possibly infinite) sequences of elements from the alphabet Σ, and *paths*, which are corresponding sequences of states.

Definition 5. *Let* $L = (S, \Sigma, s_0, \rightarrow)$ *be an IOLTS. Then, a* path *is a sequence* $\langle s_0, s_1, \ldots \rangle$ *of states from* S *with* $s_i \rightarrow s_{i+1}$ *for all* $i \geq 0$*. A* trace *(of* L*) is a*

sequence $\langle \sigma_0, \sigma_1, \dots \rangle$ *of elements of* Σ *such that there is a path* $\langle s_0, s_1, \dots \rangle$ *with* $s_i \xrightarrow{\sigma_i} s_{i+1}$ *(*$i \geq 0$*).*

We use Linear Temporal Logic (LTL) to describe properties of components. The syntax of LTL is defined as usual, i.e., given a set P of atomic propositions, LTL formulae ϕ are constructed inductively by:

$$\phi ::= p \mid \phi \vee \phi \mid \phi \wedge \phi \mid \neg\phi \mid X\phi \mid \phi U \phi \mid G\phi \mid F\phi \qquad (p \in P)$$

Now, we can use IOLTSs to interpret LTL formulas—in combination with valuations λ that map atomic propositions to the states in which they are true. The satisfaction relation is extended to more complex formulae as usual.

Definition 6. *Given an IOLTS* $L = (S, \Sigma, s_0, \rightarrow)$ *and a set* P *of atomic propositions, a* valuation λ *is a mapping from* P *to* S*. An atom* $p \in P$ *is said to be true in* $s \in S$ *iff* $s \in \lambda(p)$*.*

Given a path $c = \langle s_0, s_1, \dots \rangle$*, by* c^i *we denote the sub-path of* c *starting at* s_i*.*

Whether an LTL formula ϕ *is satisfied by a path* c *and a valuation* λ*, denoted by* $L, \lambda, c \models \phi$*, is inductively defined as follows:*

- $L, \lambda, c \models \top$
- $L, \lambda, c \models \phi$ *if* $\phi \in P$ *and* $s_0 \in \lambda(\phi)$
- $L, \lambda, c \models \neg\, \phi$ *if not* $L, \lambda, c \models \phi$
- $L, \lambda, c \models \phi \wedge \psi$ *if* $L, \lambda, c \models \phi$ *and* $L, \lambda, c \models \psi$
- $L, \lambda, c \models \phi \vee \psi$ *if* $L, \lambda, c \models \phi$ *or* $L, \lambda, c \models \psi$
- $L, \lambda, c \models \boldsymbol{X}\phi$ *if* $L, \lambda, c^1 \models \phi$
- $L, \lambda, c \models \phi \boldsymbol{U} \psi$ *if (a)* $L, \lambda, c \models \psi$ *or (b) there is some* $i \geq 1$ *s.t.* $L, \lambda, c^i \models \psi$ *and* $L, \lambda, c^k \models \phi$ *for all* $0 \leq k < i$
- $L, \lambda, c \models \boldsymbol{G}\phi$ *if* $L, c^i \models \phi$ *for all* $i \geq 0$
- $L, \lambda, c \models \boldsymbol{F}\phi$ *if* $L, c^i \models \phi$ *for some* $i \geq 0$

An LTL formula ϕ *is said to be satisfied by a valuation* λ*, denoted by* $L, \lambda \models \phi$*, iff* $L, \lambda, c \models \phi$ *for all paths* c *of* L*. And* ϕ *is said to be satisfied by* L*, denoted by* $L \models \phi$ *iff* $L, \lambda \models \phi$ *for all valuations* λ*.*

2.2 Using IOLTS Traces to Define the Semantics of GOMS Models

We now provide a formal semantics for the GOMS user modeling methodology. GOMS describes human behavior in categories of

*G*oals	The user's goals
*O*perators	Atomic actions available to the user
*M*ethods	Sequences of operators and sub-goals
*S*election Rules	Rules to decide between alternative methods

We formalize the CMN-GOMS variant of GOMS [10]. In difference to other GOMS variants, CMN-GOMS satisfies the two core requirements for the formal description of human behavior under security aspects: It allows to model

user behavior on different levels of abstractions, and CMN-GOMS's informal semantic is suitable for formalization. In CMN-GOMS, methods for achieving a goal consist of sequences of sub-goals and atomic operators (the only difference between sub-goals and atomic operators is that operators cannot be further decomposed). If there is more than one way to reach a goal, a selection rule is used to choose between alternatives.

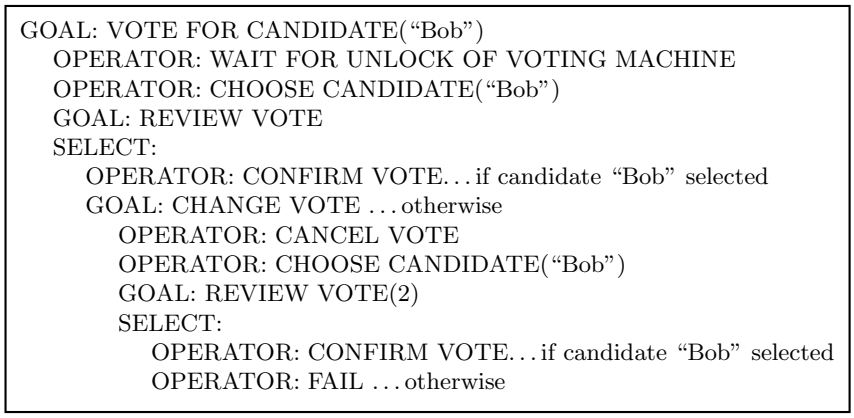

Fig. 3. GOMS model for eVoting.

Figure 3 gives an example. It models the user of an eVoting machine. In order to achieve the goal "VOTE FOR CANDIDATE('Bob')", the user executes the method consisting of the atomic operations "WAIT FOR UNLOCK OF VOTING MACHINE" and "CHOOSE CANDIDATE('Bob')". Then he reviews his vote. The sub-goal "REVIEW VOTE" can be achieved in two ways: (1) If the user has selected the right candidate, he confirms. (2) If he has selected the wrong candidate, he pursues sub-goal "CHANGE VOTE". Changing the vote leads to the sub-goal "REVIEW VOTE(2)". If the user has selected the right candidate this time, he confirms; otherwise, voting fails.

We give a formal semantics for GOMS models using the notion of IOLTS traces. That is, an IOLTS corresponds to a GOMS model if the traces of the IOLTS are identical to the possible sequences of user decisions (selections) and operations. In order to formally define, which IOLTS correspond to a given GOMS model, we use the following formal syntax for GOMS models:

Definition 7. *Given a GOMS model, the corresponding* formal GOMS model *is*

$$T = (G, O, M, R, C, g_0)$$

where

- G *is the set of (sub-)goals;*

- *O is the set of operators;*
- *C is the set of selection criteria;*
- *M is a function mapping goals to their sequences of sub-goals/operators.*
- *The function $R : G \times C \longrightarrow G$ is defined by: $R(g, c) = g'$ iff the goal g is achieved by sub-goal/operator g' in case criteria c holds;*
- *g_0 is the top-level goal.*

The formal GOMS model corresponding to the eVoting GOMS model from Figure 3 is shown in Figure 4.

$$
\begin{aligned}
&T = (G, O, M, R, C, g_0) \text{ with} \\
&\quad G = \{\text{VOTE\_FOR\_CANDIDATE(“Bob”), REVIEW\_VOTE,} \\
&\qquad \text{CHANGE\_VOTE, REVIEW\_VOTE(2)}\} \\
&\quad O = \{\text{WAIT\_FOR\_UNLOCK, CHOOSE\_CANDIDATE,} \\
&\qquad \text{CONFIRM\_VOTE, CANCEL\_VOTE, FAIL}\} \\
&\quad C = \{\text{Candidate “Bob” selected}, \neg(\text{Candidate “Bob” selected})\} \\
&\quad M(g) = \begin{cases} \langle \text{WAIT\_FOR\_UNLOCK}, \text{CHOOSE\_CANDIDATE}, \text{REVIEW\_VOTE}\rangle \\ \qquad \text{if } g = \text{VOTE\_FOR\_CANDIDATE} \\ \langle \text{CANCEL\_UNLOCK}, \text{CHOOSE\_CANDIDATE}, \text{REVIEW\_VOTE(2)}\rangle \\ \qquad \text{if } g = \text{CHANGE\_VOTE} \end{cases} \\
&\quad R(g, c) = \begin{cases} \text{CONFIRM\_VOTE} & \text{if } g = \text{REVIEW\_VOTE and} \\ & \quad c = \text{Candidate “Bob” selected} \\ \text{CHANGE\_VOTE} & \text{if } g = \text{REVIEW\_VOTE and} \\ & \quad c = \neg \text{ (Candidate “Bob” selected)} \\ \text{CONFIRM\_VOTE} & \text{if } g = \text{REVIEW\_VOTE(2) and} \\ & \quad c = \text{Candidate “Bob” selected} \\ \text{FAIL} & \text{if } g = \text{REVIEW\_VOTE(2) and} \\ & \quad c = \neg \text{ (Candidate “Bob” selected)} \end{cases}
\end{aligned}
$$

Fig. 4. Formal GOMS model for the eVoting model from Figure 3.

We define a formal semantics for GOMS models by translating the formal GOMS model into an IOLTS. The idea is to represent operators as elements of the output alphabet, selections as elements from the input alphabet, and methods as (sub-)paths. Selection rules are branching points in the IOLTS. Figure 5 illustrates this translation.

Definition 8. *Let $T = (G, O, M, R, C, g_0)$ be a formal GOMS model. And let $(S, \Sigma, S_0, \rightarrow)$ be the (generalized) IOLTS constructed for T and $S_0 = \{s_0\}$ by the algorithm shown in Figure 6.*

Then $(S, \Sigma, s_0, \rightarrow)$ is the IOLTS corresponding to *T.*

Note that the algorithm in Figure 6 constructs an IOLTS that is generalized in the sense that is may have more than one initial state. If the algorithm is

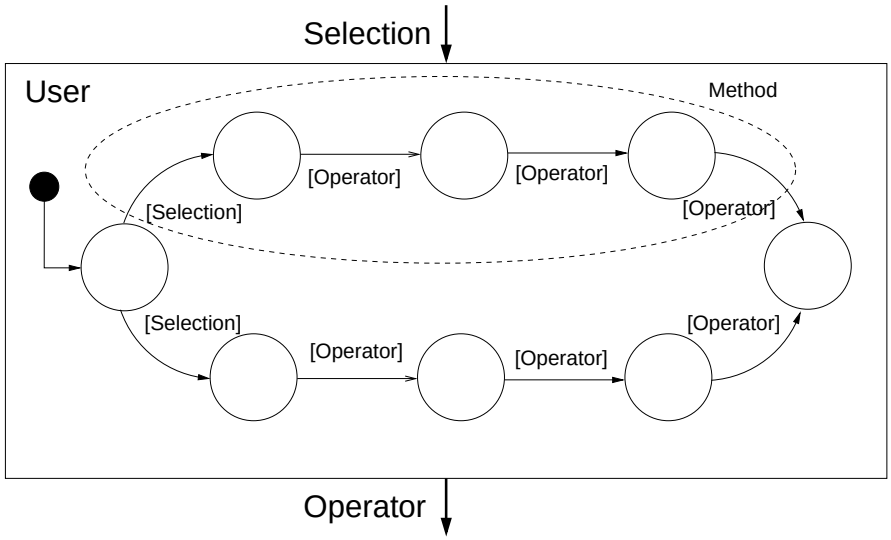

Fig. 5. Translating GOMS categories to state transition diagrams.

started with a singleton set $S_0 = \{s_0\}$ of initial states, a standard IOLTS is constructed (the more general case is only needed for the recursive calls within the algorithm). An implementation of the algorithm in the Perl programming language has been used for constructing the example IOLTSs presented in this paper.

Applying the algorithm to the eVoting example results in the following IOLTS that corresponds to the GOMS model shown in Figure 3 resp. 4. The IOLTS is shown graphically in Figure 7.

$$
\begin{aligned}
S &= \{s_0, \ldots, s_{11}\} \\
\Sigma &= \Sigma? \cup \Sigma! \\
\Sigma? &= \{\text{“Bob” selected}, \neg\,(\text{“Bob” selected}\} \\
\Sigma! &= \{\text{WAIT\_FOR\_UNLOCK, CONFIRM\_VOTE, CANCEL\_VOTE, FAIL,} \\
&\quad \text{CHOOSE\_CANDIDATE}\}
\end{aligned}
$$

Require: GOMS model $T = (G, O, M, R, c, g_0)$, and a set S_0 of initial states
Ensure: (Generalized) IOLTS $L = (S, \Sigma, S_0, \rightarrow)$ and set F of states,
s.t. $\Sigma? = C$, $\Sigma! = O$, and F contains the final states of L

if $g_0 \in O$ **then**
  {initial goal is an atomic operator}
  create new state s_1
  $S := S_0 \cup \{s_1\}$
  $\Sigma? := \varnothing$
  $\Sigma! := \{g_0\}$
  $\rightarrow := \{(s_0, g_0, s_1) \mid s_0 \in S_0\}$
  $F := \{s_1\}$
else if $M(g_0) = \langle m_1, \ldots, m_n \rangle$ **then**
  {initial goal has sub-goals $g_1, \ldots, g_n$}
  $S := \varnothing$
  $\Sigma? := \varnothing$
  $\Sigma! := \varnothing$
  $\rightarrow := \varnothing$
  $F := S_0$
  for $i = 1 \ldots n$ **do**
    create an IOLTS $L_i = (S_i, \Sigma_i, S_0^i, \rightarrow_i)$ with final states F_i
      for $T_i = (G, O, M, R, c, g_i)$ and set $S_0^i := F$ of initial states
      by recursion
    $S := S \cup S_i$
    $\Sigma? = \Sigma? \cup \Sigma?_i$
    $\Sigma! = \Sigma! \cup \Sigma!_i$
    $\rightarrow = \rightarrow \cup \rightarrow_i$
    $F = F_i$
  end for
else
  {initial goal is a selection point}
  for all g_i, c_i such that $R(g_0, c_i) = g_i$ **do**
    create a new state s_i
    $S := S \cup \{s_i\}$
    $\rightarrow = \rightarrow \cup \{(s_0, c_i, s_i) \mid s_0 \in S_0\}$
    create an IOLTS $L_i = (S_i, \Sigma_i, S_0^i, \rightarrow_i)$ with final states F_i
      for $T_i = (G, O, M, R, c, g_i)$ and set $S_0^i := \{s_i\}$ of initial state
      by recursion
    $S := S \cup S_i$
    $\Sigma? = \Sigma? \cup \Sigma?_i$
    $\Sigma! = \Sigma! \cup \Sigma!_i \cup \{c_i\}$
    $\rightarrow = \rightarrow \cup \rightarrow_i$
    $F = F \cup F_i$
  end for
end if

Fig. 6. Algorithm for constructing an IOLTS corresponding to a given GOMS model.

$$
\begin{aligned}
\rightarrow = \{&(s_0, \text{WAIT\_FOR\_UNLOCK}, s_1),\\
&(s_1, \text{CHOOSE\_CANDIDATE}(\text{“Bob”}), s_2),\\
&(s_2, \text{“Bob” selected}, s_3),\\
&(s_3, \text{CONFIRM\_VOTE}, s_4),\\
&(s_2, \neg\ (\text{“Bob selected}), s_5),\\
&(s_5, \text{CANCEL\_VOTE}, s_6),\\
&(s_6, \text{CHOOSE\_CANDIDATE}(\text{“Bob”}), s_7),\\
&(s_7, \text{“Bob” selected}, s_8),\\
&(s_8, \text{CONFIRM\_VOTE}, s_9),\\
&(s_7, \neg\ (\text{“Bob” selected}), s_{10}),\\
&(s_{10}, \text{FAIL}, s_{11})\}
\end{aligned}
$$

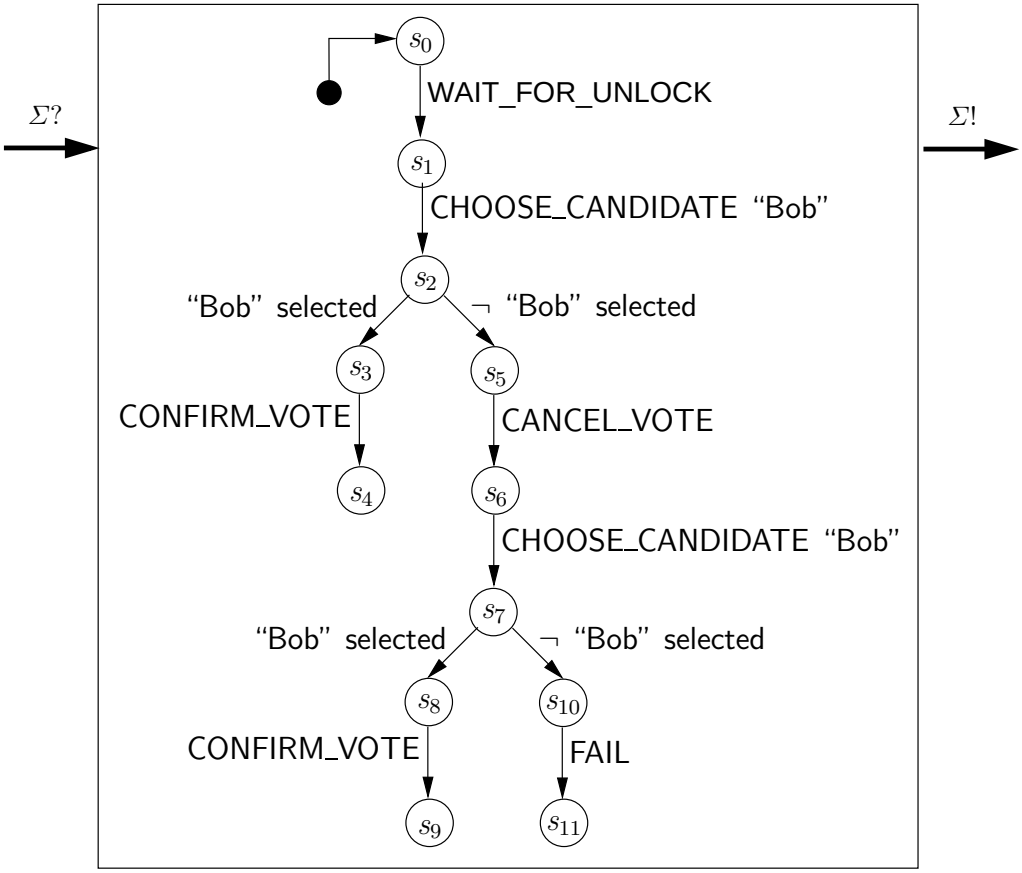

Fig. 7. IOLTS corresponding to the eVoting GOMS model.

2.3 Assumptions as Selection Rules

Selection rules in GOMS models require decision criteria. In GOMS, these criteria are only specified in an informal way. Since our goal is to provide a formal semantics for GOMS models suitable for automated reasoning, a methodology for the formal description of selection criteria is required.

If a user is in the situation to choose between multiple options, his decision will be based on the current system configuration or, more precisely, on his *perception* of the system configuration. In the eVoting example, the decision whether to confirm his vote or to change it, depends on the candidate selection shown by the voting machine and the user's corresponding perception of the machine's internal configuration.

Following our component-based approach, we define the user's assumption about the system configuration as a component. This component is combined with the (IOLTS corresponding to the) formal GOMS model by mututal composition. The rational behind mutual composition is that not only do the user's presumptions about the applicaton state influence his behavior but his assumptions about the state of the application are influenced by his actions as well. For example, when the user pushes the "confirm vote" button, he will asume that the voting process is completed, even if it takes some time before the next message appears on the screen. The other input for the assumption component—besides the users actions, i.e., the operators in the GOMS model—comes from the output of the application (application output is defined in Section 3). Figure 8 illustrates the composition of an interactive formal user model.

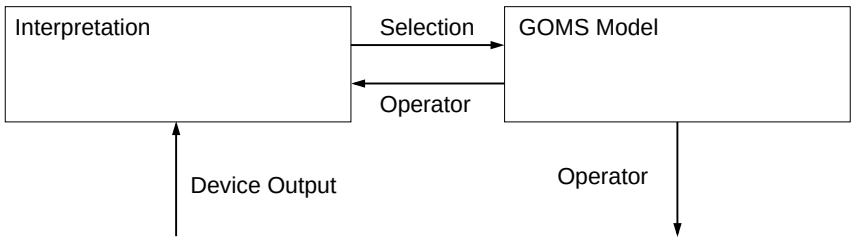

Fig. 8. Combination of GOMS model and user's interpretation of the application's configuration..

Definition 9. *An IOLTS* $L = (S, \Sigma, s_0, \rightarrow)$ *is called a* user assumption IOLTS, *if*

- $\Sigma = \Sigma? \cup \Sigma!$,
- $\Sigma? = \Sigma?_D \cup \Sigma?_A$ *where* $\Sigma?_D$ *atomic application (device) output and* $\Sigma?_A$ *are GOMS operators,*
- $\Sigma!$ *consists of GOMS selection criteria.*

An interactive formal user model $L = (L_A \,||_m\, L_I)$ *is the mutual composition of the IOLTS* L_U *corresponding to a formal GOMS model (user model) and a user assumption IOLTS* L_I*.*

2.4 Formal HCI Model: Summary

We have defined formal semantics for GOMS models and for selection criteria. Selection criteria are defined by a component modeling the user's assumptions about the application. The combination of a formal GOMS models of the user and a model of the user's assumptions allows the formal description of human behavior.

In order to reason about security of HCI, a formal application model and a formal definition of HCI security is required in addition. In Section 3, we complete the eVoting example. We provide an application model and two alternative user assumption components. In Section 4, definitions of generic formal HCI security requirements are given and applied to the eVoting example.

3 Completing the eVoting Model

In order to apply automated reasoning to human-computer interaction, we need three components: (1) A formal GOMS model and its corresponding IOLTS; (2) a component representing the assumptions of the user about the application; and (3) a component representing the application itself. In this section, we provide the missing two components for the eVoting example, starting with the application.

We assume that the eVoting machine is initially in a locked state. After some time, the machine is unlocked and the user can cast his vote. After he has selected a candidate, the machine shows the user's choice and asks for confirmation. If he confirms, the voting process finishes. If he cancels, he can change the vote. Figure 9 sketches an IOLTS modeling the voting machine. The input alphabet is identical to the output alphabet of the user model IOLTS, i.e., the operators available to the user. The output alphabet is an abstract representation of the application's output (in Section 5 we introduce hierarchical models which allow to model application output down to the pixel level). In order to make the example interesting, we have built a bug into the IOLTS: If a user votes for "Bob", the eVoting machine may mistakenly interpret this as a vote for "Fred":

$$S = \{s_0, s_1, s_2\} \cup \bigcup_{c \in \text{Candidates}} \{s_c, s'_c, s''_c, s'''_c\}$$
$$\Sigma = \Sigma? \cup \Sigma!$$
$$\Sigma? = \{\text{WAIT\_FOR\_UNLOCK, CONFIRM\_VOTE, CANCEL\_VOTE, FAIL, CHOOSE\_CANDIDATE}\}$$
$$\Sigma! = \{\text{locked}, \text{unlocked}\} \cup \bigcup_{c \in \text{Candidates}} \{\text{Vote cast}(c), \text{Vote confirmed}(c)\}$$

$$
\begin{aligned}
\rightarrow = \{&(s_0, \text{WAIT\_FOR\_UNLOCK}, s_1),\\
&(s_1, \text{unlocked}, s_2)\} \cup\\
&\{(s_2, \text{CHOOSE\_CANDIDATE}[c], s_c) \mid c \in \text{Candidates}\} \cup\\
&\{(s_c, \text{Vote cast}(c), s'_c) \mid c \in \text{Candidates}\} \cup\\
&\{(s'_c, \text{CHANGE\_VOTE}, s_2) \mid c \in \text{Candidates}\} \cup\\
&\{(s'_c, \text{CONFIRM\_VOTE}, s''_c) \mid c \in \text{Candidates}\} \cup\\
&\{(s''_c, \text{Vote confirmed}(c), s'''_c) \mid c \in \text{Candidates}\} \cup\\
&\{(s_{\text{“Bob''}}, \text{Vote cast (“Fred”)}, s'_{\text{“Fred''}})
\end{aligned}
$$

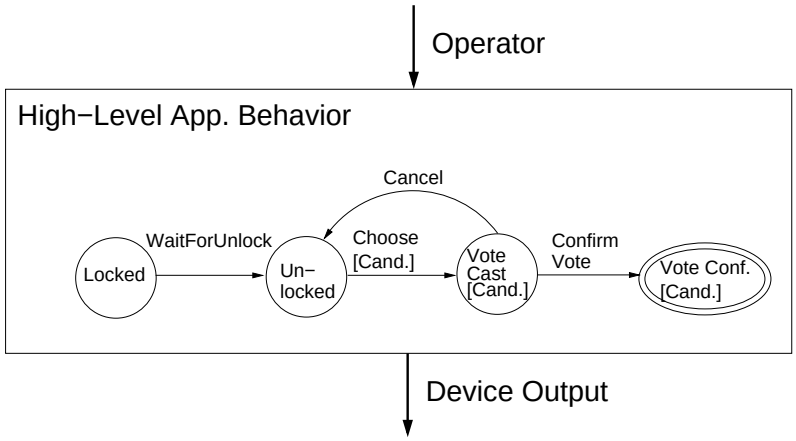

Fig. 9. Application Model for the eVoting example.

For the completion of the example, we still need a model of the user's assumptions. As defined in the last section, a user assumption component has an input alphabet consisting of the application's output and the user's operators, and an output alphabet consisting of the user's selection criteria. In this example we use user assumption components that only use the application's output as input (in order to keep the example simple). Selection rules are used at two points in the GOMS model: When the user reviews his voting decision for the first time, and when he reviews his voting decision for the second time. The user's assumption is that the eVoting application works correctly. Thus, the assumption component will output "candidate 'Bob' selected" for the input "Vote cast('Bob')", and "¬(Candidate 'Bob' selected)" for the input "Vote cast(c)" with $c \neq$ "Bob". This "error-free" model corresponds to the following user assumption IOLTS:

$$
\begin{aligned}
S &= \{s_0, s_{bob}, s_{other}\}\\
\Sigma &= \Sigma? \cup \Sigma!\\
\Sigma? &= \{\text{locked}, \text{unlocked}\} \cup \textstyle\bigcup_{c \in \text{Candidates}} \{\text{Vote cast}(c), \text{Vote confirmed}(c)\}
\end{aligned}
$$

$$
\begin{aligned}
\Sigma! = {} & \{\text{Candidate 'Bob' selected}, \neg(\text{Candidate 'Bob' selected})\} \\
\rightarrow = {} & \{(s_0, \sigma, s_0) \mid \sigma \neq \text{Vote cast}(c) \text{ for all candidates } c\} \cup \\
& \{(s_0, \text{Vote cast('Bob')}, s_{bob})\} \cup \\
& \{(s_0, \text{Vote cast}(c), s_{other}) \mid c \neq \text{"Bob"}\} \cup \\
& \{(s_{bob}, \text{Candidate 'Bob' selected}, s_0)\} \cup \\
& \{(s_{other}, \neg\ (\text{Candidate 'Bob' selected}), s_0)
\end{aligned}
$$

While standard GOMS does not allow to model user errors, our component-based approach does. As an example, we model a user who may think the system is in a state where he voted for "Bob" while in fact he voted for someone else. The changed relation $\rightarrow$ is shown below:

$$
\begin{aligned}
\rightarrow \ = \ & \{(s_0, \sigma, s_0) \mid \sigma \neq \text{Vote cast}(c) \text{ for all candidates } c\} \cup \\
& \{(s_0, \text{Vote cast}(c), s_{bob}) \mid c \in \text{Candidates}\} \cup \\
& \{(s_0, \text{Vote cast}(c), s_{other}) \mid c \neq \text{"Bob"}\} \cup \\
& \{(s_{bob}, \text{Candidate 'Bob' selected}, s_0)\} \cup \\
& \{(s_{other}, \neg\ (\text{Candidate 'Bob' selected}), s_0)
\end{aligned}
$$

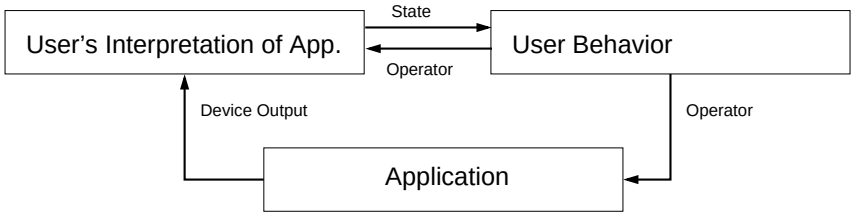

Fig. 10. Basic system model.

In this section, we showed how system models are created from formal GOMS models, user assumption components, and application models. The mutual compositions of these three components—as shown in Figure 10—provide a complete model. With this, complete formal modeling of human-computer interaction becomes possible. In difference to traditional methods, our method also allows to model erroneous user behavior.

In the next section, we define HCI security properties as LTL formulae. With the formal definition of HCI security properties and the modeling methodology developed in this section, formal methods can be used for reasoning about security of user interaction.

4 HCI Security Definitions

The aim of computer security is to guarantee access to services and resources to authorized persons, while preventing access and manipulation by unauthorized parties. The basic security threats are *Data Leaking*, *Data Manipulation*,

and *Program Manipulation* [3]. These are countered by the core security requirements, usually abbreviated as *CIA*:

Confidentiality: Information is available to authorized parties only.
Integrity: Both the assumptions of the user about the application, and the assumptions of the application about the user are correct.
Availability: Accessibility of services and data is guaranteed.

Adapting these concepts to user interface security is straightforward:

HCI Confidentiality: No secret information is leaked via the user interface.
HCI Integrity: There is a correspondence between the configuration of the application (defined by its internal state and data), and the user's assumption about the data and the state.
HCI Availability: The user interface must guarantee reachability of desirable states, and it must prevent user interactions that lead to transitions into undesirable states.

In the following, we concentrate on formalizing the integrity requirement. Informally, we define HCI Integrity as follows:

Definition 10. *HCI Integrity: Whenever the system is in a critical state, all critical properties are the same in the application and in the user's assumption about the application.*

Let the set P of atomic propositions contains the following atoms:

- $appCritical$ is true whenever the application is in a critial state.
- $a_0, \ldots, a_n$ represent the critical properties of the application.
- $u_0, \ldots, u_n$ represent the user's assumption about critical properties.

Then, we can formalize HCI Integrity using the LTL formula

$$\mathbf{G}(appCritical \rightarrow ((a_0 \leftrightarrow u_0) \wedge (a_1 \leftrightarrow u_1) \wedge \ldots \wedge (a_n \leftrightarrow u_n)))$$

In the eVoting example, the critical property is the user's vote, and the critical state is reached once the user has finished voting. If in that state the user thinks he selected the candidate of his choice, while in fact he voted for some other candidate, human-computer interaction was erroneous. Thus, we choose u_0 to represent "the user has voted for 'Bob"' and a_0 to represent "the user thinks he has voted for 'Bob"'. Critical states are those states of the application model where a vote has been confirmed.

We can now use automated reasoning techniques (e.g., model checking) to confirm that, whenever the valuation λ reflect this interpretation of the atoms, the HCI Integrity formula holds.

In the example, despite the bug in the application model (choosing "Bob" may be credited to "Fred"), the Integrity requirement holds for the model where the user makes correct assumptions about the system state, because the user will recognize the error when he is asked to confirm the vote for "Fred". In the variant of the eVoting model with the erronous user assumptions model, Integrity does not hold, because the user may mistakely confirm the vote for "Fred".

5 Hierarchical Model

In the models introduced so far, the application, the user's actions, and the user's assumptions are modeled as monolithic components. When we start to add more details to our models—for example, when application output and user perception is modeled in more detail—the components become unwieldy.

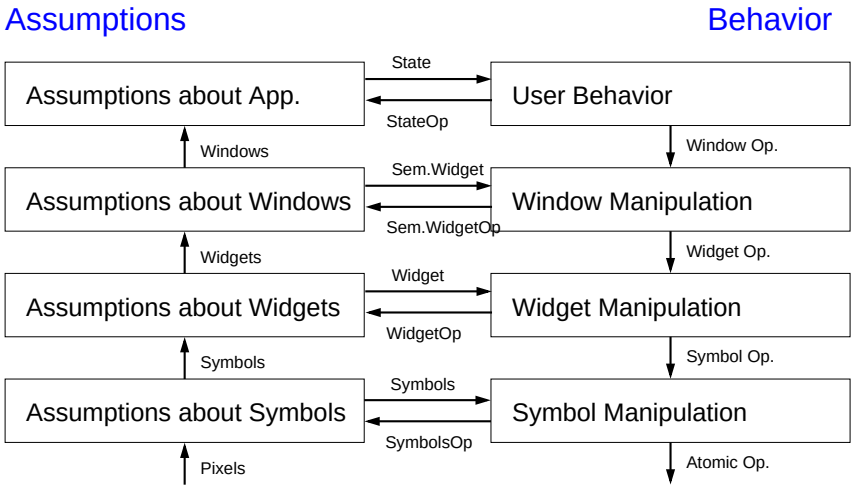

Fig. 11. Generic Hierarchical User Model

To counter this problem, we introduce hierarchical components. In a model of hierarchical components, components of different levels of abstraction are layered above each other. This allows to describe user interfaces and human-computer interaction at all levels of detail with model still managable by humans and computers.

Both in the construction of graphical user interfaces and in the perception (and interpretation) of graphical user interfaces, there are generic abstraction levels shared over a large class of interfaces. By identifying these abstraction levels and modeling user interfaces along these lines, it becomes possible to model complex user interfaces (and potential error sources in complex user interfaces) while still preserving maintainability of the models. The proposed model pattern is shown in Figure 11.

The sub-concepts of the user interface follow the well established hierarchical view of interfaces. On the uppermost level, a user interface consists of distinct screens. Each screen represents a specific view on the application. Screens themselves are built from a number of windows, windows are built from widgets, and these are built from elementary symbols.

Creating a hierarchical user interface where each component represents one level of abstraction makes it possible to model typical errors on their respective

levels. For example, the typical error that a user misses a button and pushes a wrong one, is modeled on a low level, while the error that a user misinterprets a screen is modeled on a high level.

In our eVoting example, a user may accidently push the button for "Fred" if it is next to the button for "Bob." This error can be modeled on the symbol manipulation level by the GOMS sub-model for the "CHOOSE_CANDIDATE('Bob')" operator shown in Figure 12 and the following assumption component about widget manipulation:

$$
\begin{aligned}
S &= \{s_0, s_1, \ldots, s_n\} \\
\Sigma &= \Sigma? \cup \Sigma! \\
\Sigma? &= \{(\text{“Bob's Button”} = 1), \\
&\qquad (\text{“Bob's Button”} = 2), \\
&\qquad \ldots, \\
&\qquad (\text{“Bob's Button”} = n)\} \\
\Sigma! &= \{(\text{“Bob's Button”} = 1), \\
&\qquad (\text{“Bob's Button”} = 2), \\
&\qquad \ldots, \\
&\qquad (\text{“Bob's Button”} = n) \\
\rightarrow &= \{(s_0, (\text{“Bob's Button”} = i), s_i) \mid 1 \leq i \leq n\} \cup \\
&\qquad \{(s_0, (\text{“Bob's Button”} = i - 1), s_i)) \mid 1 < i \leq n\} \cup \\
&\qquad \{(s_0, (\text{“Bob's Button”} = i + 1), s_i) \mid 1 \leq i < n\}
\end{aligned}
$$

GOAL: CHOOSE_CANDIDATE("Bob")
SELECT: OPERATOR: PUSH_BUTTON(0) ... if "Bob's Button" = 0
OPERATOR: PUSH_BUTTON(1) ... if "Bob's Button" = 1
OPERATOR: PUSH_BUTTON(2) ... if "Bob's Button" = 2
⋮

Fig. 12. Sub-Model for CHOOSE_CANDIDATE("Bob").

Our approach to the construction of sub-models of GOMS models is depicted in Figure 13. Each method for achieving a sub-goal becomes a GOMS model on its own. The IOLTSs corresponding to these GOMS models can then be combined into one component such that the operators from the higher GOMS model become selection criteria. Formally, this is defined as follows:

Definition 11. *Let*

- $T = (G, O, M, R, C, g_0)$ *be a GOMS model with the corresponding IOLTS* $L = (S, \Sigma, s_0, \rightarrow)$*, and let*
- $T_i = (G_i, O_i, M_i, R_i, C_i, g_0^i)$ *be GOMS models* $(1 \leq i \leq n)$ *with the corresponding IOLTSs* $L_i = (S_i, \Sigma, s_0^i, \rightarrow_i)$

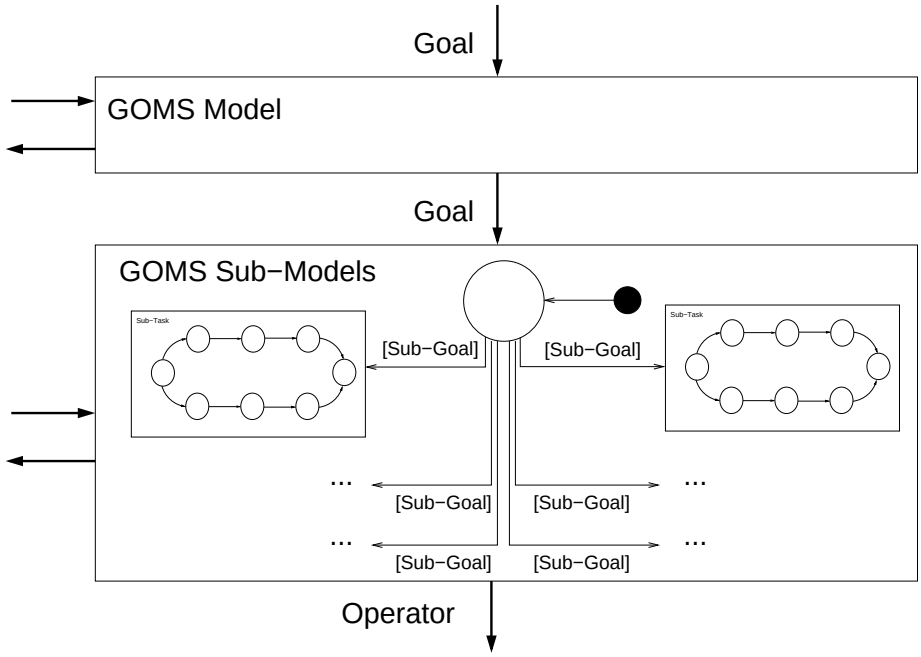

Fig. 13. Hierarchical GOMS model.

such that $O = \{g_0^1, \ldots, g_0^n\}$*, i.e., the operators of* T *are the top-level goals of* $T_1, \ldots, T_n$*.*

Then, the IOLTS $L' = (S', \Sigma', s_0', \rightarrow')$ *for the hierarchical model consisting of* T *and* $T_1, \ldots, T_n$ *is defined by*

$$S' = \{s_0'\} \cup \bigcup_{1 \leq i \leq n} S_i$$
$$\Sigma' = \Sigma!' \cup \Sigma?'$$
$$\Sigma?' = \bigcup_{1 \leq i \leq n} \Sigma?_i$$
$$\Sigma!' = \bigcup_{1 \leq i \leq n} \Sigma!_i$$
$$\rightarrow' = \{s_0' \xrightarrow{g_0^i} s_0^i) \mid 1 \leq i \leq n\} \cup \bigcup_{1 \leq i \leq n} \rightarrow_i$$

6 Summary

In this paper, we have introduced a methodolgy for formalizing, analyzing, and verifying user interfaces and human-computer interaction under computer security aspects. The main point of this work is to provide a formal semantics for an extended version of GOMS that is suitable for automatic reasoning. In this paper:

– We have introduced a formal semantics for GOMS models describing user behavior, which is based on input/output latbelled transition systems (IOLTS).

– We showed how the component-based formalization of GOMS can be augmented with components modeling the user's assumptions about the application. That allows to model both successful HCI and erroneous HCI.
– The method used to formalize GOMS models and the user's assumption can be applied to model the application as well. Combining all three components leads to a complete model of human-computer interaction suited for automated reasoning.
– We have introduced a methodology to formally describe hierarchical user interfaces. That allows to pervasively model all aspects of user interface security.
– We have formalized generic concepts of user interface security in linear temporal logic. In combination with a formal model of HCI, that allows to use automated reasoning to determine if a user interface is secure.

References

1. C. Bramwell. Formal development methods for interactive systems: Combining interactors and design rationale, 1996.
2. Geert de Haan. *ETAG: A Formal Model of Competence Knowledge for User-Interface Design*. PhD thesis, Vrije Universiteit, Amsterdam, 2000.
3. Rüdiger Dierstein. Sicherheit in der Informationstechnik: Der Begriff IT-Sicherheit. *Informatik Spektrum*, 27(4), August 2004.
4. Alan Dix and Gregory Abowd. Modelling status and event behaviour of interactive systems. *Software Engineering Journal*, 11(6):334–346, 1996.
5. Alan Dix and Colin Runciman. Abstract models of interactive systems. In P. Johnson and S. Cook, editors, *HCI'85: People and Computers I: Designing the Interface*, pages 13–22. Cambridge: Cambridge University Press, 1985.
6. Gavin Doherty and Michael D. Harrison. A Representational Approach to the Specification of Presentations. *Eurographics Workshop on Design Specification and Verification of Interactive Systems, DSVIS 97, Granada, Spain*, June 1997.
7. D. Duke, P. Barnard, D. Duce, and J. May. Systematic development of the human interface, 1995.
8. D. J. Duke and M. D. Harrison. A Theory of Presentations. In M. Naftalin, T. Denvir, and M. Bertran, editors, *Proceedings of FME'94: Industrial Benefit of Formal Methods*, pages 271–290. Srpinger-Verlag, 1994.
9. F. Hamilton. Predictive evaluation using task knowledge structures. In *Companion Proceedings of CHI'96, Vancouver, Canada*, 1996.
10. B. E. John and D. E. Kieras. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction*, 3(4):320–351, 1996.