

# Using Architecture to Reason About Information Security

Stephen Chong  
Harvard University  
chong@seas.harvard.edu

Ron van der Meyden  
University of New South Wales  
meyden@cse.unsw.edu.au

## Abstract

We demonstrate, by a number of examples, that information flow security properties can be proved at an architectural level of abstraction that describes only the causal structure of a system and local properties of a number of trusted components. Architectures are represented using a generalization of intransitive noninterference policies that admits the ability to filter information passed between communicating domains. A notion of refinement of such architectures is developed that supports top-down development of architectural specifications and proofs by abstraction of information security properties.

## 1. Introduction

Architectures are high-level designs that describe the overall structure of a system in terms of its components and their interactions. Proposals for architectural modeling languages (e.g., AADL and Acme [Garlan et al., 2000]) vary with respect to their level of detail and contents, but at the most abstract level, architectures specify the *causal structure of a system*.

The MILS (Multiple Independent Levels of Security and Safety) initiative [Alves-Foss et al., 2006; Vanfleet et al., 2005; Boettcher et al., 2008] of the US Air Force proposes to use architecture as a key part of the assurance case for high-assurance systems. The details of the MILS vision are still under development but, as articulated by Boettcher et al. [2008], it encompasses a 2-level design process, consisting of a policy level and a resource sharing level.

At the policy level, the system is described by an architecture in the form of a graph, in which vertices correspond to components and the edges specify permitted communication between components. In this respect, the architecture is like an *intransitive noninterference* security policy [Haigh and Young, 1987; Rushby, 1992; van der Meyden, 2007]. At the policy level, one might also specify which components are trusted, and the local policies that these components are trusted to enforce. According to the MILS vision, building a system according to the architecture, by composing components that satisfy their local policies, should result in *global security and safety* properties for the system.

At the resource sharing level, MILS envisages the use of a range of infrastructural mechanisms to ensure that the architectural information-flow policy is enforced despite components sharing resources such as processors, file systems, and network links. It is intended that this infrastructure will be developed to a high level of assurance, so that a systems assurance case can be obtained by the composition of the assurance cases for trusted components and systems infrastructure. It is hoped this will enable a COTS-like market for infrastructural mechanisms and trusted components.

In this paper, we investigate the extent to which architectures interpreted with respect to an abstract information-flow semantics can support such compositional derivation of systems properties.

We focus on compositional reasoning about information-flow security properties, and largely confine our attention to the policy level.

Our architecture examples are motivated by systems with interesting security requirements. These include architectures for multi-level secure databases, the Starlight Interactive Link [Anderson et al., 1996], a trusted downgrader, and a simple electronic election system.

In each example, we identify an architectural structure and a mathematically precise set of local constraints on the trusted components. We then show that information-flow properties expressed in a logic of knowledge arise as a consequence of the interaction of the local constraints and the architectural structure. Our results show that for *any* system that is compliant with the architecture, if the trusted components satisfy their local constraints then the system satisfies the global information-flow properties.

The information security properties presented in the examples provide information-theoretic and application-specific guarantees. Thus, there are no covert channels that can violate the information security properties, and the negation of each information security property would constitute an application-specific attack.

To give a precise meaning to the architectural structure, we rely upon the semantics for intransitive noninterference developed by van der Meyden [2007]. An architectural interpretation of this semantics has previously been given [van der Meyden, 2009]. Our examples lead us to extend architectures by labeling edges between components with functions that further restrict the information permitted to flow along edges. One of the contributions of the paper is to give a formal semantics to the enriched architectures that include these new types of edges. We also develop a theory of refinement for these enriched architectures, which enables top-down, correctness-preserving development of architectural specifications. It also enables simple proofs of information security properties on complex architectures to be obtained using an abstraction of that architecture.

The examples studied in the paper provide a demonstration that it is in fact possible, as envisaged in the MILS literature, to derive interesting information security properties compositionally from a high-level specification of trusted components and their architectural structure.

The structure of the paper is as follows. In Section 2 we review architectures and their semantics. In Section 3, we introduce the epistemic logic we use to express information security properties. In Section 4 we extend architectures with *filter functions* that allow fine-grained specification of what information flows between components. The extended architectures enable the proof of additional information security properties. Section 5 extends architectural refinement to account for filter functions. We use examples throughout, to illustrate and motivate. In Section 6, we briefly discuss issues related to the enforcement of architectures, and we discuss related work in Section 7. Section 8 concludes.

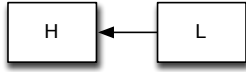
## 2. Architectures and semantics

Architectures give a policy level description of the structure of a system. We begin with a simple notion of architecture, following van der Meyden [2009]. A richer notion will be introduced later.

An architecture is a pair  $\mathcal{A} = (D, \succ)$ , where  $D$  is a set of security domains, and the binary relation  $\succ \subseteq D \times D$  is an *information-flow policy*. The relation  $\succ$  is reflexive but not necessarily transitive. Intuitively, information is allowed to flow from domain  $u$  to domain  $v$  only if  $u \succ v$ . The relation is reflexive as it is assumed that information flow within a domain cannot be prevented, so is always allowed.

### 2.1 Example: $\mathcal{HL}$ architecture

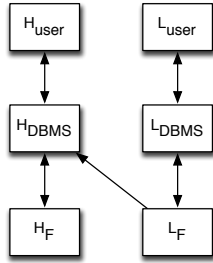
The architecture  $\mathcal{HL} = (\{H, L\}, \{(L, L), (H, H), (L, H)\})$  consists of two security domains  $H$  and  $L$ , and the information-flow policy indicates that information is allowed to flow from  $L$  to  $H$ , in addition to the reflexive information flows. We can depict  $\mathcal{HL}$  graphically, indicating security domains with rectangles, and the information-flow policy with arrows. We omit arrows for reflexive information flows.



### 2.2 Example: Hinke-Schaefer

A variety of architectures have been proposed for multi-level secure database management systems (MLS/DBMS) [Thuraisingham, 2005]. In the Hinke-Schaefer architecture [Hinke and Schaefer, 1975], several (untrusted) single-level DBMSs are composed together in a trusted operating system. Each user interacts with a single-level DBMS. The operating system enforces access control between the single-level DBMSs, allowing more restrictive DBMSs to read the storage files of less restrictive DBMSs, but not vice versa.

The following diagram shows architecture  $\mathcal{HS}$ , which represents the Hinke-Schaefer architecture for two security levels at the MILS policy level.



Domains  $H_{user}$  and  $L_{user}$  represent users of a high-security and low-security DBMS respectively; they interact with the single-level DBMSs  $H_{DBMS}$  and  $L_{DBMS}$  respectively. The single-level DBMSs store their data in database files denoted  $H_F$  and  $L_F$ . Note that information is allowed to flow to  $H_{DBMS}$  from both  $H_F$  and  $L_F$ , as the high-security DBMS is allowed to read the storage files of both the high-security and low-security DBMSs.

### 2.3 Machine model

To specify what it means for an implementation to satisfy an architecture, we must first define what an implementation is. We use the *state-observed* machine model [Rushby, 1992], which defines deterministic state-based machines.

Formally, a machine is a tuple  $M = \langle S, s_0, A, D, \text{step}, \text{obs}, \text{dom} \rangle$  where  $S$  is a set of states,  $s_0 \in S$  is the *initial state*,  $A$  is a set of

*actions*,  $D$  is a set of domains,  $\text{step} : S \times A \rightarrow S$  is a deterministic transition relation,  $\text{dom} : A \rightarrow D$  associates a domain with each action, and observation function  $\text{obs} : D \times S \rightarrow O$  describes for each state what observations can be made by each domain, for some set of observations  $O$ .

We assume that it is possible to execute any action in any state: the function  $\text{step}$  is total. Given sequence of actions  $\alpha \in A^*$ , we write  $s \cdot \alpha$  for the state reached by performing each action in turn, starting in state  $s$ . We define  $s \cdot \alpha$  inductively defined using the transition function  $\text{step}$ , by

$$\begin{aligned} s \cdot \epsilon &= s \\ s \cdot \alpha a &= \text{step}(s \cdot \alpha, a) . \end{aligned}$$

(Here  $\epsilon$  denotes the empty sequence.) For notational convenience, we write  $\text{obs}_u$  for the function  $\text{obs}(u, \cdot)$ , and  $\text{obs}_u(\alpha)$  for  $\text{obs}_u(s_0 \cdot \alpha)$ , where  $\alpha \in A^*$ .

Given a sequence  $\alpha \in A^*$ , the *view* of domain  $u$  of  $\alpha$  is the sequence of  $u$ 's observations and the actions that belong to domain  $u$ . Intuitively,  $u$ 's view is the history of its observations and the actions it has performed. The function  $\text{view}_u$  defines the view of domain  $u$ . It is defined inductively by

$$\begin{aligned} \text{view}_u(\epsilon) &= \text{obs}_u(s_0) \\ \text{view}_u(\alpha a) &= \begin{cases} \text{view}_u(\alpha) a \text{obs}_u(\alpha a) & \text{if } \text{dom}(a) = u \\ \text{view}_u(\alpha) \circ \text{obs}_u(\alpha a) & \text{otherwise} . \end{cases} \end{aligned}$$

The definition uses the absorptive concatenation operator  $\circ$ : for set  $X$ , sequence  $\alpha \in X^*$ , and element  $x \in X$ ,  $\alpha \circ x = \alpha$  if  $x$  is equal to the last element of  $\alpha$ , and  $\alpha \circ x = \alpha x$  otherwise.

Finally, for any sequence of actions  $\alpha \in A^*$ , we write  $\alpha \upharpoonright G$  for the subsequence of  $\alpha$  of actions whose domain is in the set  $G$ .

### 2.4 Semantics

A machine satisfies an architecture if in all possible executions of the machine, information flow is in accordance with the architecture's information-flow policy. We formalize this using an approach proposed by van der Meyden [2007], which involves the use of a concrete operational model to define an upper bound on the information that a domain is permitted to learn.

The operational model is captured using a function  $\text{ta}_u$ , which maps a sequence of actions  $\alpha \in A^*$  to a representation of the maximal information that domain  $u$  is permitted to have after  $\alpha$ , according to the policy  $\succ$ . (The nomenclature "ta" is derived from *transmission* of information about *actions* and the definition corrects problems identified in van der Meyden [2007] with earlier "intransitive purge" based semantics [Rushby, 1992].) An action of  $v$  should convey information to  $u$  only if  $v \succ u$ . Moreover, the information conveyed should be no more than the information that  $v$  is permitted to have. Given machine  $M = \langle S, s_0, A, D, \text{step}, \text{obs}, \text{dom} \rangle$ , function  $\text{ta}_u$  is defined inductively by  $\text{ta}_u(\epsilon) = \epsilon$ , and, for  $\alpha \in A^*$  and  $a \in A$ ,

$$\text{ta}_u(\alpha a) = \begin{cases} \text{ta}_u(\alpha) & \text{if } \text{dom}(a) \not\succ u \\ (\text{ta}_u(\alpha), \text{ta}_{\text{dom}(a)}(\alpha), a) & \text{otherwise} . \end{cases}$$

Note that if information is not allowed to flow from  $\text{dom}(a)$  to  $u$ , then  $\text{ta}_u(\alpha a) = \text{ta}_u(\alpha)$ , i.e., the maximal information permitted to  $u$  does not change. If information is allowed to flow from  $\text{dom}(a)$  to  $u$ , then the information conveyed is at most the information that domain  $\text{dom}(a)$  is permitted to have ( $\text{ta}_{\text{dom}(a)}(\alpha)$ ), and the action  $a$  that was performed. Thus, in this case we add the tuple  $(\text{ta}_{\text{dom}(a)}(\alpha), a)$  to the maximal information  $\text{ta}_u(\alpha)$  that  $u$  was permitted to have before the action  $a$  was performed.

A machine is *TA-compliant* with an architecture if it has an appropriate set of domains, and for each domain  $u$ , what  $u$  ob-

|   |   |
|---|---|
| $M, \pi, \alpha \models \top$             |   |
| $M, \pi, \alpha \models p$                | iff $\alpha \in \pi(p)$   |
| $M, \pi, \alpha \models \neg\phi$         | iff $M, \pi, \alpha \not\models \phi$   |
| $M, \pi, \alpha \models \phi \wedge \psi$ | iff $M, \pi, \alpha \models \phi$ and $M, \pi, \alpha \models \psi$                               |
| $M, \pi, \alpha \models K_u\phi$          | iff $M, \pi, \alpha' \models \phi$ for all<br>$\alpha' \in A^*$ s.t. $\alpha \approx_u \alpha'$   |
| $M, \pi, \alpha \models D_G\phi$          | iff $M, \pi, \alpha' \models \phi$ for all<br>$\alpha' \in A^*$ s.t. $\alpha \approx_G^D \alpha'$ |

**Figure 1.**  $M, \pi, \alpha \models \phi$

serves in state  $s_0 \cdot \alpha$  is determined by  $\mathbf{ta}_u(\alpha)$ . That is,  $\mathbf{ta}_u$  describes the maximal information that  $u$  may learn: if in two runs  $\alpha$  and  $\alpha'$  the maximal information that  $u$  may learn is identical ( $\mathbf{ta}_u(\alpha) = \mathbf{ta}_u(\alpha')$ ), then  $u$ 's observations in each run must be identical ( $\mathbf{obs}_u(\alpha) = \mathbf{obs}_u(\alpha')$ ).

**DEFINITION 1 (TA-compliance).** *System  $M$  is TA-compliant with architecture  $(D, \succ)$  if it has domains  $D$  and for all  $u \in D$  and all sequences  $\alpha, \alpha' \in A^*$  such that  $\mathbf{ta}_u(\alpha) = \mathbf{ta}_u(\alpha')$ , we have  $\mathbf{obs}_u(\alpha) = \mathbf{obs}_u(\alpha')$ .*

### 3. Information security properties

We present a (standard) propositional epistemic logic [Fagin et al., 1995] that we use to express information security properties. The syntax is defined as follows:

$$\phi, \psi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \psi \mid K_u\phi \mid D_G\phi$$

$G$  ranges over groups of domains .

Formulas  $\top$ ,  $p$ ,  $\neg\phi$ , and  $\phi \wedge \psi$  are standard from propositional logic:  $\top$  is always satisfied, and  $p$  is a propositional constant. Epistemic formula  $K_u\phi$  says that domain  $u$  knows  $\phi$ , and  $D_G\phi$  says that  $\phi$  is *distributed knowledge* to the group of domains  $G$ , i.e.,  $\phi$  could be deduced if the information of all domains in  $G$  were combined.

Formulas are interpreted using a possible worlds semantics, where a world is a sequence of actions  $\alpha \in A^*$ . A *proposition* is a set  $X \subseteq A^*$ . We say proposition  $X$  is *non-trivial* if  $X \neq \emptyset$  and  $X \neq A^*$ . An *interpretation function*  $\pi$  is a function from propositional constants to propositions.

We define the semantics of the logic using satisfaction relation  $M, \pi, \alpha \models \phi$ , which intuitively means that formula  $\phi$  is true given interpretation function  $\pi$ , and machine  $M$  that has executed sequence  $\alpha \in A^*$ . Figure 1 defines relation  $M, \pi, \alpha \models \phi$ . We write  $M, \pi \models \phi$  if for all  $\alpha \in A^*$  we have  $M, \pi, \alpha \models \phi$ .

To interpret epistemic formulas, we use an indistinguishability relation for each domain  $u$  that describes what sequences of actions  $u$  considers possible given its view of the actual sequence of actions. Two sequences of actions  $\alpha \in A^*$  and  $\alpha' \in A^*$  are indistinguishable to domain  $u$ , written  $\alpha \approx_u \alpha'$ , if  $u$ 's view of the two sequences are identical:

$$\alpha \approx_u \alpha' \iff \mathbf{view}_u(\alpha) = \mathbf{view}_u(\alpha').$$

We interpret distributed knowledge using an indistinguishability relation  $\approx_G^D$  defined as the intersection of  $\approx_u$  for  $u \in G$ .

A proposition is  $G$ -action-local if the actions of domains in the group  $G$  suffice to decide the proposition. Formally, for a group  $G$ , a set  $X \subseteq A^*$  is a  $G$ -action-local proposition if for all  $\alpha, \alpha' \in A^*$  if  $\alpha \upharpoonright G = \alpha' \upharpoonright G$ , then  $\alpha \in X \iff \alpha' \in X$ . We write “ $u$ -action-local” when  $G = \{u\}$ .

#### 3.1 Example: $\mathcal{HL}$ information security

The logic allows us to state information security properties about machines, in terms of the knowledge of domains. The architecture

can provide sufficient structure to prove that a given information security property holds in all machines that comply with the architecture.

For example, using the  $\mathcal{HL}$  architecture, we are able to show that in any execution of any machine that complies with  $\mathcal{HL}$ , the domain  $L$  does not know any non-trivial  $H$ -action-local proposition.

**THEOREM 1.** *If  $M$  is TA-compliant with  $\mathcal{HL}$  and  $\pi(p)$  is a non-trivial  $H$ -action-local proposition then  $M, \pi \models \neg K_L p$ .*

#### 3.2 Example: Hinke-Schaefer

In the Hinke-Schaefer database architecture  $\mathcal{HS}$ , none of the domains  $L_{user}$ ,  $L_{DBMS}$ , or  $L_F$  know anything about the domains  $H_{user}$ ,  $H_{DBMS}$  or  $H_F$ . This is true even if we consider the distributed knowledge of  $L_{user}$ ,  $L_{DBMS}$ , and  $L_F$ .

**THEOREM 2.** *Let system  $M$  be TA-compliant with  $\mathcal{HS}$ , and let  $G = \{L_{user}, L_{DBMS}, L_F\}$ . If  $\pi(p)$  is a non-trivial  $\{H_{user}, H_{DBMS}, H_F\}$ -action-local proposition, then  $M, \pi \models \neg D_G p$ .*

Since  $K_u p \Rightarrow D_G p$  is valid for  $u \in G$ , it follows that also  $M, \pi \models \neg K_u p$  for  $u \in \{L_{user}, L_{DBMS}, L_F\}$ . In particular,  $L_{user}$  does not have any information about the High side of the system.

### 4. Extended architecture

The architectures used so far impose coarse, global constraints on the causal structure of systems. If  $u \succ v$  then TA-compliance permits domain  $u$  to send to domain  $v$  any and all data it has. However, in many systems, key security properties depend on the fact that trusted components allow only certain information to flow from one domain to another. Finer specification of information flows in the architecture allow us to prove stronger information security properties.

In this section we extend the notion of architecture by introducing *filter functions* to allow fine-grained specification of what information flows between domains. We define semantics for these extended architectures, and present examples where the extended architectures allow us to prove strong information security properties.

#### 4.1 Filter functions

An extended architecture is a pair  $\mathcal{A} = (D, \succ)$ , where  $D$  is a set of security domains, and  $\succ \subseteq D \times D \times (\mathcal{L} \cup \{\top\})$ , where  $\mathcal{L}$  is a set of function names. We write  $u \xrightarrow{f} v$  when  $(u, v, f) \in \succ$ , and write  $u \succ v$  as shorthand for  $\exists f. (u, v, f) \in \succ$ , and  $u \not\rightarrow v$  as shorthand for  $\neg \exists f. (u, v, f) \in \succ$ .

Intuitively,  $u \xrightarrow{f} v$  represents that information flow from  $u$  to  $v$  is permitted, but may be subject to constraints. In case  $f = \top$ , there are no constraints on information flow from  $u$  to  $v$ : any information that may be possessed by  $u$  is permitted to be passed to  $v$  when  $u$  acts, just as in the definition of TA-compliance. If  $f \in \mathcal{L}$  then information is allowed to flow from domain  $u$  to domain  $v$ , but it needs to be *filtered* through the function denoted by  $f$ : only information output by this function may be transmitted from  $u$  to  $v$ . If  $u \not\rightarrow v$  then no direct flow of information from  $u$  to  $v$  is permitted.

In some cases, it may be possible for the operating system or network infrastructure to enforce a given filter function. However, in general, a filter function is a local constraint on a trusted component of the system. That is, if  $u \xrightarrow{f} v$  for  $f \neq \top$ , then component  $u$  is trusted to enforce that information sent to  $v$  is filtered appropriately.

We require that extended architectures have the following properties:

1. For all  $u, v \in D$ , there exists at most one  $f \in \mathcal{L} \cup \{\top\}$  such that  $u \xrightarrow{f} v$ .
2. The relation  $\succrightarrow$  is reflexive in that for all  $u \in D$  we have  $(u, u, \top) \in \succrightarrow$ .

The first condition requires that all permitted flows of information from  $u$  to  $v$  are represented using a single labeled edge. Intuitively, any policy with multiple such edges can always be transformed into one satisfying this condition, by combining the pieces of information flowing across these edges into a tuple that flows across a single edge. The second condition is motivated from the fact, already noted above, that information flow from a domain to itself cannot be prevented.

Extended architectures do not define the interpretations of the function names  $\mathcal{L}$ . If  $\mathcal{A} = (D, \succrightarrow)$  is an extended architecture, an *interpretation* for  $\mathcal{A}$  is a tuple  $\mathcal{I} = (A, \text{dom}, \mathbf{I})$ , where  $A$  is a set of actions,  $\text{dom} : A \rightarrow D$  assigns these actions to domains of  $\mathcal{A}$ , and  $\mathbf{I}$  is a function mapping each  $f \in \mathcal{L}$  to a function with domain  $A^*$  (and arbitrary codomain). We call the pair  $(\mathcal{A}, \mathcal{I})$  an *interpreted (extended) architecture*.

Intuitively, if  $u \xrightarrow{f} v$  and  $\alpha \in A^*$  and  $a \in A$  is an action with  $\text{dom}(a) = u$ , then  $\mathbf{I}(f)(\alpha a)$  is the information that is permitted to flow from  $u$  to  $v$  when the action  $a$  is performed after occurrence of the sequence of actions  $\alpha \in A^*$ . Based on this intuition, we may define a function  $\text{fta}_u$  with domain  $A^*$  that, like  $\text{ta}_u$ , captures that maximal information that domain  $u$  is permitted to have after a sequence of actions has been executed.

Given extended architecture  $(D, \succrightarrow)$  and an architectural interpretation  $\mathcal{I} = (A, \text{dom}, \mathbf{I})$ , the function  $\text{fta}_u$  is defined inductively by  $\text{fta}_u(\epsilon) = \epsilon$ , and, for  $\alpha \in A^*$  and  $a \in A$ ,

$$\text{fta}_u(\alpha a) = \begin{cases} \text{fta}_u(\alpha) & \text{if } \text{dom}(a) \not\rightarrow u \\ \text{fta}_u(\alpha) (\text{fta}_{\text{dom}(a)}(\alpha), a) & \text{if } \text{dom}(a) \xrightarrow{\top} u \\ \text{fta}_u(\alpha) \mathbf{I}(f)(\alpha a) & \text{if } \text{dom}(a) \xrightarrow{f} u. \end{cases}$$

The first two clauses resemble the definition of  $\text{ta}_u$ ; the third adds to this that the information flowing along an edge labeled by a function name  $f$  is filtered by the interpretation  $\mathbf{I}(f)$ . Note that if  $\mathbf{I}(f)(\alpha a) = \epsilon$ , where  $\text{dom}(a) \xrightarrow{f} u$ , then  $\text{fta}_u(\alpha a) = \text{fta}_u(\alpha)$ . That is, filter function  $\mathbf{I}(f)$  can specify that no information should flow under certain conditions. Note also that  $\text{fta}_u$  has implicit parameters, viz., an information flow policy  $\succrightarrow$  and an architectural interpretation  $\mathcal{I} = (A, \text{dom}, \mathbf{I})$ . When we need to make some of these parameters explicit, we write expressions such as  $\text{fta}_u^{(\succrightarrow, \mathcal{I})}$ , or  $\text{fta}_u^{\rightarrow}$ .

The function  $\text{fta}_u$  is used analogously to  $\text{ta}_u$  to define the maximal information that a domain is permitted to observe for a given sequence of actions. However,  $\text{fta}_u$  is a more precise bound than  $\text{ta}_u$ , as it uses filter functions to bound the information sent between domains.

It is reasonable to assume that information sent from  $u$  to  $v$  is information that  $u$  is permitted to have. We say that a function  $h$  with domain  $A^*$  is  $\text{fta}_u$ -compatible when for all sequences  $\alpha, \beta \in A^*$ ,  $\text{fta}_u(\alpha) = \text{fta}_u(\beta)$  implies that for all  $a \in A$  with  $\text{dom}(a) = u$  we have  $h(\alpha a) = h(\beta a)$ . We say that the interpretation  $\mathcal{I} = (A, \text{dom}, \mathbf{I})$  is *compatible* with  $\mathcal{A} = (D, \succrightarrow)$  if for all  $u \in D$  and edges  $u \xrightarrow{f} v$  with  $f \in \mathcal{L}$ , the function  $\mathbf{I}(f)$  is  $\text{fta}_u$ -compatible. In what follows, we require that interpretations be compatible with their architectures.

A machine complies with an interpreted extended architecture if it has appropriate domains and actions, and for each domain  $u$ , what  $u$  observes in state  $s_0 \cdot \alpha$  is determined by  $\text{fta}_u(\alpha)$ . We call such a machine *FTA-compliant*. (“FTA” is derived from *filtered transmission* of information about actions.)

**DEFINITION 2 (FTA-compliant).** A machine  $M = (S, s_0, A, D, \text{step}, \text{obs}, \text{dom})$  is *FTA-compliant* with an interpreted architecture  $(\mathcal{A}, \mathcal{I})$ , with  $\mathcal{A} = (D', \succrightarrow)$  and  $\mathcal{I} = (A', \text{dom}', \mathbf{I})$ , if  $A = A'$ ,  $D = D'$ ,  $\text{dom} = \text{dom}'$  and for all agents  $u \in D$  and all  $\alpha, \alpha' \in A^*$ , if  $\text{fta}_u(\alpha) = \text{fta}_u(\alpha')$  then  $\text{obs}_u(\alpha) = \text{obs}_u(\alpha')$ .

Separating extended architectures from their interpretations ensures that extended architectures can be completely represented by graphical diagrams with labeled edges. It also allows us to deal with examples where an extended architecture can be implemented in a variety of ways, and weak constraints on the set of actions and the set of filter functions suffice to enforce the security properties of interest. We will present a number of examples of this in what follows. To capture the constraints on the architectural interpretations at the semantic level, we use the notion of an *architectural specification*, which is a pair  $(\mathcal{A}, \mathcal{C})$  where  $\mathcal{A}$  is an extended architecture and  $\mathcal{C}$  is a set of architectural interpretations for  $\mathcal{A}$ . (We will not attempt in this paper to develop any syntactic notation for architectural specifications.)

**DEFINITION 3.** A machine  $M$  is *FTA-compliant* with an architectural specification  $(\mathcal{A}, \mathcal{C})$  if there exists an interpretation  $\mathcal{I} \in \mathcal{C}$  such that  $M$  is *FTA-compliant* with the interpreted architecture  $(\mathcal{A}, \mathcal{I})$ .

When drawing extended architectures, we annotate arrows between domains with the filter function names. For arrows drawn without a label, and elided reflexive arrows, the implied label is  $\top$ .

The following theorem shows that FTA-compliance generalizes TA-compliance. Thus, we are free to interpret a given architecture as an extended architecture.

**THEOREM 3.** Let  $\mathcal{A}_1 = (D, \succrightarrow_1)$  be an architecture, and let  $\mathcal{A}_2 = (D, \succrightarrow_2)$  be the extended architecture such that  $(u, v, f) \in \succrightarrow_2$  if and only if  $f = \top$  and  $(u, v) \in \succrightarrow_1$ . Let  $M$  be a machine with domains  $D$ , actions  $A$  and domain function  $\text{dom}$ . Let  $\mathcal{I} = (A, \text{dom}, \mathbf{I})$  be any interpretation for  $\mathcal{A}_2$  with this set of actions and domain function. Then  $M$  is *TA-compliant* with  $\mathcal{A}_1$  if and only if  $M$  is *FTA-compliant* with  $(\mathcal{A}_2, \mathcal{I})$ .

Whereas Theorem 3 states that architectures correspond to extended architectures in which all edges are labeled by  $\top$ , the opposite case, of extended architectures in which no edges are labeled by  $\top$ , also proves to be useful. We say that such extended architectures are *fully filtered*. Fully filtered extended architectures have all the expressive power of extended architectures. Given an interpreted extended architecture  $\mathcal{AI} = (\mathcal{A}_1, \mathcal{I}_1)$ , we may define the fully filtered interpreted extended architecture  $\mathcal{AI}^\dagger = (\mathcal{A}_2, \mathcal{I}_2)$  by replacing each edge from  $u$  labeled  $\top$  by an edge with a new function name  $f_u$  interpreted as  $\mathbf{I}(f_u)(\alpha a) = (\text{fta}_u(\alpha), a)$ , so that the following result holds.

**THEOREM 4.** For every interpreted extended architecture  $\mathcal{AI}$  and system  $M$ ,  $M$  is *FTA-compliant* with  $\mathcal{AI}$  if and only if  $M$  is *FTA-compliant* with  $\mathcal{AI}^\dagger$ .

Theorem 4 shows that there is no loss of generality in assuming that architectures are fully filtered. We note, however, that there are several significant reasons to use edges labelled  $\top$ . One is that whereas architectures that use only  $\top$ -labelled edges (i.e., the unextended notion of architecture) can be completely enforced at the resource sharing level using mechanisms such as separation kernels, physical separation, and data diodes, an edge from  $u$  to  $v$  labeled by a filter function  $f$  with interpretation  $\mathbf{I}(f)$  introduces an obligation to verify that the implementation of  $u$  releases no more information to  $v$  than specified by  $\mathbf{I}(f)$ . Thus, in such a case  $u$  is a *trusted component* in the architecture. Maximizing the number of edges labeled  $\top$  therefore reduces the number of

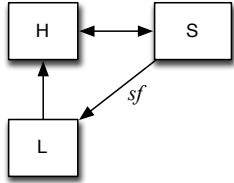
trusted components. Even for specification purposes, edges labeled  $\top$  remain a significant convenience, because the complex mutual recursion in  $\text{fta}_u^{(A_1, \mathcal{I}_1)}$  is required to define the interpretation  $\mathcal{I}_2$ .

## 4.2 Example: Starlight Interactive Link

The Starlight Interactive Link [Anderson et al., 1996] provides interactive access from a high-security network to a low-security network. This allows a user on the high-security network to have windows open on her screen at differing security levels, while ensuring no high-security information goes to the low-security network.

Starlight has both hardware and software components. The hardware device is connected to both the high-security and low-security networks, and has a keyboard and mouse attached. There is a switch that can toggle between the high-security and low-security networks; input from the mouse and keyboard are sent to the network currently selected by the switch. Starlight allows data from the low-security network to be transferred to the high-security network, but not vice versa. The Starlight software components include proxy window clients and servers to allow the windowing environment to work in the presence of the Starlight hardware.

The following diagram shows extended architecture  $\mathcal{S}\mathcal{L}$ , an architecture for the Starlight Interactive Link.



Domain  $H$  represents the high-security network (including the user's computer); domain  $L$  represents the low-security network; domain  $S$  represents the Starlight Interactive Link (including input devices), which routes keyboard and mouse events to either the high-security or low-security network. Note that there is no edge from domain  $L$  to domain  $S$ , as no information is sent directly from the low-security network to the Starlight Interactive Link. Instead, data from the low-security network (such as updates to the contents of a window) are sent to the high-security network, and thence to software components of the Starlight Interactive Link.

The edge labeled  $sf$  restricts what information is allowed to flow from the Starlight Interactive Link to low-security network  $L$ . We present an architectural specification  $\mathcal{C}_{\mathcal{S}\mathcal{L}}$  based on  $\mathcal{S}\mathcal{L}$  that expresses a constraint on interpretations of  $sf$ . An interpretation  $(A, \text{dom}, \mathcal{I})$  is included in  $\mathcal{C}_{\mathcal{S}\mathcal{L}}$  if the following conditions hold. Let  $A_S = \{a \in A \mid \text{dom}(a) = S\}$  be the set of actions belonging to domain  $S$ . We assume that there is a distinguished action  $t \in A_S$  that toggles which network is receiving the input events. For all  $i$ , let  $\alpha_i \in A^*$  be a sequence of actions that does not contain  $t$ . Intuitively,  $L$  may know about the occurrence of any  $t$  action, and the occurrence of any other action in  $A_S$  (e.g., keyboard or mouse input) that happens while the low-security network is selected (i.e., after an odd number of toggle actions). We capture this by the following assumption on interpretation  $\mathcal{I}$ :

$$\mathcal{I}(sf)(\alpha_0 t \alpha_1 t \dots t \alpha_k a) = \begin{cases} a & \text{if } a = t \text{ or} \\ & k \text{ is odd and } a \in A_S \\ \epsilon & \text{otherwise} \end{cases}$$

It is straightforward to check for such an interpretation that  $\mathcal{I}(sf)$  is  $\text{fta}_S$ -compatible.

The component  $S$ , corresponding to the Starlight Interactive Link, is a trusted component, and the  $sf$  filter is a local constraint on the component. To verify that a system satisfies specification

$(\mathcal{S}\mathcal{L}, \mathcal{C}_{\mathcal{S}\mathcal{L}})$ , we would need to verify that  $S$  appropriately filters information sent to  $L$ , and that all other communication in the system complies with the architecture, to wit, that  $H$  cannot communicate directly with  $L$ , and  $L$  cannot communicate directly with  $S$ .

If a system does satisfy specification  $(\mathcal{S}\mathcal{L}, \mathcal{C}_{\mathcal{S}\mathcal{L}})$ , then we can show that domain  $L$  never knows any  $H$ -action-local proposition. Indeed, we can show something stronger, that  $L$  never knows any proposition about  $H$  actions and  $S$  actions that occur while the high-network is selected.

We call proposition  $X$  *toggle- $H$  dependent* if membership in  $X$  depends only on the subsequence of actions consisting of (1) all actions  $a$  with  $\text{dom}(a) = H$ , and (2) all actions  $a$  with  $\text{dom}(a) = S$  that occur after an even numbered occurrence of  $t$  and before any subsequent occurrence of  $t$ . That is,  $X$  is toggle- $H$  dependent if it depends only on  $H$  and  $S$  actions that occur while the high-network is selected. Note that if  $X$  is  $H$ -action-local, then  $X$  is toggle- $H$  dependent.

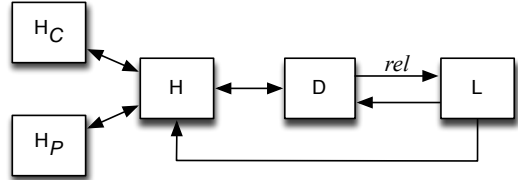
**THEOREM 5.** *Let system  $M$  be FTA-compliant with architectural specification  $(\mathcal{S}\mathcal{L}, \mathcal{C}_{\mathcal{S}\mathcal{L}})$ . If  $\pi(p)$  is a non-trivial toggle- $H$  dependent proposition, then  $M, \pi \models \neg K_L p$ .*

Note that we would not be able to show this security property in an architecture without filter functions, since the domain  $S$  can communicate with both  $H$  and  $L$ . It is the filter function that allows us to show that  $S$ 's communication with  $L$  reveals nothing about  $H$  actions, and only limited information about  $S$  actions.

## 4.3 Example: Downgrader

Many systems downgrade information, releasing confidential information to untrusted entities. Because it is a sensitive operation, downgrading is typically restricted to certain trusted components, called downgraders.

The following diagram shows extended architecture  $\mathcal{D}\mathcal{G}$ , containing low-security domain  $L$ , high-security domain  $H$  with two high-security users  $H_C$  and  $H_P$ , and downgrader  $D$ .



This architecture represents a system where some information is allowed to flow from the high-security domain  $H$  to low-security domain  $L$ , but only through the downgrader  $D$ . In particular, we assume that there are two high-security users,  $H_C$  and  $H_P$ , and domain  $L$  is permitted to know about the actions of  $H_P$  (via downgrader  $D$ ), but should never know anything about the actions of  $H_C$ . This may be an appropriate model for a government agency where some sensitive information may be released to the public, but certain sensitive data (e.g., the identity of covert employees) should never be released.

The filter function for  $rel$  restricts the information that may be released from  $D$  to  $L$ . It is a local constraint on the trusted component  $D$ . The filter function should ensure that nothing is revealed about the actions of  $H_C$ . We define an architectural specification  $\mathcal{C}_{\mathcal{D}\mathcal{G}}$  to restrict our attention to architecture interpretations with suitable filter functions.

Let  $\mapsto'$  be the information flow policy on  $D$  obtained by restricting policy  $\mapsto$  to the domains  $H_P, H$ , and  $D$ . Thus  $(u, v, f) \in \mapsto'$  if and only if either  $(u, v, f) \in \mapsto$  and  $\{u, v\} \subseteq \{H_P, H, D\}$  or  $u = v$ .

Let  $(A, \text{dom}, \mathcal{I}) \in \mathcal{C}_{\mathcal{D}\mathcal{G}}$  if and only if  $(A, \text{dom}, \mathcal{I})$  is an architecture interpretation of  $\mathcal{D}\mathcal{G}$  such that  $\mathcal{I}(rel) = \text{fta}_D^{\mapsto'}$ . Thus, archi-

ecture interpretations in  $\mathcal{C}_{\mathcal{D}\mathcal{G}}$  ensure that  $\text{fta}_{\vec{D}}$  is an upper-bound on information that may be released from  $D$  to  $L$ .

The architectural specification imposes restrictions on what information  $L$  knows and when. Domain  $L$  can only learn about other domains via downgrader  $D$ . The restriction on filter functions for  $\text{rel}$  ensures that  $L$  cannot know anything about the actions of  $H_C$ , since  $H_C \not\rightarrow' H$ , and so the function  $\text{fta}_{\vec{D}}$  does not contain any information about the actions of  $H_C$ . However, information about the actions of  $H$ ,  $H_P$  and  $D$  may be released to  $L$ . Also,  $L$  cannot learn about actions of other domains that occurred after the last  $D$  action. The following two theorems express these restrictions.

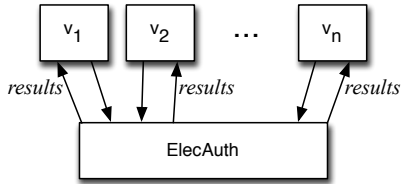
**THEOREM 6.** *If  $M$  is FTA-compliant with architectural specification  $(\mathcal{D}\mathcal{G}, \mathcal{C}_{\mathcal{D}\mathcal{G}})$  and  $\pi(p)$  is a non-trivial  $H_C$ -action-local proposition then  $M, \pi \models \neg K_{LP}$ .*

To state that  $L$  knows nothing about any action of other domains that occurred after the last  $D$  action, we say that proposition  $X$  is about  $H_C$ ,  $H_P$ , and  $H$  activity after the last  $D$  action if there exists a set  $Y \subseteq A^*$  such that for all  $\alpha \in A^*$ ,  $\alpha \in X \iff (F(\alpha) \upharpoonright \{H_C, H_P, H\}) \in Y$ , where  $F(\alpha)$  is the longest suffix of  $\alpha$  that does not contain any  $D$  actions. Intuitively,  $F(\alpha) \upharpoonright \{H_C, H_P, H\}$  is the  $H_C$ ,  $H_P$ , and  $H$  actions that occur after the last  $D$  action, and so proposition  $X$  is about  $H_C$ ,  $H_P$ , and  $H$  activity after the last  $D$  action if membership in  $X$  depends only on the  $H_C$ ,  $H_P$ , and  $H$  actions that occur after the last  $D$  action.

**THEOREM 7.** *If  $M$  is FTA-compliant with architectural specification  $(\mathcal{D}\mathcal{G}, \mathcal{C}_{\mathcal{D}\mathcal{G}})$  and  $\pi(p)$  is a non-trivial proposition about  $H_C$ ,  $H_P$ , and  $H$  activity after the last  $D$  action, then  $M, \pi \models \neg K_{LP}$ .*

#### 4.4 Example: Electronic election

The following diagram shows architecture  $\mathcal{E}\mathcal{E}$ , an electronic election for  $n$  voters coordinated by election authority  $\text{ElecAuth}$ .



In many elections, the behavior of individual voters is sensitive information: it should not be known how voters voted.

By specifying an additional local constraint on the election authority, we can show that this architecture enforces anonymity on the identity of the voters. The election authority's compliance with this local constraint might be assured by means of a careful verification of its implementation, or carefully designed cryptographic protocols (which may remove some or all of the trust required to be placed in the election authority).

We first assume that voters are homogenous in that they have the same set of actions. Let  $(\mathcal{E}\mathcal{E}, \mathcal{I})$  be an interpreted architecture where  $\mathcal{I} = (A, \text{dom}, \mathbf{I})$ . We say that architecture interpretation  $\mathcal{I}$  is *voter homogenous* if for any voter  $v$ , the set of possible actions  $A_v = \{a \in A \mid \text{dom}(a) = v\}$  equals  $\{a^v \mid a \in A_v\}$ , where  $A_v$  is the set of actions available to voters.

We define a voter permutation  $P$  as a permutation over the set of voters  $V$ . Since all voters have the same set of actions in a voter-homogenous interpretation, we can apply a permutation  $P$  to sequence  $\alpha \in A^*$ , written  $P(\alpha)$ , as follows:

$$\begin{aligned}
 P(\epsilon) &= \epsilon \\
 P(\alpha a) &= P(\alpha) a && \text{if } \text{dom}(a) \notin V \\
 P(\alpha a^v) &= P(\alpha) a^{P(v)} && \text{if } \text{dom}(a) \in V.
 \end{aligned}$$

We apply permutation  $P$  to proposition  $X \subseteq A^*$  by applying  $P$  to each sequence  $\alpha \in X$ , i.e.,

$$P(X) = \{P(\alpha) \mid \alpha \in X\}.$$

Intuitively, if  $X$  is a  $u$ -action-local proposition, for  $u \in V$ , then  $P(X)$  is a  $P(u)$ -action-local proposition.

Using voter permutations, we can now state the local constraint that election results do not depend on (and thus do not reveal) the identity of any voter.

**A1.** Election results have the following *identity-oblivious* property: Given voter-homogenous interpreted architecture  $(\mathcal{E}\mathcal{E}, \mathcal{I})$  where  $\mathcal{I} = (A, \text{dom}, \mathbf{I})$ , for all voter permutations  $P$ , and sequences  $\alpha \in A^*$ ,  $\mathbf{I}(\text{results})(\alpha) = \mathbf{I}(\text{results})(P(\alpha))$ .

There are several possible interpretations of *results* that satisfy this constraint, such as a function that returns each candidate and her vote tally, or a function that returns the total number of votes submitted. However, a function that returns a ballot and the identity of the voter that submitted it doesn't satisfy constraint A1.

We define architecture specification  $\mathcal{C}_{\mathcal{E}\mathcal{E}}$  such that  $\mathcal{I} \in \mathcal{C}_{\mathcal{E}\mathcal{E}}$  if and only if  $\mathcal{I}$  is a voter-homogenous interpretation of  $\mathcal{E}\mathcal{E}$  that satisfies constraint A1.

Given the local constraint A1, we can show that if voter  $v$  believes that some proposition  $X$  may be satisfied, then  $v$  also believes that  $P(X)$  may be satisfied, for voter permutation  $P$ . For example, if Alice considers it possible that Bob voted for Obama and Charlie voted for McCain, then Alice considers it possible that Charlie voted for Obama and Bob voted for McCain.

**THEOREM 8.** *Let system  $M$  be FTA-compliant with  $(\mathcal{E}\mathcal{E}, \mathcal{C}_{\mathcal{E}\mathcal{E}})$ . Let  $v$  be a voter. For all voter permutations  $P$  such that  $P(v) = v$ , if  $\pi'(p) = P(\pi(p))$ , and  $M, \pi, \alpha \models \neg K_v \neg p$  then  $M, \pi', \alpha \models \neg K_v \neg p$ .*

## 5. Architectural refinement

During the process of system development, architectural designs are often refined by decomposing components into sub-components. We investigate the preservation of information security properties with respect to architectural refinement, including refinement of extended architectures.

Van der Meyden [2009] defines architectural refinement for information-flow architectures as follows: architecture  $\mathcal{A}_1 = (D_1, \rightarrow_1)$  is a refinement of architecture  $\mathcal{A}_2 = (D_2, \rightarrow_2)$  if there is a function  $r : D_1 \rightarrow D_2$  such that  $r$  is onto  $D_2$  and for all  $u, v \in D_1$ , if  $u \rightarrow_1 v$  then  $r(u) \rightarrow_2 r(v)$ . We write  $\mathcal{A}_1 \leq_r \mathcal{A}_2$  if  $\mathcal{A}_1$  refines  $\mathcal{A}_2$  via refinement function  $r$ .

Intuitively, if  $\mathcal{A}_1$  refines  $\mathcal{A}_2$ , then  $\mathcal{A}_1$  provides more detail than  $\mathcal{A}_2$ , as it is a finer-grain specification of security domains and the information flows between them. Refinement function  $r$  indicates how domains in  $D_2$  are decomposed into subdomains in  $D_1$ : for all  $u \in D_1$ ,  $u$  is a subdomain of  $r(u) \in D_2$ . The refinement function  $r$  ensures that information flows between subdomains in  $\mathcal{A}_1$  are in accordance with information flows between domains in  $\mathcal{A}_2$ .

**Example:** The Hinke-Schaefer architecture  $\mathcal{H}\mathcal{S}$  refines architecture  $\mathcal{H}\mathcal{L}$  via refinement function  $r$  that maps  $H_{user}$ ,  $H_{DBMS}$ , and  $H_F$  to  $H$ , and maps  $L_{user}$ ,  $L_{DBMS}$ , and  $L_F$  to  $L$ . All information flows within  $\mathcal{H}\mathcal{S}$  are permitted by  $\mathcal{H}\mathcal{L}$ , including the flow from  $L_F$  to  $H_{DBMS}$ , which is permitted as  $L \rightarrow H$ .  $\square$

It is shown in van der Meyden [2009] that TA-compliance is preserved by refinement. That is, if machine  $M = \langle S, s_0, A, D_1, \text{step}, \text{obs}, \text{dom} \rangle$  is TA-compliant with architecture  $\mathcal{A}_1$ , and  $\mathcal{A}_1 \leq_r \mathcal{A}_2$ , then  $r(M)$  is TA-compliant with  $\mathcal{A}_2$ , where  $r(M) = \langle S, s_0, A, D_2, \text{step}, \text{obs}', \text{dom}' \rangle$  is identical to  $M$  but has domains  $D_2 = r(D_1)$ , and the functions  $\text{dom}'$  and  $\text{obs}'$  are defined using

domains from  $\mathcal{A}_2$ . In particular,  $\text{dom}' = r \circ \text{dom}$ , and  $\text{obs}'_u(s) = \text{obs}_{\{v \in D_1 \mid r(v)=u\}}(s)$ , where  $\text{obs}_G(s) = (\text{obs}_{u_1}(s), \dots, \text{obs}_{u_n}(s))$  for  $G = \{u_1, \dots, u_n\}$ .

We generalize architectural refinement to interpreted extended architectures. We first note that there is a different notion of refinement that can be defined at the semantic level. Let  $\mathcal{AI}_1 = (\mathcal{A}_1, \mathcal{I}_1)$  and  $\mathcal{AI}_2 = (\mathcal{A}_2, \mathcal{I}_2)$  be interpreted extended architectures, where  $\mathcal{A}_i = (D_i, \succ_i)$ ,  $\mathcal{I}_i = (A, \text{dom}_i, \mathbf{I}_i)$  for  $i = 1, 2$ . (We assume that the interpretations have the same set of actions  $A$ —we do not attempt to deal with action refinement.)

**DEFINITION 4.** We say that a mapping  $r$  is a semantic refinement mapping from  $\mathcal{AI}_1$  to  $\mathcal{AI}_2$ , and write  $\mathcal{AI}_1 \preceq_r \mathcal{AI}_2$ , if  $r$  is onto  $D_2$ ,  $\text{dom}_2 = r \circ \text{dom}_1$  and for all  $u \in D_1$ , and sequences  $\alpha, \beta \in A^*$ , if  $\text{fta}_{r(u)}^{\mathcal{AI}_2}(\alpha) = \text{fta}_{r(u)}^{\mathcal{AI}_1}(\beta)$ , then  $\text{fta}_u^{\mathcal{AI}_1}(\alpha) = \text{fta}_u^{\mathcal{AI}_2}(\beta)$ .

This definition directly implies preservation of FTA-compliance:

**THEOREM 9.** If  $\mathcal{AI}_1 \preceq_r \mathcal{AI}_2$ , then for all machines  $M$ , if  $M$  is FTA-compliant with  $\mathcal{AI}_1$ , then  $r(M)$  is FTA-compliant with  $\mathcal{AI}_2$ .

While semantic refinement therefore has the key property that we want from a notion of architectural refinement, establishing semantic refinement requires reasoning about the functions  $\text{fta}_u$ , which have a complicated inductive definition that, for a given domain  $u$ , potentially ranges over the entire set of domains.

We now develop a notion of refinement that is stated in a more local way. Intuitively, if  $\mathcal{AI}_1$  and  $\mathcal{AI}_2$  are interpreted extended architectures, the former refining the latter, then if  $\mathcal{AI}_1$  allows information flow from domain  $u$  to  $v$ , filtered by function  $h_1$ , then  $\mathcal{AI}_2$  must allow information flow from domain  $r(u)$  to domain  $r(v)$  filtered by a function  $h_2$  that permits at least as much information flow as  $h_1$ . We give the formal definition just for the case of fully filtered interpreted architectures. By Theorem 4, the definition can be extended to architectures that are not fully filtered. We later give some sufficient conditions that simplify the refinement conditions for architectures using  $\top$ .

Let  $\mathcal{AI}_1 = (\mathcal{A}_1, \mathcal{I}_1)$  and  $\mathcal{AI}_2 = (\mathcal{A}_2, \mathcal{I}_2)$  be fully filtered interpreted extended architectures, where  $\mathcal{A}_i = (D_i, \succ_i)$ ,  $\mathcal{I}_i = (A, \text{dom}_i, \mathbf{I}_i)$  for  $i = 1, 2$ . (As above, we require that the interpretations have the same set of actions  $A$ .) Formally, define a refinement mapping from  $\mathcal{AI}_1$  to  $\mathcal{AI}_2$  to be a function  $r : D_1 \rightarrow D_2$  such that the following conditions hold.

- R1 The function  $r$  is onto  $D_2$ , and  $\text{dom}_2 = r \circ \text{dom}_1$ .
- R2 For all  $u, v \in D_1$ , if  $u \xrightarrow{f_1} v$  then there exists an edge  $r(u) \xrightarrow{f_2} r(v)$ , such that for  $\alpha \in A^*$  and  $a \in A$  with  $\text{dom}_1(a) = u$ , if  $\mathbf{I}_2(f_2)(\alpha a) = \epsilon$ , then  $\mathbf{I}_1(f_1)(\alpha a) = \epsilon$ .
- R3 For  $v \in D_1$  and  $\alpha, \beta \in A^*$  and  $a, b \in A$ , if  $\text{dom}_2(a) \xrightarrow{f_2} r(v)$  and  $\text{dom}_2(b) \xrightarrow{g_2} r(v)$  and  $\mathbf{I}_2(f_2)(\alpha a) = \mathbf{I}_2(g_2)(\beta b) \neq \epsilon$ , then
  - (a) if  $\text{dom}_1(a) \xrightarrow{f_1} v$  and  $\text{dom}_1(b) \xrightarrow{g_1} v$  then  $\mathbf{I}_1(f_1)(\alpha a) = \mathbf{I}_1(g_1)(\beta b)$ , and
  - (b) if  $\text{dom}_1(a) \xrightarrow{f_1} v$  and  $\text{dom}_1(b) \not\xrightarrow{g_1} v$  then  $\mathbf{I}_1(f_1)(\alpha a) = \epsilon$ .

Intuitively, this generalizes the previous definition of refinement (for non-extended architectures) to information flow policies with edges labeled by filter functions, and requires that at the concrete level (i.e., in  $\mathcal{AI}_1$ ), no more information is permitted to flow along an edge  $u \xrightarrow{f_1} v$  than is permitted to flow along the corresponding edge  $r(u) \xrightarrow{f_2} r(v)$  at the abstract level (i.e., in  $\mathcal{AI}_2$ ). In particular, condition R2 ensures that if  $f_2$  does not allow any information to flow ( $\mathbf{I}_2(f_2)(\alpha a) = \epsilon$ ), then neither does  $f_1$ . Condition R3 ensures that if domain  $r(v) \in D_2$  cannot distinguish two potential inputs  $\mathbf{I}_2(f_2)(\alpha a)$  and  $\mathbf{I}_2(g_2)(\beta b)$ , then neither can domain  $v \in D_1$

distinguish the corresponding inputs it could receive according to the concrete policy. Note that if we take  $\text{dom}_1(a) = \text{dom}_1(b)$  (which implies  $f_1 = g_1$  and  $f_2 = g_2$ ) then we obtain the following more intuitive condition:

- R4 For  $\alpha, \beta \in A^*$  and  $a, b \in A$  such that  $\text{dom}_1(a) = \text{dom}_1(b)$ , if  $\text{dom}_1(a) \xrightarrow{f_1} v$  and  $\text{dom}_2(a) \xrightarrow{f_2} r(v)$ , and  $\mathbf{I}_2(f_2)(\alpha a) = \mathbf{I}_2(f_2)(\beta b)$ , then  $\mathbf{I}_1(f_1)(\alpha a) = \mathbf{I}_1(f_1)(\beta b)$ .

This states more transparently that at least as much information is permitted to flow along abstract edges as is permitted to flow along corresponding concrete edges.

We overload notation, and write  $\mathcal{AI}_1 \leq_r \mathcal{AI}_2$  if interpreted extended architecture  $\mathcal{AI}_1$  refines interpreted extended architecture  $\mathcal{AI}_2$  via refinement function  $r$ . Similarly, we write  $(\mathcal{A}_1, \mathcal{C}_1) \leq_r (\mathcal{A}_2, \mathcal{C}_2)$  if for every architectural interpretation  $\mathcal{I}_1 \in \mathcal{C}_1$ , there is an architectural interpretation  $\mathcal{I}_2 \in \mathcal{C}_2$  such that  $(\mathcal{A}_1, \mathcal{I}_1) \leq_r (\mathcal{A}_2, \mathcal{I}_2)$ .

## 5.1 Preservation of security

FTA-compliance is also preserved by refinement, as the following theorem demonstrates.

**THEOREM 10.** Let  $\mathcal{AI}_1$  and  $\mathcal{AI}_2$  be fully filtered interpreted extended architectures such that  $\mathcal{AI}_1 \leq_r \mathcal{AI}_2$ . Then  $\mathcal{AI}_1 \preceq_r \mathcal{AI}_2$ . Hence, if  $M$  is FTA-compliant with  $\mathcal{AI}_1$  then  $r(M)$  is FTA-compliant with  $\mathcal{AI}_2$ .

Certain information security properties are also preserved by refinement. Suppose  $\mathcal{A}_1 \leq_r \mathcal{A}_2$ , and there is a proposition  $p$  that  $u \in D_2$  does not know. Then for any domain  $v \in D_1$  such that  $r(v) = u$ ,  $v$  does not know  $p$  either. Indeed for any group of domains  $G \subseteq D_1$  such that for all  $v \in G$ ,  $r(v) = u$ , group  $G$  does not have distributed knowledge of  $p$ .

**THEOREM 11.** Let  $(\mathcal{A}_1, \mathcal{I}_1)$  and  $(\mathcal{A}_2, \mathcal{I}_2)$  be interpreted extended architectures such that  $(\mathcal{A}_1, \mathcal{I}_1) \leq_r (\mathcal{A}_2, \mathcal{I}_2)$ , where  $\mathcal{A}_i = (D_i, \succ_i)$  for  $i = 1, 2$ . If  $M$  is FTA-compliant with  $(\mathcal{A}_1, \mathcal{I}_1)$  then for all  $u \in D_2$ , and all  $G \subseteq D_1$  such that  $r(v) = u$  for all  $v \in G$ , for all sequences  $\alpha \in A^*$ , and all propositional constants  $p$ , if  $r(M), \pi, \alpha \models \neg K_u p$  then  $M, \pi, \alpha \models \neg D_G p$ .

We can also reason about how refinement affects  $u$ -action-local propositions.

**LEMMA 1.** Let  $(\mathcal{A}_1, \mathcal{I}_1)$  and  $(\mathcal{A}_2, \mathcal{I}_2)$  be interpreted extended architectures such that  $(\mathcal{A}_1, \mathcal{I}_1) \leq_r (\mathcal{A}_2, \mathcal{I}_2)$ . If  $X$  is a  $u$ -action-local proposition then  $X$  is a  $r(u)$ -action-local proposition.

## 5.2 Refinement using $\top$

We have defined refinement above assuming fully filtered interpreted architectures. To extend the definition to interpreted architectures that include edges labeled  $\top$ , we may apply the transformation to an equivalent fully filtered architecture, and define  $\mathcal{AI}_1 \leq_r \mathcal{AI}_2$  to hold if  $\mathcal{AI}_1^\dagger \leq_r \mathcal{AI}_2^\dagger$ . However, to prove the latter refinements requires that we reason about relationships between the complicated functions  $\text{fta}_u$  used in interpretations of edges of  $\mathcal{AI}_i^\dagger$ . This is undesirable, since the aim of refinement is essentially to obtain a conclusion about those same functions! We show in this section that it is possible in many cases to eliminate this (non-vicious) circularity and to simplify the conditions that need to be checked to prove refinement of architectures involving  $\top$ .

Let  $\mathcal{AI}_1 = (\mathcal{A}_1, \mathcal{I}_1)$  and  $\mathcal{AI}_2 = (\mathcal{A}_2, \mathcal{I}_2)$  be interpreted architectures, where  $\mathcal{A}_i = (D_i, \succ_i)$ , and  $\mathcal{I}_i = (A, \text{dom}_i, \mathbf{I}_i)$  for  $i = 1, 2$ . Say that a function  $r : D_1 \rightarrow D_2$  is a *strict refinement function* from  $\mathcal{AI}_1$  to  $\mathcal{AI}_2$  if it satisfies the following conditions:

- SR1 The function  $r$  is onto  $D_2$ , and  $\text{dom}_2 = r \circ \text{dom}_1$ .

SR2 If  $u \xrightarrow{\top}_1 v$  then  $r(u) \xrightarrow{\top}_2 r(v)$ .

SR3 For every edge  $u \xrightarrow{f_1}_1 v$  with  $f_1 \neq \top$ , there is an edge  $r(u) \xrightarrow{f_2}_2 r(v)$  such that if  $f_2 \neq \top$  then

- (a) for all  $\alpha \in A^*$  and  $a \in A$  with  $\text{dom}_1(a) = u$ , if  $\text{I}_2(f_2)(\alpha a) = \epsilon$  then  $\text{I}_1(f_1)(\alpha a) = \epsilon$ .
- (b) for all  $\alpha, \beta \in A^*$  and  $a, b \in A$  with  $\text{dom}_1(a) = \text{dom}_1(b) = u$ , if  $\text{I}_2(f_2)(\alpha a) = \text{I}_2(f_2)(\beta b)$  then  $\text{I}_1(f_1)(\alpha a) = \text{I}_1(f_1)(\beta b)$ .

SR4 Messages in  $\mathcal{I}_2$  are source-identifying with respect to  $r$  for edges not labeled  $\top$ , i.e., there exists a function  $\mathcal{S}$  mapping the union of the ranges of the filter functions  $\text{I}_2(f)$  of  $\mathcal{AI}_2$  where  $f \neq \top$  to the set of domains  $D_1$  of  $\mathcal{AI}_1$ , such that for all  $\alpha \in A^*$  and  $a \in A$ , if  $r(\text{dom}_1(a)) \xrightarrow{f_2}_2 w$  and  $\text{I}_2(f_2)(\alpha a) \neq \epsilon$ , then  $\mathcal{S}(\text{I}_2(f_2)(\alpha a)) = \text{dom}_1(a)$ .

The following result shows that strict refinement is a sufficient condition for refinement.

**THEOREM 12.** *If  $r$  is a strict refinement function from  $\mathcal{AI}_1$  to  $\mathcal{AI}_2$  then  $\mathcal{AI}_1 \preceq_r \mathcal{AI}_2$ .*

This result can significantly simplify the verification of refinement. However, it provides sufficient but not necessary conditions for refinement. For example, it is possible to have refinements where we have an edge  $u \xrightarrow{\top}_1 v$  corresponding to an abstract edge  $r(u) \xrightarrow{f}_2 r(v)$ , where  $f \neq \top$ . This cannot be established using a strict refinement function since condition SR2 is not satisfied.

### 5.3 Example: Hinke-Schaefer

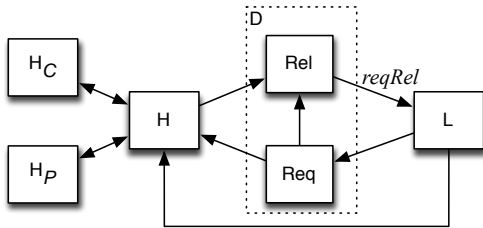
Since the Hinke-Schaefer architecture  $\mathcal{HS}$  refines architecture  $\mathcal{HL}$ , we can apply the information security result for  $\mathcal{HL}$ , Theorem 1, to  $\mathcal{HS}$ : since domain  $L$  never knows any  $H$ -action-local proposition, the domains  $L_{user}$ ,  $L_{DBMS}$ , and  $L_F$  never know any  $u$ -action-local proposition, for  $u \in \{H_{user}, H_{DBMS}, H_F\}$ . This information security property was stated as Theorem 2, in Section 2.2. We give a simple proof for it here.

**Proof of Theorem 2:** Follows easily from Theorem 1, Lemma 1, and Theorem 11, since  $\mathcal{HS} \preceq_r \mathcal{HL}$  for refinement function  $r$  such that  $r(L_{user}) = r(L_{DBMS}) = r(L_F) = L$  and  $r(H_{user}) = r(H_{DBMS}) = r(H_F) = H$ .  $\square$

Thus, we were able to prove an information security property about  $\mathcal{HS}$  by proving an appropriate policy in the much simpler architecture  $\mathcal{HL}$ .

### 5.4 Example: Downgrader

The following diagram shows architecture  $\mathcal{DGR}$ , a refinement of architecture  $\mathcal{DG}$  of Section 4.3.



In  $\mathcal{DGR}$ , the downgrader  $D$  is decomposed into subdomains,  $Req$  and  $Rel$ , and more restrictions are placed on the release of information. Domain  $Req$  receives and approves requests from domain  $L$  for information release. Domain  $Rel$  should release information to domain  $L$  only if there is an approved release request.

The edge labeled  $reqRel$  restricts what information is released to  $L$ . We restrict architecture interpretations so that  $\text{I}(reqRel)$  only releases information about actions from domains  $H$ ,  $H_P$ , and  $D$  that occurred before the last  $Req$  action. Thus, if there is no  $Req$  action (i.e., no request has been approved), then no information is released.

Let  $\xrightarrow{\cdot}_{\mathcal{DGR}}$  denote the information flow policy of  $\mathcal{DGR}$ . We define specification  $\mathcal{C}_{\mathcal{DGR}}$  so that  $(A, \text{dom}, \text{I}) \in \mathcal{C}_{\mathcal{DGR}}$  if and only if  $(A, \text{dom}, \text{I})$  is an interpretation of  $\mathcal{DGR}$  such that  $\text{I}(reqRel)$  is defined recursively as follows:

$$\begin{aligned} \text{I}(reqRel)(\epsilon) &= \epsilon \\ \text{I}(reqRel)(\alpha a) &= \begin{cases} \text{I}(reqRel)(\alpha) & \text{if } \text{dom}(a) \neq Req \\ \text{fta}_{Rel}^{\xrightarrow{\cdot}_{\mathcal{DGR}}}(\alpha) & \text{if } \text{dom}(a) = Req. \end{cases} \end{aligned}$$

Here  $\xrightarrow{\cdot}_{\mathcal{DGR}}$  is the policy obtained by restricting  $\xrightarrow{\cdot}_{\mathcal{DGR}}$  to domains  $H_P$ ,  $H$ ,  $Req$ , and  $Rel$ . Thus  $(u, v, f) \in \xrightarrow{\cdot}_{\mathcal{DGR}}$  if and only if either  $(u, v, f) \in \xrightarrow{\cdot}_{\mathcal{DGR}}$  and  $\{u, v\} \subseteq \{H_P, H, Req, Rel\}$  or  $u = v$ . Note the similarity to policy  $\xrightarrow{\cdot}'$  that was used to restrict interpretations of filter  $rel$  of architecture  $\mathcal{DG}$  in Section 4.3 above.

**THEOREM 13.**  $\mathcal{C}_{\mathcal{DGR}}$  refines  $\mathcal{C}_{\mathcal{DG}}$ .

Any machine  $M$  that is compliant with architectural specification  $(\mathcal{DG}, \mathcal{C}_{\mathcal{DG}})$  does not reveal any non-trivial  $H_C$ -action-local proposition to  $L$ . Since  $\mathcal{C}_{\mathcal{DGR}}$  refines  $\mathcal{C}_{\mathcal{DG}}$ , the same property holds for any machine  $M$  that is compliant with  $(\mathcal{DGR}, \mathcal{C}_{\mathcal{DGR}})$ . We thus prove an information security property about architecture  $\mathcal{DGR}$  by reference to the more abstract architecture  $\mathcal{DG}$ .

**THEOREM 14.** *If  $M$  is FTA-compliant with architectural specification  $(\mathcal{DGR}, \mathcal{C}_{\mathcal{DGR}})$  and  $\pi(p)$  is a non-trivial  $H_C$ -action-local proposition then  $M, \pi \models \neg K_L p$ .*

## 6. Architecture to implementation

This paper focuses on showing that strong information security properties of a system can be derived from an architectural specification of a system. Architectures are high-level descriptions of a system's structure. Architectural refinement allows more details of a system to be specified, but an architecture is not an implementation or even a low-level design.

In order to conclude that a concrete system satisfies the security properties we have considered in our examples, it is necessary to demonstrate that the implementation complies with the architecture, and that all (trusted) components satisfy their local constraints. In an extended version of this paper we show that this is the case for a number of concrete implementations, but how to do this in general is a topic requiring significant further research. Here we just briefly discuss the range of techniques that might be involved in such a demonstration.

Boettcher et al. [2008] survey techniques to achieve *separation* of components, that is, to ensure that communication between components is in accordance with the architecture. In the case of unextended architectures, a very common technique to ensure compliance with the architecture is to map information flow edges to physical causality and use physical separation where there is no edge. Thus, the architecture  $\mathcal{HL}$  is commonly enforced in military settings by mapping  $H$  and  $L$  to distinct processors and/or networks and using trusted devices (data diodes) to ensure a one-way information flow from  $L$  to  $H$ . The Starlight Interactive Link [Anderson et al., 1996] is a trusted device that can be added to such an implementation to extend it to an implementation of the architecture  $\mathcal{SL}$ .

One of the longstanding objectives of research on military-grade security has been how to avoid the redundancy and consequent expense of such physical implementations, through the use

of implementations that enable different security levels to share resources such as memory, processors and networks. An implementation technique that forms the basis for much work in MILS security is the use of separation kernels, which are highly simplified operating systems with the sole functionality of enforcing an information flow policy. Use of such a system introduces the risk that there are covert channels, but much progress has been made in recent years towards formal proofs that separation kernels enforce an information flow policy (e.g., Greve et al. [2003]; Heitmeyer et al. [2006]). The key mechanism used to achieve this is typically enforcement of an access control policy: results of Rushby [1992], improved by van der Meyden [2007], show that there is a very close relationship between information flow and access control. (A concrete example of how access control can be used to enforce an architecture is given in van der Meyden [2009].)

Once a basic (unextended) architecture has been shown to be enforced by the implementation, it remains to demonstrate that the trusted components in the architecture satisfy their local constraints. The extended architectures introduced in Section 4 express such constraints using filter functions, and we used architectural specifications to semantically represent restrictions on the trusted components and these filter functions. As the additional local constraints are application specific, and implementation dependent, it seems unlikely that a single methodology will suffice. We expect that theorem proving, model checking and language-based information flow techniques [Sabelfeld and Myers, 2003] may all be used to provide assurance of satisfaction of the local constraints and the filtering requirements introduced by our extended architectures. We note that our framework is highly expressive, and it may not always be possible to show compliance with a filtering requirement locally. For example, enforcing the filter functions in our downgrader architectures may require cooperation of High level components (e.g., provision of secure provenance information) to ensure that the downgrader does not release information concerning  $H_C$ . Identifying sufficient conditions for local verification of compliance is an interesting topic for future research.

## 7. Related Work

The most closely related work is that of van der Meyden [2007, 2009], who defines TA-security and considers refinement of architectures on the basis of this semantics. Our contribution is to show that the definition of TA-security supports derivation of global information security properties, to extend TA-security to architectures that include filter functions, and to develop an account of architectural refinement for these extended architectures. The extension provides a way to specify the behavior of trusted components in a system: intuitively, if a component is the source of an edge in the architecture labeled by a filter function, then the component is trusted to ensure that the interpretation of the filter function limits the information that may flow along the edge. We have presented a number of examples that show that interesting information-theoretic global security properties can be derived from the very abstract statement that a system complies with such an extended architecture and a set of additional local properties. We note that these global properties are more general and application specific than the very particular property “Low does not know any High secrets” that is most often considered in the literature.

Other work has sought to formally describe system architecture (e.g., AADL and Acme [Garlan et al., 2000]), and reason about the properties of systems conforming to a given architecture. There are many software engineering concerns that can be reasoned about in architectural design, such as maintainability, and reliability. This work focuses on reasoning about the information security of systems, and, as such, our architectures specify local constraints on information that may be communicated between components. The

local constraints allow the inference of global information security properties. This work is complementary to work on other aspects of system design.

Relatively little theoretical work takes an architectural perspective on information security. One interesting line of work [Hansson et al., 2008] that takes a similar perspective to ours is conducted in the context the architectural modeling framework AADL. However, this is based on the Bell La Padula model [Bell and La Padula, 1976] rather than the more abstract noninterference-based approach taken in our work.

Standard notions of refinement reduce the possible behaviors of a system. However, arbitrary behavioral refinements of a system will, in general, not preserve information security properties [Jacob, 1989]. Previous work has investigated restricting behavioral refinements to preserve information security properties [Graham-Cunning and Sanders, 1991; O’Halloran, 1992; Roscoe, 1995; Bossi et al., 2003; Morgan, 2006]. Architectural refinement as used in this paper allows only refinements that reduce the information communicated between system components. As shown in Section 5.1, this notion of refinement preserves certain information security properties. Information security can also be preserved under refinement by modifying the refined system [Jacob, 1989; Mantel, 2001]. Other work distinguishes nondeterminism of a system specification from nondeterminism inherent in the system, and allows refinement only of specification nondeterminism [Seehusen and Stolen, 2006; Jürjens, 2005; Bibighaus, 2006]. These works typically consider only the simple policy  $L \mapsto H$ , rather than the more general intransitive policies considered here.

Most work on architectural refinement is not directly concerned with information security. Zhou and Alves-Foss [2006] propose architecture refinement patterns for Multi-Level Secure systems development, but do not provide formal semantics. A series of papers [Moriconi and Qian, 1994; Moriconi et al., 1995, 1997] express architectural designs as logical theories and refinement as a mapping from an abstract theory to a concrete theory. This approach is used to establish security properties in variants of the X/Open Distributed Transaction Processing architecture, using the Bell La Padula model [Bell and La Padula, 1976], which lacks the kind of information flow semantics that we have studied here. It is not clear if this approach is sufficiently expressive to represent architectural refinement as used in this paper, and reason about the preservation of information security under such refinement.

Preservation of information flow properties under system composition has, like refinement, been considered problematic. In general, the composition of two secure systems is not guaranteed to be secure. The reason is essentially the same as for refinement: composition reduces the set of possible behaviors of a system, enabling an observer to make additional deductions.

However, a number of approaches have been developed that allow security of a composed system to be derived from security of its components. These include use of a stronger definitions of security such as restrictiveness [McCullough, 1990] or bisimulation-based nondeducibility on compositions [Focardi and Gorrieri, 1994]. McLean [1996] proposes a framework for specifying and reasoning about system composition, and the preservation of possibilistic security properties. In this work, we are not concerned with showing that security properties that hold of components also hold of a composite system. Instead we are concerned with proving global security properties, and identifying local constraints that components must satisfy. The literature on preservation of information flow security under composition has also largely limited itself to the simple policy  $L \mapsto H$ .

To some extent, process algebraic operations can be viewed as expressing architectural structure. For example, one could take the view that a process constructed as the parallel composition

of two processes  $P$  and  $Q$ , with actions  $A$  of the composition then hidden, corresponds to an architecture in which  $P$  and  $Q$  are permitted to interfere, but which the environment is not permitted to influence through the set of actions  $A$ . However, the semantics of these operators usually do not preserve this view: typically this composition is understood in terms of its possible behaviors with respect to the actions that have not been hidden, and the fact that the system has been composed out of two components permitted to interfere in a particular way is lost in the meaning of the composition.

Downgrading has historically been one of the motivations for generalizing the notion of noninterference to intransitive policies. Roscoe and Goldsmith [1999] argued against the ipurge-based semantics for noninterference on the grounds that the meaning it gives to the downgrader policy  $H \rightsquigarrow D \rightsquigarrow L$  is too permissive. According to this semantics, any action by the downgrader  $D$  “opens the floodgates,” in the sense that it allows all information about the High security domain  $H$  to flow to the low security domain  $L$ . Roscoe and Goldsmith proposed to deal with this issue by making the semantics of intransitive noninterference significantly more restrictive, in effect reverting to the purge-based definition of Goguen and Meseguer [1982, 1984]. Our approach to downgrading, using a filter function, provides an alternative approach that enables explicit specification of the information permitted to be released by the downgrader.

More recent work on downgrading has concentrated on downgrading in the setting of language-based security. Sabelfeld and Sands [2005] briefly survey recent work on downgrading in language-based settings, and propose several dimensions of downgrading, and prudent principles for downgrading. They regard intransitive noninterference as specifying *where* (in the security levels) downgrading may occur. Since we interpret security levels as system components, our architectures specify where in the system downgrading may occur. The filter functions that we propose in this work specify *what* information can be downgraded and *when* this may occur. Thus, our work combines the *what*, *where*, and *when* dimensions of downgrading. Recent work also considers multiple dimensions of downgrading, including Barthe et al. [2008], Banerjee et al. [2008], Mantel and Reinhard [2007] and Askarov and Sabelfeld [2007b].

Recent work [Askarov and Sabelfeld, 2007a; Banerjee et al., 2008] considers “knowledge-based” approaches to downgrading in language-based settings. However, they do not reason about security properties as general and application specific as used in this paper. O’Neill [2006] uses epistemic logic to specify many information security properties, but does not directly consider downgrading.

## 8. Conclusion

Through the examination of a number of examples, we have shown that strong information security properties can be proven about a system from a high-level architectural description of the system. Any system that complies with the architecture will satisfy the information security properties that can be proven about the architecture.

We extended the notion of architecture to allow finer-grain specification of what information may be sent between components. This enables the proof of stronger security properties, while continuing to providing the benefits of using a high-level architectural description. We generalized the notion of architectural refinement [van der Meyden, 2009] for the extended architectures. Certain security properties are preserved by architectural refinement.

The MILS vision is to build high-assurance systems with well-understood security properties by composition of COTS infrastructure and trusted components. This work brings us closer to that

goal, by demonstrating that it is possible to compositionally derive strong information-flow security properties from high-level system specifications.

## References

- AADL. Architecture analysis and design language (AADL). SAE Standard AS5506/A, Jan. 2009.
- J. Alves-Foss, W. Harrison, P. Oman, and C. Taylor. The MILS architecture for high-assurance embedded systems. *International Journal of Embedded Systems*, 2(3/4):239–47, Feb. 2006.
- M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg, and K. Yiu. Starlight: Interactive link. *Annual Computer Security Applications Conference*, pages 55–63, 1996.
- A. Askarov and A. Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 207–221. IEEE Computer Society, 2007a.
- A. Askarov and A. Sabelfeld. Localized delimited release: combining the what and where dimensions of information release. In *Proceedings of the 2007 Workshop on Programming Languages and Analysis for Security*, pages 53–60. ACM Press, 2007b.
- A. Banerjee, D. A. Naumann, and S. Rosenberg. Expressive declassification policies and modular static enforcement. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2008.
- G. Barthe, S. Cavadini, and T. Rezk. Tractable enforcement of declassification policies. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*. IEEE Computer Society, June 2008.
- D. Bell and L. La Padula. Secure computer system: unified exposition and multics interpretation. Technical Report ESD-TR-75-306, Mitre Corporation, Bedford, M.A., March 1976.
- D. Bibighaus. *Applying the doubly labeled transition system to the refinement paradox*. PhD thesis, Naval Postgraduate School, Monterey, 2006.
- C. Boettcher, R. DeLong, J. Rushby, and W. Sifre. The MILS component integration approach to secure information sharing. In *Proceedings of the 27th IEEE/AIAA Digital Avionics Systems Conference*, pages 1.C.2–1–1.C.2–14, Oct. 2008.
- A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Refinement operators and information flow security. In *Proceedings of the International Conference on Software Engineering and Formal Methods*, pages 44–53, 2003.
- R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995.
- R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1994.
- D. Garlan, R. T. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In G. T. Leavens and M. Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.
- J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, Apr. 1982.
- J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 75–86. IEEE Computer Society, Apr. 1984.

- J. Graham-Cunning and J. Sanders. On the refinement of noninterference. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 35–42, 1991.
- D. Greve, M. Wilding, and W. M. Vanfleet. A separation kernel formal security policy. In *Proceedings of the Fourth International Workshop on the ACL2 Prover and Its Applications*, July 2003.
- J. T. Haigh and W. D. Young. Extending the noninterference version of MLS for SAT. *IEEE Transactions on Software Engineering*, 13(2):141–150, 1987.
- J. Hansson, P. H. Feiler, and J. Morley. Building secure systems using model-based engineering and architectural models. *CrossTalk: The Journal of Defense Software Engineering*, 21(9), Sept. 2008.
- C. L. Heitmeyer, M. Archer, E. I. Leonard, and J. McLean. Formal specification and verification of data separation in a separation kernel for an embedded system. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 346–355, New York, NY, USA, 2006. ACM.
- T. H. Hinke and M. Schaefer. Secure data management system. Technical Report RADC-TR-75-266, System Development Corporation, Nov. 1975.
- J. Jacob. On the derivation of secure components. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 242–247, 1989.
- J. Jürjens. *Secure Systems Development with UML*. Springer, 2005.
- H. Mantel. Preserving information flow properties under refinement. In *Proceedings of the IEEE Symposium Security and Privacy*, pages 78–91, 2001.
- H. Mantel and A. Reinhard. Controlling the what and where of declassification in language-based security. In R. D. Nicola, editor, *Proceedings of the 16th European Symposium on Programming*, volume 4421 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2007.
- D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.
- J. McLean. A general theory of composition for a class of “possibilistic” properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.
- C. Morgan. *The Shadow Knows*: refinement of ignorance in sequential programs. In *Mathematics of Program Construction*, pages 359–378, 2006.
- M. Moriconi and X. Qian. Correctness and composition of software architectures. In *Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 164–174, 1994.
- M. Moriconi, X. Qian, and R. Riemenschneider. Correct architecture refinement. *IEEE Transactions on Software Engineering*, 21(4):356–372, April 1995.
- M. Moriconi, X. Qian, R. A. Riemenschneider, and L. Gong. Secure software architectures. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 84–893, 1997.
- C. O’Halloran. Refinement and confidentiality. In *Proceedings of the Fifth Refinement Workshop*, pages 119–139. British Computer Society, 1992.
- K. R. O’Neill. *Security and Anonymity in Interactive Systems*. PhD thesis, Cornell University, Aug. 2006.
- A. Roscoe. CSP and determinism in security modelling. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 114–221, 1995.
- A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 1999.
- J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report CSL-92-02, SRI International, Dec 1992.
- A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, Jan. 2003.
- A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop*, pages 255–269. IEEE Computer Society, June 2005.
- F. Seehusen and K. Stolen. Information flow property preserving transformation of UML interaction diagrams. In *Proceedings of the ACM Symposium on Access Control Models and Technologies*, pages 150–159, 2006.
- B. M. Thuraisingham. *Database and Applications Security: Integrating Information Security and Data Management*. CRC Press, 2005.
- R. van der Meyden. Architectural refinement and notions of intransitive noninterference. In *Proceedings of the 1st International Symposium on Engineering Secure Software and Systems*, volume 5429 of *Lecture Notes in Computer Science*, pages 60–74. Springer, Feb. 2009.
- R. van der Meyden. What, indeed, is intransitive noninterference? In *Proceedings of the 12th European Symposium On Research In Computer Security*, volume 4734 of *Lecture Notes in Computer Science*, pages 235–250. Springer, Sept. 2007.
- W. Vanfleet, R. Beckworth, B. Calloni, J. Luke, C. Taylor, and G. Uchenick. MILS:architecture for high assurance embedded computing. *Crosstalk: The Journal of Defence Engineering*, pages 12–16, Aug. 2005.
- J. Zhou and J. Alves-Foss. Architecture-based refinements for secure computer system design. In *Proceedings of the Policy, Security and Trust*, Nov 2006.