Non-interference on UML State-charts [⋆]

Martín Ochoa[1,2], Jan Jürjens[1], and Jorge Cuéllar[2]

[1] Siemens AG, Germany
[2] TU Dortmund, Germany
{martin.ochoa,jan.jurjens}@cs.tu-dortmund.de
jorge.cuellar@siemens.com

Abstract. Non-interference is a semantically well-defined property that allows one to reason about the security of systems with respect to information flow policies for groups of users. Many of the security problems of implementations could be already spotted at design time if information flow would be a concern in early phases of software development. In this paper we propose a methodology for automatically verifying the interaction of objects whose behaviour is described by deterministic UML State-charts with respect to information flow policies, based on the so-called unwinding theorem. We have extended this theorem to cope with the particularities of state-charts: the use of variables, guards, actions and hierarchical states and derived results about its compositionality. In order to validate our approach, we report on an implementation of our enhanced unwinding techniques and applications to scenarios from the Smart Metering domain.

1 Introduction

Secure Information Flow analysis is a fine-grained methodology for studying the confidentiality and integrity of systems. This kind of analysis (first introduced by Goguen and Meseguer [12] in 1982) is mathematically defined over the inputs and outputs visible to groups of users. Its main advantage over other security analyses is that it allows to pin down subtle flows of information that are difficult to spot when focusing merely on analysing security mechanisms (such as access control mechanisms). Although information flow properties assume perfect access control to guarantee that different groups of users do not see certain inputs and outputs of other users directly, this is a reasonable assumption: attackers usually exploit the information that is shared by victims through common interfaces instead of trying to break directly the access control mechanisms. In words of Anderson [6] "(...) in practice, security is compromised most often not by breaking dedicated mechanisms such as encryption or security protocols, but by exploiting weaknesses in the way they are being used".

[⋆] This research was partially supported by the MoDelSec Project of the DFG Priority Programme 1496 "'Reliably Secure Software Systems – RS^3" and the EU project NESSoS (FP7 256890).

In the past decades different information flow properties have been proposed for coping with different system models such as non-deterministic systems, distributed systems and imperative programming languages. At the abstract level results about compositionality and refinement have been published for many security properties, for example [18,19,24]. In the 'Language Based' realm (i.e. analysis of source code) mature tools for information flow analysis on annotated code exist, like [1,11,2]. All of these are indeed promising steps towards the industrial application of the fine-grained analysis offered by the property-centric point of view of Information Flow. Nevertheless, it seems that production environments are still far from adopting these techniques.

In this paper we propose a light-weight, automatic strategy for checking non-interference on a deterministic fragment of UML state-charts. Our aim is to make a formally sound step towards the usability of these techniques based in the so-called *unwinding* theorem, that provides sufficient conditions for non-interference. We have extended previous work on unwinding to cope with the complexity of UML state-charts: the use variables for keeping history of the state, guards for transitions, hierarchical states and actions. Moreover, we aim at verifying systems where object interaction plays a fundamental role. In order to achieve this, we discuss sufficient conditions for deciding on the composition of the behaviour of already verified components. This is a key factor in the scalability of our approach, which is also an important criterion for the success of verification in realistic settings.

In order to validate our theoretical results, we report on a prototypical machine implementation that automatically verifies models where our unwinding theorem is applicable. We apply this implementation to examples motivated by a case study from the Smart Grid domain. Since unwinding conditions are only sufficient conditions, some secure models might be rejected. However it is important to show that non-trivial secure models are actually accepted and verified. The case study allows to discuss and validate the utility of our approach.

The rest of the paper is organized as follows: Sect. 2 discusses some preliminaries about non-interference and Harel state-charts and sets the notation for the rest of the paper. Sect. 3 is the central section of the paper, were the verification strategy is described. In Sect. 4 we discuss the notion of composition used for reasoning about interacting objects and the composition theorem. In Sect. 5 we illustrate our approach on examples from the Smart Metering domain. Sect. 6 reports on related work and we conclude in Sect. 7.

2 Preliminaries

In this section we recall some definitions and set the notation for the rest of the paper. Starting with the original definition of non-interference by [12] many other subtle information flow properties have been proposed for dealing with non-determinism and distributed systems. In this paper we will nevertheless use the original definition, because our focus will be the analysis of deterministic automata.

2.1 Non-interference

Assume a system is a deterministic black-box transforming sequences of input events I into sequences of output events O by means of a semantic function:

$$[\,] : I \to O$$

We further assume there are two types of users : *high* users H and *low* users [3] L. The sets of input and output events can be divided into the events a high or low user is allowed to see. Lists of input and output events can then be filtered according to the type of user allowed to see them by the purging functions $\cdot|_H$ and $\cdot|_L$. Non-interference is the property :

$$\forall\ \overrightarrow{i}\ \ [\overrightarrow{i}]|_L = [\overrightarrow{i}|_L]|_L \tag{1}$$

In other words, the output seen by the lower users is independent of the input by higher users, up to the point that is not even noticeable whether the high users perform any action on the system.

Some authors (for example [17]) use the equivalent definition:

$$\forall\ \overrightarrow{i_1}, \overrightarrow{i_2}\ \ \overrightarrow{i_1}|_L = \overrightarrow{i_2}|_L \Rightarrow [\overrightarrow{i_1}]|_L = [\overrightarrow{i_2}]|_L \tag{2}$$

which corresponds to the intuition that two runs where the high user perform different actions are equivalent to low users. A stronger version of this property is usually used in the language-based information flow analysis domain [14,7,10].

2.2 State-charts

In order to model the function $[\cdot]$ of the last subsection, consider *Mealy machines* [20]. Syntactically, a Mealy machine can be represented by a directed graph with annotated transitions of the form α/β, meaning that the input event α triggers the output event β. Formally, a Mealy machine M is defined as a 6-tuple $(S, s_0, \Sigma, \Gamma, T, G)$ where S is a finite set of states, s_0 is an initial state, Σ is finite input alphabet, Γ is a finite output alphabet, $T : S \times \Sigma \to S$ is a transition function defined over states and input symbols and $G : S \times \Sigma \to \Gamma$ is an output function defined over states and input symbols. A Mealy machine induces thus a semantics $[\cdot]$ by $[(\sigma_1, \sigma_2, \ldots, \sigma_n)] = G(s_0, \sigma_1) :: \cdots :: G(s_n, \sigma_n)$ where $s_n = T(\ldots T(T(s_0, \sigma_1), \sigma_2) \ldots, \sigma_n)$ and :: denotes concatenation. Notice that the functions T and G are naturally induced by the graph representation. In the following we will assume that if an input is not defined in a given state, the machine enters in a state where no further inputs are processed and no outputs are produced.

[3] For simplicity of exposition and historical reasons we will discuss about high and low users in the rest of the paper. However the definition can be extended to an arbitrary partition of groups of users. Also we will restrict to analysing non-interference from high with respect to low (no-down-flows, usually associated to confidentiality), to analyse the converse (i.e. integrity) one can just switch L for H

If we further divide the input and output events into high an low events, we can apply the definition of non-interference to a Mealy machine.

Example 1. Consider the system defined by the state-machine:

$$\begin{tikzcd}
A \arrow[loop above, "\alpha/\beta"] \arrow[rr, "\gamma/\gamma'"] & & B \arrow[loop above, "\alpha/\beta'"]
\end{tikzcd}$$

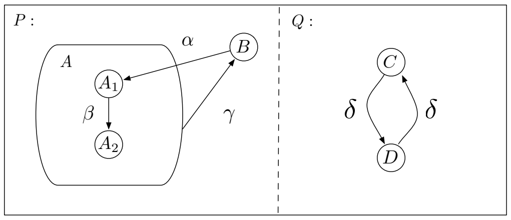
where $\alpha, \beta, \beta' \in L$ and $\gamma, \gamma' \in H$. Then non-interference does not hold since $[(\gamma, \alpha)]|_L = \beta' \neq [(\gamma, \alpha)|_L]|_L = [\alpha]|_L = \beta$.

To deal with the so-called 'state-explosion' problem, that arises when the number of states and transitions increases due to the specification of complex behaviours, other formalisms have been proposed that include the notion of sub and super-states. More prominently Harel [15] proposed the notion of *statechart* that has been used as the basis for UML. This is basically an extension to Mealy machines that allows the following:

– *Hierarchical states:* Single states can contain sub-states and transitions among the sub-states up to arbitrary depth. Let A be a super state containing finitely many sub-states A_i. Then an external state B can have a transition directly to s or to a sub-state A_i. In the first case the transition is to be interpreted as to go to the initial sub-state of s. In the second case it simply goes to s_i. This allows to modularize certain common behaviours into super-states, improving considerably the presentation of complex state charts.
– *Clustering:* To graphically summarize events that trigger a transition to the same state in a group of states, a transition with event γ going out of a super-state A to state B stands for a transition from each sub-state of A with event α to B.
– *Orthogonality and concurrency:* In some cases, processes within the system are orthogonal between each other, in the sense that they could be described with two separate state-charts with disjoint inputs. Thus, Harel state-charts allow multiple sub-state-charts to be modelled as concurrent processes within the same state-chart, compressing notably the notation, since for each two independent Mealy machines with n and m states respectively, $n \cdot m$ states are needed to represent them in a single machine.

Thus formally, a Harel state-chart can be seen as a set (to represent the concurrent processes) of 6-tuples $(S_i, s_0^i, \Sigma_i, \Gamma_i, T_i, G_i)$ where S_i is as a finite set of super-states and s_0^i is an initial super-state. A super-state is defined as a either a state or a state-chart, such that for every super-state there are only finitely many nested state-charts. T_i and G_i are then similar as in the Mealy machine case, where for a given state T_i depends also on the transitions defined at higher hierarchical states (if any).

Example 2. The following state-chart:



contains a (sub) state-chart P containing a superstate with clustering running concurrently with (sub) state-chart Q.

Notice that all these extensions are syntactic sugar for improving the graphical representation: any deterministic Harel state-chart can be represented by a Mealy-Machine with equivalent semantics, and therefore we can use the same definition of non-interference given in the previous subsection for reasoning about the security of Harel State-charts.

3 Extending Unwinding for UML State-charts

Verifying system designs for non-interference is a computationally difficult task, because the definition uses universal quantifiers on inputs and outputs: to verify accurately an arbitrary system implies running and comparing all possible input sequences. Therefore, in order to achieve a trade-off between security and efficiency, one usually needs to sacrifice some precision on the verification. In this section we discuss how to obtain sufficient conditions for the non-interference analysis on UML state-charts by extending traditional unwinding theorems for finite state machines. We will first introduce the fragment of the UML state-charts considered and discuss briefly the unwinding theorem. Then we report on our extension for UML state-charts.

3.1 UML state-charts à la UMLsec

UML has adopted an extension of Harel state-charts to represent the behaviour of classes. It allows a list of *actions* as a consequence of an event, including calling methods, updating variables and outputting values. In this paper we will restrict to a fragment of UML state-charts defined as follows :

- Input events labelling transitions can be either methods of the associated class with concrete parameters or with variables to represent calls with different parameters or global system events (like the tics of a system clock).
- Actions associated to an input event can be either outputting an event (written `return event`) or a variable assignment, where the variables are attributes of the associated class or parameters of the input.
- Guards are decidable conditions on the input parameters or the values of the attributes.

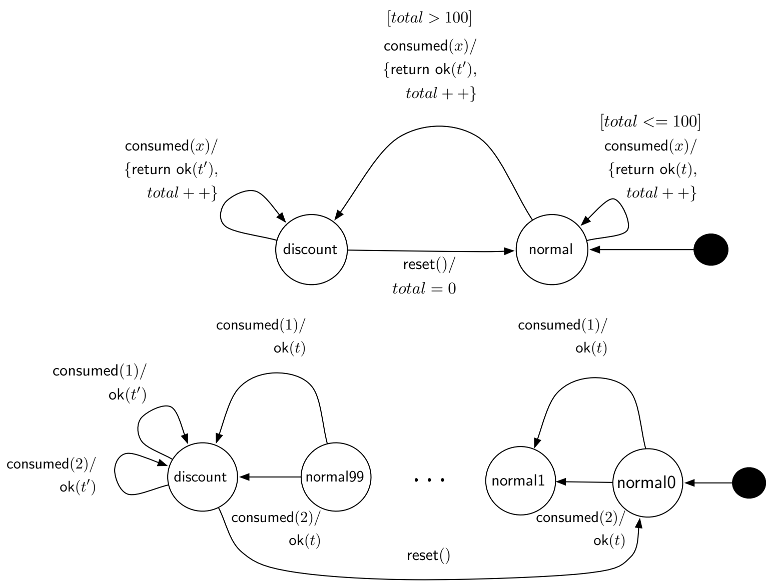


Fig. 1. A model of the smart metering payment processing system as an UML state-chart and a semantically equivalent Mealy Machine for x in range [1..2] in the parameters of consumed.

We will restrict to the sub-set of deterministic UML State-charts as defined above. This is similar to the UML state-charts as defined in [17], with the fundamental difference that there state-charts can be non-deterministic, resulting in a more complex semantics. The semantics of the deterministic fragment defined above can be seen as an extension of the Harel state-chart semantics (based on their Mealy machine translation), where the guards and variables are syntactic sugar for describing the history of the state and where parametrized method calls stand for as many transitions as the respective guard allows.

More precisely: a transition labelled with a parametric input stands for multiple transitions, one for each concrete value of the parameter. Transitions where the actions perform variable assignation stand for multiple transitions with distinct targets (one for each possible value of the assignation). Guards with condition C represent the fact that in states where C hold then that particular labelled transition is present, and not present in states where C does not hold. For an example of a UML state-chart and its semantically equivalent Mealy machine see Fig. 1. We will discuss this example in detail in Sect. 5.

3.2 Unwinding

Unwinding theorems were first proposed by Goguen and Meseguer [13]. They provide sufficient conditions for non-interference that are amenable to verify since they rely on local conditions of pairs of states. The idea is that if there exists a reflexive relation R of the states in an input/output state machine M for a policy dividing events in high H and low L such that:

- R is locally consistent: given a state s and $s' = T(s,h)$ the state resulting after a transition triggered by an event $h \in H$ then $(s,s') \in R$.
- R is step-consistent: for all inputs i, if $(s_1,s_2) \in R$ then $(T(s_1,i),T(s_2,i)) \in R$ where $T(s_1,i)$ and $T(s_2,i)$ are the states resulting from the transition triggered by i in s and t.
- R is output-consistent: if $(s_1,s_2) \in R$ then the output of an event $l \in L$ in s_1 is equal to the output of l in s_2.

then non-interference holds on M for the H and L partition. For a proof see for example [13,26].

Example In example 1, (A,B) must be in R because of locality. However, R is not output consistent and therefore non-interference cannot be established.

3.3 Unwinding for UML Statecharts

There are two main difficulties for extending the unwinding theorems from Mealy machines to the subset of UML state-charts as described in Sect. 3.1: a) The graphical syntactic sugar of Harel state-charts and b) the use of variables and guards for keeping history of the state and for parametrizing inputs. One possibility to verify UML state-charts would be to remove all syntactic sugar, unfold a semantically equivalent Mealy machine and then find an unwinding relation satisfying the conditions described in the previous subsection. This would be however computationally quite expensive in general: the purpose of the UML state-chart notation is to avoid state and transition explosion. We have extended the unwinding theorem accordingly for coping soundly with these differences in the notation in an efficient way. Intuitively, we statically analyse guarded transitions with actions by simultaneously extending an unwinding relation and a tainted set associated to each state. Tainting keeps track of variables whose value is directly or indirectly dependent on high inputs, in the spirit of language based information flow analysis. This information allows to soundly decide on the output consistency of the relation.

In the following when we refer to a state, we mean a state that does not contain further nested sub-states. When we refer to the transitions going out of a state s, we mean all transitions going out of all the super states containing s. Without loss of generality we will analyse concurrent state-charts separately: by definition (Sec. 2) two concurrent Harel state-charts have disjoint inputs, and we further assume they also do not share variables in their UML representation.

Let R' be a relation over the states of a UML state-chart U, H a subset of the inputs of U and $\mathsf{tainted}(s_i)$ a set associated to each state s_i, such that:

Local consistency For a label on the transition t_1 from s_1 to s_2 of the form

$$[C_1]\ \alpha(y_1)\ /\ \{\mathsf{return}\ \beta_1,\ x_1 := E_1\} \tag{3}$$

then s_1 is in relation with the initial sub-state of s_2 if there exists a parameter a such that $\alpha(a) \in H$. Moreover $x_1 \in \mathsf{tainted}(s_2)$ and $\mathsf{tainted}(s_2) \supseteq \mathsf{tainted}(s_1)$.

Step consistency If $(s_1, s_2) \in R'$ then for every transition t_1 of the form (3) with target s'_1 originating from s_1 and every transition t_2:

$$[C_2]\ \alpha(y_2)\ /\ \{\mathsf{return}\ \beta_2,\ x_2 := E_2\} \tag{4}$$

with target s'_2 originating from s_2 then it follows $(s'_1, s'_2) \in R'$. Moreover, if $\alpha(a) \in H$ for some a or there is a variable $z_i \in \mathsf{tainted}(s_i)$ such that $z_i \in C_i$ or $z_i \in E_i$ then $x_i \in \mathsf{tainted}(s'_i)$ and $\mathsf{tainted}(s'_i) \supseteq \mathsf{tainted}(s_i)$.

Output consistency If $(s_1, s_2) \in R'$ with t_1 of form (3) with a such that $\alpha(a) \in L$ we distinguish two cases:

– If there exists x such that $x \in \mathsf{tainted}(s_1)$ and $x \in C_1$ then for all t_2 in s_2 of form (4) it must follow $\beta_1 = \beta_2$.
– Otherwise: if there exists t_2 of form (4) in s_2 such that $C_1 = C_2$ then $\beta_1 = \beta_2$. If no such t_2 exists, then for all other t'_2 in s_2 of form (4) it holds $\beta_1 = \beta_2$.

Moreover there exists no variable x such that $x \in \mathsf{tainted}(s_i)$ and $x \in \beta_i$.

Theorem 1. *If U admits a relation R' as defined above then it respects non-interference.*

Proof. It suffices to show is that the relation R induced by R' on the unfolded Mealy machine M of U is an unwinding relation. A state in M can be seen as a pair $(s, \overrightarrow{v})$ where s is an identifier for a state in U and $\overrightarrow{v}$ is a vector of concrete values $v_1, \ldots, v_n$ for the variables $x_1, \ldots, x_n$ used in U. R is defined thus as $((s_1, \overrightarrow{v}), (s_2, \overrightarrow{w})) \in R \Leftrightarrow (s_1, s_2) \in R'$. Is easy to see that R satisfies local consistency, because R' covers all possible transitions induced by high inputs. Step consistency also holds on R by construction of R'. The extended definition of output consistency is similar to the original one, except for a) it is forbidden to output an expression depending on a tainted variable and b) the output consistency relation is relaxed in case an output is guarded by a condition not depending on tainted variables. It is not hard to see that a) is a necessary condition. Now consider $(s_1, s_2) \in R'$ and w.l.o.g. belonging to the same connected graph of U. Moreover consider a condition C depending on variables $X' = x'_1, \cdots, x'_n$ such that $x'_j \notin \mathsf{tainted}(s_i)$. By the definition of R' there exist ancestors p_1 and p_2 (of s_1 and s_2 respectively) such that there is a high transition between p_1 and p_2 and by definition of tainting this transition does not change the value of any variable in X'. For any input η changing the state of p_1 and p_2 to p'_1 and p'_2 respectively then if $\eta \in H$ then the valuation of X' remains unaltered. If $\eta \in L$ triggers an action changing the value of a variable in X' then the transition was triggered on a condition depending on variables in X'. By hypothesis variables in X' had the same value on p_1 and p_2, and therefore η changes the valuation in both states equivalently. The same reasoning can be done inductively obtaining that the values of X' in s_1 and s_2 depend on the values of X' in p_1 and side-effects triggered by low inputs exclusively. Therefore if C holds in $(s_1, \overrightarrow{v})$ for a given input trace starting on p_1, then it must also hold in $(s_2, \overrightarrow{w})$ for an equivalent trace on the low inputs that reaches s_2.

Notice that it would be also sound to simply compare all outputs in s_1 and s_2, but this would be too coarse for practical uses, where usually a condition an its negation are defined as guards for the same input on a given state, as we will see in Sect. 5. It suffices to compare the outputs guarded by C and not both the outputs of C and those of $\neg C$, because in the unfolded Mealy machine the states where $\neg C$ holds are not necessarily in the minimal unwinding relation.

4 Object interaction

As discussed in Sect. 3.1 UML state-charts are commonly used to represent the behaviour of a class. In the previous section we have discussed unwinding theorems that can be used to decide on the security of single, monolithic state-charts. To reason about the security of a system that is built upon interacting objects, we would need to obtain composed state-charts out of the state-charts defining the single object's behaviour. It would be however desirable to have sufficient conditions that allow to reason on information flow policies on the single components mainly for achieving scalability. In this section we discuss the notion of composition we will use and present sufficient conditions that guarantee that a composition respects non-interference for a given policy.

4.1 Composition

We will follow [17] by reasoning at the instance level: we will assume that the behaviour described by a state-chart is that of an instantiated object [4]. The notion of composition we will use is based on message passing between state-charts: the output messages generated by a state-chart A can be input messages for a state-chart B but not vice-versa. This corresponds formally to a special case of parallel composition as defined for example in [8,21] where we restrict the feedback only to occur in one direction. In other words we do not allow call-backs, which are related to recursive method calls. This is indeed a difficult topic on its own, since subtleties on the semantics play a fundamental role (as discussed for example in [28]), and goes outside the scope of this paper. Nevertheless, the composition notion defined here is useful to reason about the security of non-trivial object interaction, as we will see in Sect. 5, and has nice preservation properties for non-interference.

More precisely, let classes A and B with inputs I_A and I_B respectively and outputs O_A and O_B. A and B are composable if $O_A \cap I_B \neq \emptyset$ and $O_B \cap I_A = \emptyset$. The resulting composed object has inputs $I = I_A \cup (I_B \setminus O_A)$ and outputs $O = O_B \cup (O_A \setminus I_B)$. Semantically, $A \otimes B$ is defined by the product of the states in A and B where the states with at least one output $o \in O_A$ such that $o \in I_B$ cannot be processed by B are discarded, because synchronisation cannot take place. This is similar to the notion parallel composition in CCS [21]. The outputs of transitions of A matching inputs of B in every state are then replaced by the induced outputs in B.

[4] Therefore if we want to reason about different instantiations of an object we would need to define as many classes as desired objects.

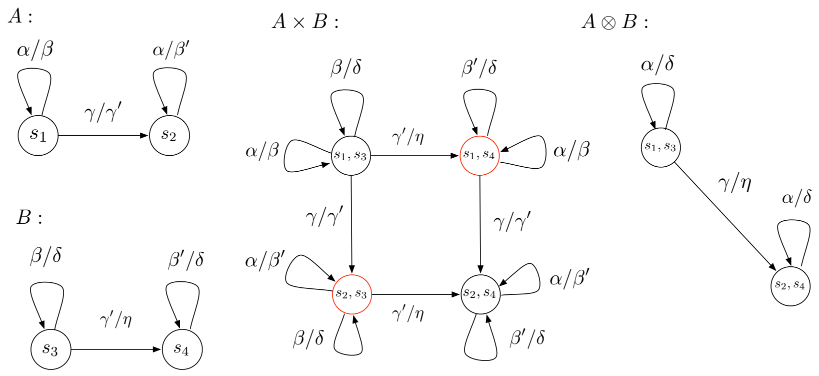


Fig. 2. State-chart U_B and its composition with the state-chart U_A

Example 3. Consider the state-charts A and B of Fig. 2. Although in principle there exists four possible states in the product state-chart $A \times B$ we discard the states where an output of A that is in the interface of B cannot be processed by B.

4.2 Compositionality and non-interference

In general, for scalability reasons, it is desirable that verification results on single components can be re-used efficiently for deciding on their composition. In our setting this means, for a given partition of the set of inputs $I = I|_H \cup I|_L$ and outputs $O = O|_H \cup O|_L$ of the composition $A \otimes B$ there exists sufficient conditions on A and B such that this composition respects non-interference. This is notably not the case in general for information flow properties [18]. However, in our case we can derive a positive result in this sense. We first observe that given a policy on a composition $A \otimes B$, the events on $I_B \cap O_A$ remained unspecified since they are not part of the interface of the composition. Although formally possible, it is not sound from a security point of view to mark events from $I_B \cap O_A$ as high in one component and low in the other (or vice-versa), and therefore we will exclude that possibility in the following.

Theorem 2. *Let $I = I|_H \cup I|_L$ and $O = O|_H \cup O|_L$ a partition of the input and output alphabets of $A \otimes B$. If non-interference holds for an extension of the policy in I and O to the unspecified events in $I_B \cap O_A$ in A and B, then non-interference holds on $A \otimes B$.*

Proof. First consider the case where $O_A = I_B$ (sequential composition). If there exists a sequence $\overrightarrow{i}$ of inputs of $A \otimes B$ such that $[\overrightarrow{i}]_{A\otimes B}|_L \neq [\overrightarrow{i}|_L]_{A\otimes B}|_L$. Then, because of sequentiality and subsequent application of non-interference of B followed by non interference of A and of B again:

$$[\overrightarrow{i}]_{A\otimes B}|_L \overset{S}{=} [[\overrightarrow{i}]_A]_B|_L \overset{B}{=} [[\overrightarrow{i}]_A|_L]_B|_L \overset{A}{=} [[\overrightarrow{i}|_L]_A|_L]_B|_L \overset{B}{=} [[\overrightarrow{i}|_L]_A]_B|_L \quad (5)$$

now observe that:

$$[\overrightarrow{i}|_L]_{A\otimes B}|_L \overset{S}{=} [[\overrightarrow{i}|_L]_A]_B|_L \tag{6}$$

but by hypothesis (5) $\neq$ (6), contradiction.

The other cases follow easily by observing that whenever an input i_B of B is not an output of A then A can be extended by adding a single non connected stated with a transition i'_B/i_B, thus returning to the sequential case and without harming the sufficient conditions (and similarly when output o_A is not an input to B).

Notice that the hypothesis of Theorem 2 although sufficient, are not necessary: in fact, the Example 3 has a component A violating non-interference for $H = \{\gamma, \gamma'\}$, but the composition $A \otimes B$ respects it for $H = \{\gamma, \eta\}$

5 Validation

In this section we report on experiments made to implement the enhanced unwinding technique and the compositionality theorem and applications to examples from our case study.

5.1 Tool support

There are two basic strategies to construct the relation R' on a given state-chart. One possibility is to proceed top down: first put in relation all the states that respect output consistency and then check for local consistency and step consistency. It is however not clear how to proceed from there if the relationship does not respect the unwinding conditions. We have opted to construct it bottom up: first, put every state in relation with itself. Then we compute all relationships due to local consistency, and subsequently for each pair, we enlarge R' by step consistency. When constructing R' we do a preliminary tainting analysis that could be imprecise in presence of loops. This was enough to evaluate our examples, where tainting occurs only in one step (a more accurate tainting analysis are matter of current work). Finally we check for output consistency. If output consistency does not hold, there exists no unwinding relationship, because there exists no minimal one.

We have prototypically implemented the algorithm described in Sect. 3 in Haskell [3] because of its compact and elegant syntax. A state-chart is represented as a pair of list of nodes and list of transitions where the nodes contain the tainting set and a list of 5-tuples:

```
type Node = (Label,Tainted)
type Transition = (Condition, Input, Output, Origin, Target)
type StateChart = (Nodes,Transitions)
```

For example, to check for output consistency of a pair in R' with respect to a set of low inputs `low` we have implemented the following code:

```
compareLowOutput :: StateChart -> Low -> (Node,Node) -> Bool
compareLowOutput (nodes,transitions) low (x,y) =
                 ( null[(tran1,tran2) | (tran1,tran2) <-lowTransitions,
                 (getInputMethod tran1 == getInputMethod tran2),
                 (getReturn tran1 /= getReturn tran2)] )
            where lowTransitions = (getLowTransitions transitions low x y)
```

where `getLowTransitions` is defined as the filtering function including the exception based on the tainting analysis.

5.2 Case study

Smart grids use information and communication technology (ICT) to optimize the transmission and distribution of electricity from suppliers to consumers, allowing smart generation and bidirectional power flows – depending on where generation takes place. With ICT the Smart Grid enables financial, informational, and electrical transactions among consumers, grid assets, and other authorized users[22]. The Smart Grid integrates all actors of the energy market, including the customers, into a system which supports, for instance, smart consumption in cars and the transformation of incoming power in buildings into heat, light, warm water or electricity with minimal human intervention. Smart grid represents a potentially huge market for the electronics industry [27]. The importance of the smart grid for the society is due to the expectation that it will help optimize the use of renewable energy sources [25] and minimize the collective environmental footprint [9]. Two basic reasons why the attack surface is increasing with the new technologies are: a) The Smart Grid will increase the amount of private sensitive customer data available to the utility and third-party partners and b) Introducing new data interfaces to the grid through meters, collectors, and other smart devices create new entry points for attackers.

Among other requirements, confidentiality and privacy of user data is an important security issue. There are many privacy issues, related to the use of sensitive personally identifiable information (PII) related to the consumption of energy, the location of the electric car, etc. This data must be kept secure from unauthorized access, and the measurement process is subject to strict lawful requirements in terms of accuracy, dependability and security, see in particular the European Union directive [4]. See also [16] for a current version of proposed technologies to solve this power systems management and associated information exchange issues. In the following we will model two scenarios in this domain (for details see [23]).

Scenario 1 Consider the behaviour described in Fig. 1. This models an energy provider that processes the amount energy a user x consumes, described by the event $\mathsf{consumed}(x)$ (x is a positive integer, the user's ID) representing one unit of energy consumed. After one unit is consumed, a confirmation $\mathsf{ok}(t)$ with the price t of the consumption is sent to the user. If all consumers of a given region consume more than 100 units, the price of the unit drops from t to t'. Now, assuming that a given user with id 1 is not supposed to know about the consume of other users, we would like to check whether this requirement holds for this system. By setting the events $\{\mathsf{consumed}(x) \mid x \neq 1\}$ as high, we can check whether the unwinding conditions hold automatically via our Haskell implementation. In this case, the model is rejected because of output inconsistency. Because unwinding only provides an approximation, it would still be possible that the system is secure. However, by just seeing a difference in the reported price in two consumptions (given that he does not consume himself more than 100 units), the low user could infer bounds on the number of consumptions of his neighbours. A possibility to obtain a positive security guarantee is to modify the model as follows: a discount is given if a *single* user consumes more than 100 units. By modelling two concurrent machines, one accepting only the inputs of the low user and the other from high, we can positively prove the security of the model.

Scenario 2 In this scenario an electric vehicle buys power from a given provider at an agreed price. It does so at a public recharging station. For convenience, the car will automatically stop the recharging when the total consumption exceeds a given value (for example 10 €) or when the battery capacity k is reached. The behaviour of the single components and their composed model is depicted in Fig. 3, where for illustrative purposes also the composition is spelled out. To fulfil privacy requirements of both users and companies, it is desirable that the recharging stations do not learn the single unit price of the energy sold to the vehicle. In other words, we want to treat the events $\mathsf{setPrice}(x)$, $\mathsf{readPrice}$ and $\mathsf{getPrice}$ as confidential for the charging station. All other events are public. We use then Theorem 2 and proceed to verify the single components. In this case, the behaviour of object V violates non-interference, so we cannot use the compositionality result for $P \otimes V \otimes C$. However, if the recharging policy is modified by not being dependent on the price, thus replacing $[t = 10/p]$ with $[t = k]$, then we can verify the composed model automatically, because all components respect the information flow policy.

6 Related Work

Starting with the work of Goguen and Meseguer [12], many information-flow properties have appeared for specific system models and to capture different notions of security. Rushby discusses unwinding theorems in a more modern notation [26] along with transitive and intransitive information flow policies. General Unwinding theorems for a wide range of information flow properties have also been suggested by Mantel in [13]. Mantel has also unified most of these properties into a common framework, the Modular Assembly Kit for Security MAKS [19], also deriving new unwinding theorems. This work is also probably the best reference for a discussion on the different properties proposed for abstract non-deterministic and distributed system designs.

In the Language-based world, different static approaches have been suggested for verifying information-flow properties, prominently type-based systems like Volpano-Smith [29] or more recently Barthe et al. [7]. Also works based on

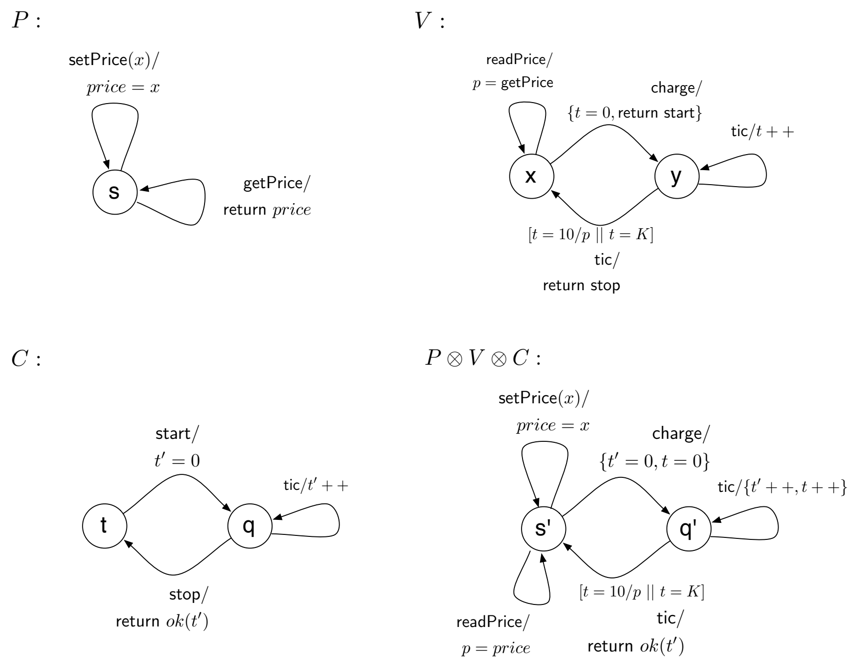


Fig. 3. Single components and composition of the Provider P, the Vehicle V and the Charging station C.

abstract interpretation and analysis of Program Dependency Graphs [10,14] give approximations to non-interference for JavaCard-bytecode and Java respectively. Tools for information flow analysis on annotated code using these techniques are for example Jif [1], JOANA [11] and STAN [2]. Works in the language-based domain, in particular program slicing and tainting, are related with our analysis. However the non-interference property analysed at the code level is generally a stronger property tailored for a non reactive system model, where inputs at the initial program state determine all outputs in all subsequent states.

Jürjens [17] defined a stereotype for non-interference on non-deterministic state-charts that is equivalent to the notion used in this paper for the deterministic case, but no verification strategy or compositionality results are discussed. In [5] Alghathbar et al. model flows of information with UML Sequence diagrams and Horn clauses. However their focus is on high-level information flow policies where only actors and the messages their exchange are modelled, and no explicit relation between the information control rules and a semantic property is given.

7 Conclusions

We have presented an efficient verification strategy for state-charts that is sound with respect to classical non-interference. Our technique is fully automatic and can help to narrow the gap between theory and practice for information-flow in secure-software development in industrial context, by applying our results to a non-trivial subset of UML Statecharts, extending previous work in the area. On a technical level, we have shown how to link unwinding theorems defined in input/output state-machines with verification techniques related to the imperative programming language domain.

In order to validate our approach we have modelled interesting aspects of a Smart Metering scenario where subtle information flows related to confidentiality can be captured by non-interference. These examples show that although approximate, our unwinding theorems are fine-grained enough to verify non-trivial state-charts. We have also prototypically implemented the construction and the verification of the newly defined unwinding relation and unwinding conditions. This implementation allowed us to verify the examples of the case study and discuss about the practical efficiency of the procedure.

There are many directions in which this work could be extended. On the one hand, extensions to other information flow properties for non-deterministic state-machines are interesting to study in the context of UML. On the other hand, a more fine-grained approximation using automatic theorem provers or SMT solvers for evaluating expressions could improve the precision of our analysis. Moreover, studying the preservation of non-interference on code generated from secure UML specifications (refinement) constitutes also a necessary step towards industrial acceptance of these verification techniques.

Acknowledgments The authors would like to thank David von Oheimb and Arnaud Fontaine for useful comments made to an early draft of this paper and Andrei Popescu, Kurt Stenzel and Henning Sudbrock among other researchers from the German Research Foundation RS^3 Priority Programme for interesting discussions on the verification of information flow properties.

References

1. Jif: Java + Information Flow. `http://www.cs.cornell.edu/jif/`.
2. STAN: Information flow analysis for small embedded systems. `http://stan-project.gforge.inria.fr/`.
3. The Haskell Programming Language. `http://www.haskell.org/`.
4. The European Parliament and Council. Measuring instruments directive (2004/22/ec). Official Journal of the EU, 2004.
5. K. Alghathbar, C. Farkas, and D. Wijesekera. Securing UML information flow using flowUML. In *Journal of Research and Practice in Information Technology*, pages 229–238. INSTICC Press, 2006.
6. R. J. Anderson. *Security engineering - a guide to building dependable distributed systems (2. ed.)*. Wiley, 2008.
7. G. Barthe, D. Pichardie, and T. Rezk. A Certified Lightweight Non-interference Java Bytecode Verifier. In *Programming Languages and Systems*. Springer LNCS, 2007.
8. M. Broy. A logical basis for component-oriented software and systems engineering. *Comput. J.*, 53:1758–1782, December 2010.

9. D. Das, F. Kreikebaum, D. Divan, and F. Lambert. Reducing transmission investment to meet renewable portfolio standards using smart wires. In *2010 IEEE PES Transmission and Distribution Conference and Exposition: Smart Solutions for a Changing World*, 2010.
10. D. Ghindici, G. Grimaud, and I. Simplot-Ryl. Embedding verifiable information flow analysis. In *Proc. Annual Conference on Privacy, Security and Trust*, pages 343–352, Toronto, Canada, nov 2006.
11. D. Giffhorn and C. Hammer. Precise Analysis of Java Programs using JOANA (Tool Demonstration). In *8th IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 267–268, September 2008.
12. J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.
13. J. A. Goguen and J. Meseguer. Unwinding and inference control. *IEEE Symposium on Security and Privacy*, 1984.
14. C. Hammer. Information flow control for Java based on path conditions in dependence graphs. In *In IEEE International Symposium on Secure Software Engineering*, 2006.
15. D. Harel. Statecharts: A visual formalism for complex systems, 1987.
16. International Electrotechnical Commission (IEC). IEC 62351 Parts 1-8, Information Security for Power System Control Operations.
17. J. Jürjens. *Secure Systems Development with UML*. Springer, 2005.
18. H. Mantel. On the composition of secure systems. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 88 – 101, 2002.
19. H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2003.
20. G. H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.
21. R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
22. National Energy Technology Laboratory. A vision for the smart grid. Report, June 2009. `http://www.netl.doe.gov/moderngrid/`.
23. Network of Excellence on Engineering Secure Future Internet Software Services and Systems (Nessos). Deliverable 11.2, 2011.
24. D. v. Oheimb. Information flow control revisited: Noninfluence = Noninterference + Nonleakage. In P. Samarati, P. Ryan, D. Gollmann, and R. Molva, editors, *Computer Security – ESORICS 2004*, volume 3193 of *LNCS*, pages 225–243. Springer, 2004.
25. C. W. Potter, A. Archambault, and K. Westrick. Building a smarter smart grid through better renewable energy information. In *2009 IEEE/PES Power Systems Conference and Exposition, PSCE 2009*, 2009.
26. J. Rushby. Noninterference, transitivity and channel-control security policies. Technical report, 1992.
27. R. Schneiderman. Smart grid represents a potentially huge market for the electronics industry. *IEEE Signal Processing Magazine*, 27(5):8–15, 2010.
28. J. Tenzer and P. Stevens. On modelling recursive calls and callbacks with two variants of unified modelling language state diagrams. *Form. Asp. Comput.*, 18:397–420, November 2006.
29. D. Volpano, C. Irvine, and G. Smith. A sound type system for secure flow analysis. *J. Comput. Secur.*, 4:167–187, January 1996.