Scientific Annals of Computer Science vol. 21, 2011, pp. 107–149

Designing, Capturing and Validating History-Sensitive Security Policies for Distributed Systems

Alejandro Mario HERNANDEZ[1], Flemming NIELSON[1], Hanne RIIS-NIELSON[1]

Abstract

We consider the use of Aspect-oriented techniques as a flexible way to deal with security policies in distributed systems. We follow the approach of attaching security policies to the relevant locations that must be governed by them, and then combining them at runtime according to the interactions that happen. Recent work suggests using Aspects in this way to analyse the future behaviour of programs and to make access control decisions based on this; this gives the flavour of dealing with information flow rather than mere access control. We show in this paper that it is beneficial to augment this approach with history-based components, as is traditional in reference-monitor-based approaches to mandatory access control. Our developments are performed in an Aspect-oriented coordination language, aiming to describe the Bell-LaPadula policy as elegantly as possible. Furthermore, the resulting language has the capability of combining both history-sensitive and future-sensitive policies, providing even more flexibility and power. Moreover, we propose a global Logic for reasoning about the systems designed with this language. We show how the Logic can be used to validate the combination of security policies in a distributed system, either with or without exploring the entire state space.

[1] Department of Informatics, Technical University of Denmark, Richard Petersens Plads, Building 322, 2800 Kgs. Lyngby, Denmark.
Email: {aher, nielson, riis}@imm.dtu.dk

1 Introduction

Distributed Systems are supposed to manage large amounts of information, so they must be secured [25, 33] to provide confidentiality for the information managed by them. The well-consolidated Aspect-Orientation [23] field has been targeted to some security approaches, and in this paper we aim to exploit these ideas even more.

Aspect-orientation provides flexibility in a system, allowing the inclusion of new features without extensive modification in the existing components. Security is in general an evolving concern during the lifetime of a system. Security policies [17], in particular, can be seen as individual security goals given by specific security designers. New designers may involve new policies, or adaptation of the existing ones. Even simple overall-goal changing may involve policy modification. Aspect-orientation seems to be appropriate for this kind of system extension.

Traditional approaches to enforce security policies at runtime follow the reference monitors [24] concept. The typical way is to assess security compliance for each runtime operation, forbidding those that do not comply with the security policy. This is generally accomplished by an external (hardware or software) system, that interacts with the target basic system, monitoring its operations, and stopping it in the event of a security policy violation [32, 5]. Again, Aspect-orientation provides the means to include these features within the system, though without over-modifying it [21, 19].

When security policies are designed by multiple authors (something not uncommon in distributed systems), the various security goals could be conflicting. Solving conflicts among several security policies that aim to govern the same operation could be tricky. Moreover, analysing the expected result of a conflict handler could be very complicated. There has been some work proposing ways to deal with multi-author security policies in an effective way [10, 9]. These works are based on a four-valued Logic [4], which provides means for analysing the interaction of policies and for delaying the decision of whether to grant or deny access until all the relevant policies have been considered.

Recently, a framework named **AspectKB** [20] has been proposed, with which it is possible to model process calculi-like distributed systems. Furthermore, it allows capture of security properties in a realistic way: by attaching security policies to each location and then combining the relevant security policies when an interaction between locations takes place. The framework is also based on a four-valued Logic for solving conflicts. Finally,

since it follows an Aspect-oriented approach, it allows accomplishment of a reference-monitor-based enforcing style without having to implement a separate enforcing system.

However, the way of expressing security policies in the **AspectKB** framework refers to the traditional non-distributed information-flow [30] style of assuring security, which statically analyses the possible behaviours of the system in order to avoid any potential misuse in the future. In **AspectKB**, this is exploited by making access control decisions dynamically, yet not considering the current state of any location of the system, although possibly considering some potential future behaviour. This is opposite to the traditional reference-monitor-based approaches, as they are generally based on past behaviour and (thereby) on current state.

In this paper, we shall consider a multilevel access control policy [31], the Bell-LaPadula model [6]. We aim at showing some difficulties when capturing such a policy in a distributed framework. Moreover, in a framework whose security policies focus on *looking to the future*, the difficulties are even greater, since such multilevel policies are better suited for past analysis of how the system reached its current state. We shall see that some precision is missing by not considering past behaviour.

We then propose an extension to the **AspectKB** framework, and also allow expression of policies that *look to the past*. We do this by adding the notion of a *localised state* to the locations modelled in the extended framework and by allowing the security policies accessing those states to make their decisions. With this, multilevel policies such as the Bell-LaPadula policy can be easily and precisely captured, and we show how.

Since the original **AspectKB** framework was already intended to combine different security policies, with the extension done in this paper, both policies that look to the past and policies that look to the future can be expressed and even combined. This is not only beneficial when trying to capture specific policies (such as the Bell-LaPadula policy), but it also allows modelling of every policy in its original way, providing more flexibility in the resulting extended framework. Moreover, we shall argue that, for some situations, expressing a policy in its original way could more precisely capture what is intended, and this insight would mean that our extended framework is also more powerful.

Finally, we propose a Logic for reasoning about the global behaviour of the distributed system. With this Logic, which follows an ACTL (Action-based Computation Tree Logic [28]) approach, the intended global security

property of the entire distributed system can be expressed. Then, the property can be validated against the Labelled Transition System (LTS) induced by the Semantics of our extended version of the **AspectKB** language.

We shall start by informally discussing precision while capturing a policy and then discuss our approach in the remainder of this Section (Section 1.1). In Section 2 we present a review of the Bell-LaPadula policy in its original formulation, and then we assess the challenges of adapting it to a distributed setting. Section 3 gives a brief review of the Logic used for dealing with the combination of policies. In Section 4 we present our extended framework, and show how to precisely capture the Bell-LaPadula policy. We also discuss why the resulting framework is more flexible than the existing one. In Section 5 we present our variant of the ACTL Logic used for validating our distributed systems. We also discuss a faster way to perform this model checking, without exploring the entire LTS, thereby overcoming the state-explosion problem. In Section 6 we conclude.

1.1 Limitations of Looking to the Future

The framework we shall be dealing with throughout this paper is the formal language **AspectKB**. In this framework, which follows a process algebraic approach, the processes are modelled as actions taking place in specific locations, and interacting with other locations modelled as well. Furthermore, security policies can also be modelled following an Aspect-oriented manner. The policies can express their intentions by analysing the continuation (namely the process after the current action) of the processes involved, making it possible to know in advance what a process might do in the future. This reflects an information-flow style of providing security.

However, this style is not as adequate as it is for sequential programs. Indeed, the only process that can be analysed statically is the one that continues after the current action, since the others could be interleaved at any possible point during runtime. As a result, the possible outcomes that may occur during runtime due to other processes could not be predicted statically. This means that, deciding whether or not to allow the interaction to happen has to be done by looking to the future of just one process, namely the one involved. This can lead to two possible ways of obtaining imprecise decisions, either *over-approximation* or *under-approximation*.

In order to understand what over-approximation is, let us assume we *pessimistically* expect that a particular action done by a process could, because of other processes we do not know, lead to an insecure state. Then

we may disallow the process to execute that action, but in some cases there might be no other process performing anything that could lead to an insecure state.

In order to understand what under-approximation is, let us assume we *optimistically* expect that a particular action done by a process will not lead to an insecure state because the very same process will not perform another related action that leads to such a state. Then we may allow the process to execute that action, but in some cases there might be some other process that makes the system reach some insecure state, due to some interactions that could have been avoided, if the action was disallowed.

In some cases, the information-flow approach must be taken. If this is intended by the security policy to be captured, then over- (resp. under-) approximating the behaviour is correct. But, if the security policy to be captured expresses some *precise* situations where the interactions must be allowed / disallowed, then if we over- (resp. under-) approximate them, it would imply that we are missing some precision.

Let us discuss a simple example, without going into syntactic and semantic details, but still thinking about distributed processes and policies.

Let us think about a security policy where we have different security levels, and every location is assigned to some level. We do not want any information to be leaked from any security level to lower ones. Then, we should allow a process, running in a given location, to read data from another location, as long as the following two conditions are met: first, the other location, where the data is right now, is in a security level not higher than the one where the process is running; second, the process will not try, in the future, to write information to locations with security levels lower than the level of the location where the data is right now, since this writing may be influenced by the reading previously done.

Let us assume now a particular situation where we have 4 locations (say A, B, C and D), and 3 security levels (say 1, 2 and 3). Let us assume the security levels are ordered as their values in natural numbers ($3 > 2 > 1$). Let us assume that location A is in security level 1, locations B and C are both in security level 2, and location D is in security level 3. Figure 1 contains three cases of such a situation, showing the locations and their security levels in different layers.

Illustrated in Figure 1a, there is a process in location D, that tries to read information from location B at t1, and then tries to write some information to location A at t2. This process should clearly be forbidden,

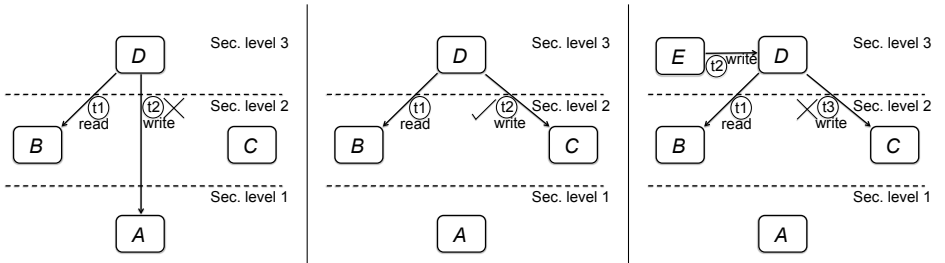
because it does not meet the second condition of the policy we are trying to capture (although it meets the first one). This can of course be done following the information-flow approach, looking to the future at t1, since we know that the process trying to read from B will try to write to A, and this should not be allowed.

However, let us think about another case, illustrated in Figure 1b. Let us say that the process running in location D, whose first action is to read information from location B at t1, then tries to write some information to location C (in the same level as B) at t2. This *does* meet the second condition of the policy, since the only information the process could write to C is what it has read from B. Therefore, this should be allowed.

Now let us consider the next extension to the example, illustrated in Figure 1c. Assume there is a fifth location E that is in security level 3. Assume there is a process running in E that writes some information to D at t2, after the process running in D has read from B at t1. In this case, the future writing to C by the process running in D (which in this case will be done at t3) should be forbidden, because it might be influenced by the new information learned by location D at t2. Anyway, since the process that writes to C is not the same as the one running in D, the process-algebraic way of modelling does not permit us to know in advance (at t1) that this will happen. If we had taken an approach looking to the past, then we would have checked the insecure operation of writing to C right in the moment of the writing (at t3), and we would have known that some information from E was leaked, thereby avoiding the write operation.

We could take the information-flow approach using over-approximation, and always avoid this type of write operation (e.g. from D to C, since the former is in level 3 while the latter in level 2), but that would be very imprecise (and restrictive), since sometimes there is nothing insecure in doing that write operation, as shown in the case of Figure 1b. Taking the information-flow approach using under-approximation would mean allowing the process in D to perform the read and the subsequent write, since this write operation is not insecure. This will be secure enough in the case of Figure 1b, but not in the case of Figure 1c.

Hence, we have found some possible situations where using an information-flow approach in a distributed setting is not completely precise, and therefore another approach might be taken, for instance looking to the past. In the rest of this work, we shall be studying how to deal with looking to the past, and how to extend our distributed systems framework to achieve this. We



(a) Insecure case, detectable by information flow.

(b) Secure case, but over-approximation would incorrectly disallow it.

(c) Insecure case, but under-approximation would incorrectly allow it.

Figure 1: Examples of situations that might happen.

shall see that the resulting framework allows us to combine both approaches, therefore obtaining the advantages of both of them. In particular, we shall see that the simple example we have seen is just one possible instance of something that can be easily (and more precisely) captured by the Bell-LaPadula policy.

1.2 Discussion

This work extends the framework of [20] with historic information, with which it is possible to precisely capture security policies that rely on this. This part of the work was already included in [22], but here we give more detail in some Sections, in particular 2.3, 3.1, 4.1, 4.2 and 4.4, and we also include/modify some Tables. Section 5 is completely new with respect to [22], and it follows an original idea also proposed in [20] but with a twist: some differences appear on how the Semantics are defined; they are defined compositionally and interpreted directly over the induced Labelled Transition System (LTS). Besides, the proposal to validate the Semantics without exploring the entire state space has not been even mentioned before.

Our language is an extension of the coordination language KLAIM [26], which follows a tuple space distribution with respect to the "Linda" approach to coordination [16]. However, our development is indeed useful in a range of process calculi, as we mainly focus on the security policies we attach to the locations in the form of Aspects, and moreover we validate the system using them. Finally, as this is a work on security policies and not on security protocols, we do not mention the concept of attacker at all. Indeed, a security policy could (implicitly) encode some types of attacks by forbidding some

possible interactions that are assumed to happen in the presence of some known attacker, but the concept of Dolev-Yao attacker falls beyond our focus of attention.

1.2.1 Related Work

In [30], a comprehensive survey on Information Flow is given, though mainly focused for sequential programs. Anyway, some research directions on enforcing security for distributed systems are mentioned. [29] develops a type-system-based approach to statically enforcing security in distributed systems. There, it is argued that the type system is not over-restrictive, implicitly assuming that some precision might (and must) be lost.

[13] suggests that most traditional security properties can be encoded in a process-calculi manner. This supports our approach, as we could model some properties, and later we could even validate them. Later, [14] shows how to encode some traditional cryptographic-protocols attacks. This is done by explicitly modelling the man-in-the-middle behaviour. We have said that we do not approach the problem assuming a Dolev-Yao attacker, but we could explicitly model some specific attacker as part of the system, and validate the resulting system. Therefore, if we find a way to model in our framework the man-in-the-middle behaviour following the lines of [14], we should be able to deal with the examples shown there. We could even deal with the Chinese Wall [8] security policy. Although we do not explicitly show how in this paper, we suggest it is possible with a small example in the conclusion of Section 6.

Some early work suggests that the problem of globally understanding systems consisting of many locations needs attention [11, 12]. In our case, we focus on security of such systems. More precisely on access-control-based security. In [1], a formal approach to model such a problem is given, although history is not explicitly discussed. Later, [2] proposes a way to keep the historic information of components requesting access. The approach is shown to be stronger than stack inspection [15] since it *remembers* what a method did even after termination. This is very similar to what we propose in this paper, as we keep information about past behaviour of processes according to interactions, whereas they keep such information according to method calls. Moreover, they use the concept of *rights* and their *intersection*, while we use the concept of *security levels* and their *least upper bound*. However, unlike us, they give no way to validate systems.

2 Assessment of the Bell-LaPadula Model

In Section 1.1 we saw that, for distributed systems, the information-flow approach is not as adequate as it was for sequential programs. In this Section, we review another approach, the Bell-LaPadula (BLP) policy, and discuss the challenges of using it in a distributed setting, while aiming to show that this *can be* as adequate as in its original formulation.

2.1 The Operating System View of BLP

The BLP model is the most traditional Mandatory Access Control model. Here we briefly introduce it, inspired by [18], but abstracting some unnecessary details that do not contribute to our study.

State. The computer system will be checked for security by looking into its state. In order to represent this, some sets must be introduced:

- S is the set of subjects (processes in our setting) that may use the information stored in the system,
- O is the set of objects (pieces of information) stored in the system,
- $A = \{\texttt{read}, \texttt{write}\}$ is the set of operations a subject may do over an object,
- L is a lattice of security levels.

Apart from this, there are three functions that are fixed during the lifetime of the specific system. They are named f_S, f_C and f_O and their types are $S \to L$, $S \to L$ and $O \to L$. The functions are supposed to be total functions, and they will give, respectively, the maximum security level a subject can have (its *clearance*), the *current* security level a subject has[2], and the security level an object has (its *classification*). Without loss of generality, the functions can be considered as part of the state, yet keeping in mind that they will not change. Therefore, every state of the system is composed of a set of tuples of the form (s, o, a) (each tuple would mean that subject s is doing an a operation over object o), and of a tuple of functions (f_S, f_C, f_O). Formally, a state $(B, F) \in \mathcal{B} \times \mathcal{F}$, where $F = (f_S, f_C, f_O)$, and where:

[2] A subject can log into the system with a lower security level than its corresponding clearance. Once it did so, that security level cannot be changed until it logs in again.

- $\mathcal{B} = \mathbb{P}(S \times O \times A)$
- $\mathcal{F} = (S \to L) \times (S \to L) \times (O \to L)$

Policies. The BLP model specifies two properties that every state should meet in order to be considered secure.

- **ss-property**[3]**.** A state (B, F) satisfies this property iff $\forall (s, o, a) \in B : a = \texttt{read} \implies f_S(s) \geq f_O(o)$. This means that each object being read by a subject should be in a level not higher than the level the subject is able to reach, which is usually called *no read-up*.

- **$\star$-property**[4]**.** This property consists of two parts. A state (B, F) satisfies the first part (let us name it **$\star$-property.1**) of this property iff $\forall (s, o, a) \in B : a = \texttt{write} \implies f_O(o) \geq f_C(s)$. This means that each object being written by a subject should be in a level not lower than the level where the subject is currently in, which is usually called *no write-down*. On the other side, a state (B, F) satisfies the second part (let us name it **$\star$-property.2**) of this property iff $\forall (s, o, a) \in B : a = \texttt{write} \implies [\forall (s, o', a') \in B : a' = \texttt{read} \implies f_O(o) \geq f_O(o')]$. This means that if a specific subject (note the use of the same s in both quantifications) is operating with many objects, some being read and some being written, then no object being read could be in a higher level than any object being written. This prevents the subject to read some high-level object and then write a low-level one.

A state is said to be *secure* if it satisfies both properties.

Figure 2 illustrates the properties. It should be clear that the purpose of allowing the *current* level to be lower than the clearance permits a subject to write to objects below its clearance, otherwise the **$\star$-property.1** would forbid this. The **$\star$-property.2** avoids a subject leaking information that must not be leaked if working with two objects between its clearance and its current level (i.e. between the dotted lines in Figure 2).

[3] For "simple security".

[4] Read "*star* property". In some formulations of the BLP model, this property only consists of the first part because the ss-property uses f_C instead of f_S, and then the second part is just a consequence. However, that kind of formulation is again too restrictive, since a subject cannot perform read operations in levels up to its clearance, but just up to the level it has logged in.

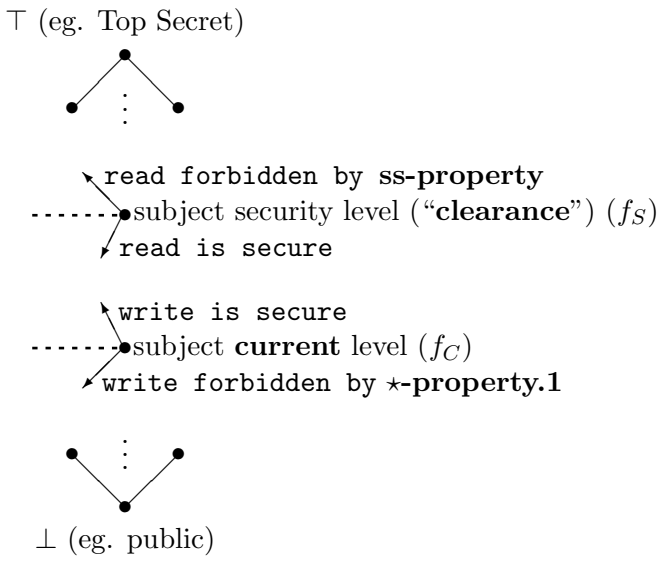


Figure 2: Schematisation of the Bell-LaPadula model.

2.2 The Challenges of Distribution

The BLP model was originally meant for Operating Systems. These have a particular feature: they are centralised, this means that a central controller (i.e. the Operating System) takes care of everything that happens in the system. In particular it can control (and in some cases restrict) the processes that try to access resources. Moreover, one key concept needed to check BLP policy compliance is the *state*, and since Operating Systems have a centralised state, they can do the calculations to ascertain whether the BLP policy is met or not.

Lack of Central Controller. In a distributed setting we do not have any central controller, many locations run in parallel and share information, but no location could know what other locations are doing. Therefore, once a location is allowed access to some resource, there is no way the other locations can forbid it to do whatever it wants with the resource. In particular, there is no notion of state, the processes interact and synchronise, but no central entity knows what has happened in the whole system so far.

It should be clear that a distributed framework is not trivially able to meet security properties that were originally developed for simpler systems, such as centralised systems or sequential programs. In the case of Information Flow, we have seen some simple examples where we can lose precision. In the case of BLP, we propose in the next Subsection an extension that will help us to adapt the policy to a distributed setting.

2.3 Extending BLP

The original formulation of the BLP policy relies on three functions, which can be computed by the Operating System every time an action is to be executed. If the resulting state will be secure, then the action is allowed, otherwise it is not. Two of the functions can be applied to every subject and one to every object. In a setting without a central controller we may want to call any of them with any possible entity of the system without distinguishing between objects and subjects. Here we propose an extension to their domains in order to have common signatures. We also propose a fourth function which captures information about the past interactions for each entity. Later in the paper we will see that this latter function can be used to have a form of localised state.

As for the existing functions, their types are changed to $S \cup O \to L$ for all of them, and their definitions are extended in a straightforward way as follows:

$$\forall o \in O, s \in S : f_S(o) = f_O(o) \wedge f_C(o) = f_O(o) \wedge f_O(s) = f_S(s)$$

As for the new function, we will call it f_H since it keeps track of (a part of) the *history* of the system. When we apply this function to a particular input subject (resp. object) we should learn what kinds of interactions the subject (resp. object) has been involved in during the past. Therefore, the output of the function would be a kind of *current state* of the argument subject (resp. object). To capture this notion of state, the function will not be fixed once and for all, as the original three functions were. Indeed, the output of this function will be:

- (*case1*) For a particular subject: the least upper bound of the security levels of all the objects read by the subject so far.
- (*case2*) For a particular object: the least upper bound of the (logged-in) security levels of all the subjects that have written to the object so far.

For a subject, the value of the historic component can therefore range between its initial value and that of the subject's clearance[5]. A lower value cannot be possible due to the least upper bound restriction, and a greater value cannot be possible because it would mean the **ss-property** was violated. Analogously, for an object, the value of the historic component can range between its initial value and that of the object's security level[6]. A greater value would mean the **⋆-property.1** was violated.

Furthermore, the historic components are *non-decreasing*, and this means that every time there is a state change from $(B, (f_S, f_C, f_O, f_H))$[7] to some $(B', (f_S, f_C, f_O, f'_H))$ due to some interaction, the output of f'_H for some input may be higher or equal to that of f_H. This is straightforward by the following simple lattice-theory result:

$$\sqcup(\mathcal{L}) = \sqcup\{\sqcup(\mathcal{L} \setminus \{a\}), a\} \qquad (\forall a \in \mathcal{L}) \tag{1}$$

This establishes that the least upper bound obtained from a set $\mathcal{L}$ ($\subseteq L$) is the same as the one obtained from $\mathcal{L}$ minus any element and that very same element. We should also observe that $\sqcup(\emptyset) = \bot(\in L)$.

New Property The property expected from a system to satisfy this extended BLP policy would be that a subject is only allowed to read objects with historic component lower than or equal to the subject clearance. Conversely, a subject would only be allowed to write objects with historic component greater than or equal to the subject's historic component. These properties would replace the **⋆-property.2** since, instead of information about the objects being operated by a subject right now, we now have information about the past behaviour of the subject (process).

Formally, capturing the notion of state and the conditions that have to be satisfied can be expressed as follows:

$$\begin{aligned} \forall (s,o,a) \in B : \quad & (\ (a = \texttt{read} \implies f_H(s) \geq f_O(o) \\ & \qquad \wedge f_H(s) \geq f_H(o) \\ & \qquad \wedge f_S(s) \geq f_H(o)) \ \wedge \\ & \ (a = \texttt{write} \implies f_H(o) \geq f_C(s) \\ & \qquad \wedge f_H(o) \geq f_H(s) \\ & \qquad \wedge f_O(o) \geq f_H(s)) \ ) \end{aligned} \tag{2}$$

[5] Assuming the initial value is lower than or equal to the clearance.

[6] Assuming the initial value is lower than or equal to the security level.

[7] Assuming $(B, (f_S, f_C, f_O, f_H))$ to be some *"virtual" global* state that depends on the interactions that have happened.

The first conjunct establishes that if a subject reads an object, then the historic component of the subject is greater than or equal to both the security level and the historic component of the object. Also, the clearance of the subject is greater than or equal to the historic component of the object. The first and third inequalities mimic the **ss-property** (and they are both indeed necessary, as here we are considering non-fixed information). The second inequality relates the history of both entities, to keep the track from both sides. In the second conjunct, something analogous is established: the first and third inequalities mimic the **⋆-property.1** and the second relates both histories.

We shall use these four functions to capture this extended version of BLP in a distributed setting.

3 A Brief Review of Belnap Logic

In order to grant access according to some security policy, the traditional Boolean values (**tt** and **ff**) are enough: **tt** grants while **ff** denies access. However, for a distributed setting, where policies might be contradictory (or not sufficiently informative), these two values might not be enough. We shall consider an extension to the Boolean Logic proposed by Belnap [7, 4], which has been used to combine security policies [10, 9].

In this extension to the Boolean logic, two more values are considered: $\bot$ and $\top$ (read "bottom" and "top"). The traditional **tt** would mean "the policy accepts the interaction" whereas the traditional **ff** would mean "the policy does not accept the interaction". Since different locations might aim at different security properties, their policies could be contradictory or they may lack information about some particular interaction. These situations can be represented by the two extra values that we have: $\bot$ meaning "no decision" and $\top$ meaning "contradiction".

With this set of values, which we will call here **Four** (i.e. **Four** $= \{\bot, \mathbf{tt}, \mathbf{ff}, \top\}$), it is possible to extend the usual Boolean operators ($\wedge$ and $\vee$) and to define new ones ($\otimes$ and $\oplus$). In order to obtain this, the set **Four** is equipped with two partial orderings, say $\leq_k$ and $\leq_t$, as shown in Figure 3.

The usual Boolean $\wedge$ is extended as computing the greatest lower bound in the $\leq_t$ lattice, and the usual $\vee$ as computing the least upper bound (thereby obtaining the same results as in Boolean logic in the event that the operands belong to $\{\mathbf{tt}, \mathbf{ff}\}$). Analogously, the new operators over **Four** can be defined as computing the greatest lower bound (the $\otimes$ operator) and the

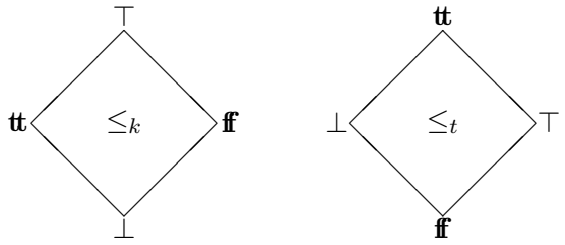


Figure 3: The Belnap bilattice **Four**: $\leq_k$ and $\leq_t$.

least upper bound (the $\oplus$ operator), both in the $\leq_k$ lattice.[8]

The negation operator $\neg$ is extended by leaving the two new values unchanged (i.e. $\neg\bot = \bot$ and $\neg\top = \top$). This is done in this way because negation of a conflict would mean negating each of the conflicting policies, and that would again result in a conflict. Similar reasoning holds for $\bot$.

For the implication $\Rightarrow$, we have to consider that having no information about a situation would in principle mean not caring about it, so **ff** is not present. Analogously, having a conflict means having (at least) some reason for denying it, so **ff** is present (as well as **tt**, of course). If that situation happens in the antecedent of the implication, one can easily figure out how to extend the operator, defined as follows:

$$p_1 \Rightarrow p_2 = \begin{cases} p_2 & \text{if } p_1 \leq_k \mathbf{tt} \\ \mathbf{tt} & \text{otherwise} \end{cases} \qquad \forall p_1, p_2 \in \mathbf{Four}$$

Another useful operator is the priority $>$, which returns the first operand unless it is $\bot$, in which case it returns the second operand. This would always consider what the first operand suggests unless it has no decision, in which case the second operand is considered.

[8]Notice that this could also be done by just extending the "truth tables" of the usual Boolean operators and defining new ones for the new operators. However, this would mean having not just 2 truth tables with 4 cells each but 4 truth tables with 16 cells each, making it difficult for a human to remember what each operator produces. Furthermore, it would also make it difficult to *assess* the usefulness of new operators that might be defined.

3.1 Power of Belnap Logic

As discussed in several places [10][20], Belnap Logic is enough for dealing with conflicting policies, and therefore it is appropriate for our intended use of combining several policies occurring in different locations of a distributed system.

The decision we take is that, in order to allow an interaction, no policy should recommend that the interaction should be denied, otherwise we just consider the *negative* policy and deny the interaction. This is in line with the definition of the implication operator $\Rightarrow$ and it follows a *conservative* approach, since an interaction is denied as long as there is one policy that suggests so. The proper Belnap operator for combination of policies, which could indeed be used for achieving this aim, is $\oplus$. This will be made clearer below.

Another issue that is worth mentioning about Belnap Logic is that it is not capable of expressing *voting*. Indeed, if one aims at some combination of policies where no conservative approach (or some other approaches with which Belnap Logic could cope) is taken, but some voting among the involved policies is intended, aiming at taking the decision most of the policies agree on, then Belnap Logic is not the right choice.

4 An Aspect-oriented Framework for Security

As mentioned, the **AspectKB** [20] framework allows us to express location-based systems in a process-calculus-oriented manner. This is achieved by extending the KLAIM [26] coordination [16] language. These *located processes* interact with other locations when they try to gather (or put) information from (or into) them (maybe themselves). The locations holding data are usually named as *tuple spaces*. Every location in the system may have either processes or data, or both. The possibility of attaching some security policy to each location turns the **AspectKB** language into an Aspect-oriented language. Then, whenever an interaction takes place, the relevant policies are considered by the semantics to either grant or deny the interaction, using the four-valued Belnap Logic for combining policies and deciding in a consistent way.

In this Section, an extension to that framework is made, mixing all process locations and tuple locations into just *entity locations*, and attaching to them more Aspects than just the security policies. The extra information attached to each location refers to security levels in the sense of a multi-

$N \in \mathbf{Net}$	N	$::=$	$N_1 \,\|\|\, N_2 \mid l ::^w P \mid l ::^w \langle \overrightarrow{l} \rangle$	
$P \in \mathbf{Proc}$	P	$::=$	$P_1 \mid P_2 \mid \sum_i a_i.P_i \mid *P$	
$a \in \mathbf{Act}$	a	$::=$	$\mathbf{out}(\overrightarrow{\ell}) @ \ell \mid \mathbf{in}(\overrightarrow{\ell^\lambda}) @ \ell \mid \mathbf{read}(\overrightarrow{\ell^\lambda}) @ \ell$	
$\ell, \ell^\lambda \in \mathbf{Loc}$	ℓ	$::=$	$u \mid l$	$\ell^\lambda ::= \ell \mid !u$
$w \in \mathbf{Annot}$	w	$::=$	$< lst, pol >$	
$lst \in \mathbf{LocSt}$	lst	$::=$	$< \gamma^S, \gamma^C, \gamma^H, \gamma^O >$	
$\gamma \in L$				

Table 1: **AspectKB+** Syntax – Nets, Processes, Actions and States.

level security policy. Moreover, the mechanisms of this extended language explicitly keep track of some information (at a certain level of abstraction) regarding the interactions that have taken place so far. This gives the flavour of a *localised state*, which the semantics of the language keep updated[9].

One can also write Aspects using that extra information, which is basically the output of the functions mentioned in Section 2.3 (considering that every entity location can be either a subject and/or an object in the whole system, so every location can be a potential input to all those functions). This will then allow us to capture, among others, the BLP policies without losing precision.

Following this informal introduction to our extension, which we shall call **AspectKB+** due to its enhanced features, we present its formalities.

4.1 Syntax

The syntax of the **AspectKB+** language is given in Tables 1 and 2. Table 1 gives the syntax for nets, the basic modules that can be described in the language, and here we explain it informally. A net belongs to the set **Net** and can be a parallel composition of (smaller) nets, or it can be the basic located processes and/or located tuples (data), together with an annotation, w, explained below. As in KLAIM, locations are first-class data so they can be stored in the tuples. Each process belongs to **Proc** and can

[9]As one can argue, having information inside the locations, namely the tuples, also gives us the flavour of state. However, that is information that changes according to what processes do, and not due to the semantics of the language, so we cannot rely on that information for guaranteeing any property.

$pol \in \mathbf{Pol}$	pol	$::=$	$asp \mid \neg pol \mid pol \oplus pol \mid pol \otimes pol \mid pol \Rightarrow pol \mid$
			$pol > pol \mid pol \wedge pol \mid pol \vee pol \mid \mathbf{true} \mid \mathbf{false}$
$asp \in \mathbf{Asp}$	asp	$::=$	$[rec \ \underline{\text{if}} \ \ cut : cond]$
$cut \in \mathbf{Cut}$	cut	$::=$	$\ell :: a^t \,.\, X$
$a^t \in \mathbf{Act}^t$	a^t	$::=$	$\mathbf{out}(\overrightarrow{\ell^t}) \,@\, \ell \ \mid \ \mathbf{in}(\overrightarrow{\ell^{t\lambda}}) \,@\, \ell \ \mid \ \mathbf{read}(\overrightarrow{\ell^{t\lambda}}) \,@\, \ell$
$rec \in \mathbf{Rec}$	rec	$::=$	$\ell_1 = \ell_2 \mid \neg rec \mid rec \oplus rec \mid rec \otimes rec \mid$
			$rec \wedge rec \mid rec \vee rec \mid rec \Rightarrow rec \mid \mathbf{true} \mid \mathbf{false} \mid$
			$a \ \mathbf{occurs\text{-}in} \ X \mid v_1 \geq v_2$
$cond \in \mathbf{Cond}$	$cond$	$::=$	$\ell_1 = \ell_2 \mid \neg cond \mid cond_1 \wedge cond_2 \mid cond_1 \vee cond_2 \mid$
			$\mathbf{true} \mid \mathbf{false} \mid \ a \ \mathbf{occurs\text{-}in} \ X$
$v \in \mathbf{Lev}$	v	$::=$	$S_s \mid C_s \mid H_s \mid O_t \mid H_t \mid \gamma$
	ℓ^t	$::=$	$\ell \mid \_ \qquad\qquad \ell^{t\lambda} ::= \ \ell \mid \ \_$

Table 2: **AspectKB+** Syntax - Aspects for Security Policies.

be a parallel composition of processes, a non-deterministic choice between processes following an action, or a replicated process (denoted by $*$). For simplicity, we distinguish the symbol for parallel composition of nets, $||$, from the one for processes, $|$. A process not performing any action will be written **0**. The possible actions are reading from a location (**in** and **read**, resp. with or without deleting the data read) or writing to it (**out**), and they belong to **Act**. In the reading actions, we can bind a variable using !, or just mention a previously bound one or a constant, just as in the writing actions.

Every location has an annotation w, which belongs to **Annot** and consists of two parts. The first part (lst) is intended to keep track of the interactions that the location has been involved in so far. This *localised state* belongs to **LocSt** and consists of 4 pieces of information (namely $\gamma^S, \gamma^C, \gamma^H$, and γ^O) that are elements of the lattice L of security levels (introduced in Section 2.1). Since every location can be input to the four functions f_S, f_C, f_H and f_O, and since the result of evaluating them is in L, we can keep attached to each location the result of evaluating each of those four functions. We do not consider these values first-class data, and therefore the vector (not tuple) of values is written with $< . >$ instead of $\langle . \rangle$.

The second part (pol) of the annotation in every location is the actual security policy governing the location, which follows the syntax of Table 2. A policy belongs to **Pol** and can be a Belnap combination of policies, a Boolean

$$
\begin{aligned}
a \textbf{ occurs-in } (P_1 \mid P_2) &= (a \textbf{ occurs-in } P_1) \vee (a \textbf{ occurs-in } P_2) \\
a \textbf{ occurs-in } (\textstyle\sum_i a_i.P_i) &= \textstyle\bigvee_i (a \textit{ matches } a_i \vee a \textbf{ occurs-in } P_i) \\
a \textbf{ occurs-in } (*P) &= a \textbf{ occurs-in } P \\
a \textbf{ occurs-in } (\mathbf{0}) &= \mathbf{ff}
\end{aligned}
$$

Table 3: Continuation analysis operator **occurs-in**.

value, or a single Aspect *asp* that belongs to **Asp**. This latter consists of a *cut* (the action, together with its continuation, to be trapped by the Aspect), a condition *cond* (a Boolean applicability condition) and a recommendation *rec* (a four-valued Belnap Logic advice for the Aspect)[10]. While defining a recommendation $rec \in \mathbf{Rec}$, one may refer to the security levels stored in the trapped interaction or to a single value from the lattice L. To do the former, one can refer to some of the five syntactic constants $(S_s, C_s, H_s, O_t, H_t)$ specified in the category $v \in \mathbf{Lev}$, which will later be matched by the semantics to the specific values kept in the trapped interaction. To do the latter, one can provide a specific value from L, as the category $v \in \mathbf{Lev}$ permits (by having γ among its choices). Finally, the **occurs-in** operator, defined inductively in Table 3, checks whether the action occurs in the continuation process. We omit the (trivial) definition of $a \textit{ matches } a_i$, which is the source of a **tt**.

Running Example Assume that in a given Hospital we have a Health Care System where there is a data base, named EHDB (for Electronic Health Data Base), with some information about some patients. In this case, let us assume there is one tuple (piece of data) regarding Alice, and that the tuple specifies a given Care Plan for her. In addition, there is another tuple regarding Bob, and it is related to some Private Notes some Doctor might have taken about him. This can be written in **AspectKB+** as follows:

$$
\begin{aligned}
NetData = \quad & EHDB ::^{w_{EHDB}} \langle \texttt{Alice, CarePlan, alicetext} \rangle \ \| \\
& EHDB ::^{w_{EHDB}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle
\end{aligned}
$$

Note that, according to the syntax, although both tuples are in the same location, it must be written explicitly both times (and as a net composition).

[10]Note that we use **true** and **false** for syntactic constants whereas **tt** and **ff**, as did previously, for Boolean (or Belnap) values.

Note also, that each of the occurrences of the location has an annotation w_{EHDB} attached. Let us omit for now the definition of this latter.

Assume now that there is also another location with information about the staff of the Hospital, which could be defined in the following way:

$$\begin{aligned} NetRoles = \quad & ROLES ::^{w_{ROLE}} \langle \texttt{Doctor, Hansen} \rangle \ || \\ & ROLES ::^{w_{ROLE}} \langle \texttt{Nurse, Olsen} \rangle \end{aligned}$$

Now, assume that both employees have some location, and there is a Process running on each of them. Doctor Hansen might try to read patient Bob's information, and then leak it to Nurse Olsen. On her side, Nurse Olsen might also try to read Bob's information directly. This could be defined as follows:

$$\begin{aligned} NetHansen = \quad & Hansen ::^{w_{staff}} \\ & \quad \mathbf{read}(\texttt{Bob, PrivateNotes}, !content)@EHDB. \\ & \quad \mathbf{out}(\texttt{Bob, PrivateNotes}, content)@Olsen.\ \mathbf{0} \end{aligned}$$

$$\begin{aligned} NetOlsen = \quad & Olsen ::^{w_{staff}} \\ & \quad \mathbf{read}(\texttt{Bob, PrivateNotes}, !content)@EHDB.\ \mathbf{0} \end{aligned}$$

Finally, the entire network could be defined using the previous definitions as follows:

$$NetData \ || \ NetRoles \ || \ NetHansen \ || \ NetOlsen \tag{3}$$

4.2 Semantics

The semantics are given by a one-step reduction relation on nets whose rules are defined in Table 4 and explained below. Some auxiliary inference rules are given in Table 5, to make the reaction rules simpler. They make use of a structural congruence relation on nets, consisting of the usual congruence rules and those given in Table 6. The semantics also make use of an operator *match*, for matching input patterns to actual data, defined in Table 7. With this, $\rightarrow$ is a relation from **Net** to $\mathbf{Lab}^{11}$ to **Net**. The relation $\rightarrow$ defines from which nets we can move to other ones and what the label of the transition is, and it induces a Labelled Transition System (LTS). Also, $\equiv$ is a relation from **Net** to **Net**, and it defines which pairs of net expressions actually identify the same net.

[11] In Section 5.1 it shall be clear why we need labels in the transitions.

$[Rule - read]$
$(l_s ::^{w_s} \mathbf{read}(\overrightarrow{\ell^{\lambda}})@l_t.P) \;||\; (l_t ::^{w_t} \langle \overrightarrow{l} \rangle)$
$\quad \rightarrow^{l_s(w_s):\mathbf{r}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w'_s} P\theta \;||\; l_t ::^{w_t} \langle \overrightarrow{l} \rangle \qquad \text{if } b \;\wedge\; match(\overrightarrow{\ell^{\lambda}}; \overrightarrow{l}) = \theta$

$\quad$ where $w_\delta =< lst_\delta, pol_\delta >, \qquad (\delta \in \{s,t\})$;
$\quad$ and where $b = \mathsf{grant}([\![pol_s \oplus pol_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^{\lambda}})@l_t.P, lst_s, lst_t))$;
$\quad$ and where $w'_s = w_s[(\gamma_s^H \sqcup (\gamma_t^O \sqcup \gamma_t^H))/\gamma_s^H]$.

$[Rule - in]$
$(l_s ::^{w_s} \mathbf{in}(\overrightarrow{\ell^{\lambda}})@l_t.P) \;||\; (l_t ::^{w_t} \langle \overrightarrow{l} \rangle)$
$\quad \rightarrow^{l_s(w_s):\mathbf{i}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w'_s} P\theta \qquad \text{if } b \;\wedge\; match(\overrightarrow{\ell^{\lambda}}; \overrightarrow{l}) = \theta$

$\quad$ where $w_\delta =< lst_\delta, pol_\delta >, \qquad (\delta \in \{s,t\})$;
$\quad$ and where $b = \mathsf{grant}([\![pol_s \oplus pol_t]\!](l_s :: \mathbf{in}(\overrightarrow{\ell^{\lambda}})@l_t.P, lst_s, lst_t))$;
$\quad$ and where $w'_s = w_s[(\gamma_s^H \sqcup (\gamma_t^O \sqcup \gamma_t^H))/\gamma_s^H]$.

$[Rule - out]$
$(l_s ::^{w_s} \mathbf{out}(\overrightarrow{l})@l_t.P) \;||\; (l_t ::^{w_t} Q)$
$\quad \rightarrow^{l_s(w_s):\mathbf{o}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w_s} P \;||\; l_t ::^{w'_t} \langle \overrightarrow{l} \rangle \;||\; l_t ::^{w_t} Q \qquad \text{if } b$

$\quad$ where $w_\delta =< lst_\delta, pol_\delta >, \qquad (\delta \in \{s,t\})$;
$\quad$ and where $b = \mathsf{grant}([\![pol_s \oplus pol_t]\!](l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P, lst_s, lst_t))$;
$\quad$ and where $w'_t = w_t[(\gamma_t^H \sqcup (\gamma_s^C \sqcup \gamma_s^H))/\gamma_t^H]$.

Table 4: Reaction Semantics of **AspectKB+** .

The three reaction rules of Table 4 prescribe how the system may evolve in the presence of some *process location* and some *target location*. For this purpose, each rule only defines a transition if the policies agree on allowing the interaction to take place. This is the purpose of calling the auxiliary function $\mathsf{grant}()$, with the $\oplus$ combination of the involved policies, together with the intended action and the localised state of the involved locations. The function $\mathsf{grant}()$ turns the four-valued policies' recommendations into an actual Boolean decision. The way this is achieved will be explained further in Section 4.3. When the interaction is actually allowed by the policies, the transition is performed (subject to the existence of some actual data in the

$$\frac{N_1 \to N_1'}{N_1 \,||\, N_2 \to N_1' \,||\, N_2} \qquad \frac{N \equiv M \quad M \to M' \quad M' \equiv N'}{N \to N'}$$

$$\frac{l_s ::^w a_1.P_1 \,||\, N \to N_1}{l_s ::^w a_1.P_1 + a_2.P_2 \,||\, N \to N_1} \qquad \frac{l_s ::^w a_2.P_2 \,||\, N \to N_2}{l_s ::^w a_1.P_1 + a_2.P_2 \,||\, N \to N_2}$$

Table 5: Semantics of **AspectKB+** (auxiliary).

$$\begin{aligned} l ::^w P_1 \mid P_2 &\equiv l ::^w P_1 \,||\, l ::^w P_2 \\ l ::^w *P &\equiv l ::^w P \mid *P \\ l ::^w P &\equiv l ::^w P \,||\, l ::^w \mathbf{0} \end{aligned} \qquad \frac{N_1 \equiv N_2}{N \,||\, N_1 \equiv N \,||\, N_2}$$

Table 6: Structural Congruence.

case of a **read** or **in** action).

In rules $[Rule - read]$ and $[Rule - in]$, if the action is granted then the process location l_s is subject to a substitution θ, using the result of the matching done with the *match* operation. Moreover, according to the last line of each rule, the localised state of that location might be modified, changing the historic component of its annotation by the least upper bound of the previous value and the security level of the target location l_t. This follows the suggestion of Equation (1).

In rule $[Rule - out]$, if the action is granted then the data is stored in the target location. However, this is not done directly, but actually another "virtual" location is created, with a special localised state. This is intended to permit the virtual location holding the pre-existing process Q to keep running as it was, without being interfered with. The virtual location now holding the data has an historic component on the annotation that is the least upper bound of the previous value in the location l_t and the security level of the process location l_s that has written the data. It can of course happen to result the same as in the original l_t.

To simulate the log-in of a subject in a lower level than its clearance, a process can be annotated with a value for γ^C lower than the γ^S. This value will then never change, just as with the γ^S and γ^O components of the

$$\begin{array}{rcll} match(\,!u, \overrightarrow{\ell^{\lambda}}\,;\, l, \overrightarrow{l}\,) & = & [l/u] \circ match(\,\overrightarrow{\ell^{\lambda}}\,;\,\overrightarrow{l}\,) & \\ match(\,l, \overrightarrow{\ell^{\lambda}}\,;\, l, \overrightarrow{l}\,) & = & match(\,\overrightarrow{\ell^{\lambda}}\,;\,\overrightarrow{l}\,) & \\ match(\,\epsilon\,;\,\epsilon\,) & = & id & \\ match(\,\cdot\,;\,\cdot\,) & = & \mathsf{fail} & \text{otherwise} \end{array}$$

Table 7: Matching Input Patterns to Data.

localised state. Note also that the security policy annotating each location never changes either, and this is due to a *well-formedness* condition imposed on nets (inherited from **AspectKB**).

Example Given the network for the Hospital we could obtain, for instance, the following path according to the Semantics:

$$\begin{array}{l} \quad NetData \;||\; NetRoles \;||\; NetHansen \;||\; NetOlsen \\ \rightarrow^{Hansen(w_{staff}):\mathbf{r}(\texttt{Bob, PrivateNotes, bobtext})@EHDB(w_{EHDB})} \\ \quad NetData \;||\; NetRoles \;||\; NetOlsen \;|| \\ \qquad Hansen ::^{w_{staff}} \mathbf{out}(\texttt{Bob, PrivateNotes, bobtext})@Olsen.\mathbf{0} \\ \rightarrow^{Hansen(w_{staff}):\mathbf{o}(\texttt{Bob, PrivateNotes, bobtext})@Olsen(w_{staff})} \\ \quad NetData \;||\; NetRoles \;||\; NetOlsen \;|| \\ \qquad Hansen ::^{w_{staff}} \mathbf{0} \;|| \\ \qquad Olsen ::^{w_{staff}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle \\ \rightarrow^{Olsen(w_{staff}):\mathbf{r}(\texttt{Bob, PrivateNotes, bobtext})@EHDB(w_{EHDB})} \\ \quad NetData \;||\; NetRoles \;||\; Olsen ::^{w_{staff}} \mathbf{0} \;|| \\ \qquad Hansen ::^{w_{staff}} \mathbf{0} \;|| \\ \qquad Olsen ::^{w_{staff}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle \end{array}$$

This is the path that is followed if both actions from the process in location *Hansen* take place before the single action in location *Olsen*. There are also 2 other paths that are possible if the interleaving is different.

Example with History Let us modify slightly our example just to show how the historic information is updated according to the Semantics. Assume now that in the HealthCare Data Base, the tuples corresponding to Private Notes have a higher security level than those corresponding to Care Plans. Assume we use the set of natural numbers as our lattice L. Then, the

$NetData$ could be redefined as:

$$\begin{aligned} NetData' = \quad & EHDB ::^{<<\gamma^S_{EHDB},\gamma^C_{EHDB},1,1>,pol_{EHDB}>} \\ & \quad \langle \texttt{Alice, CarePlan, alicetext} \rangle \\ & \| \\ & EHDB ::^{<<\gamma^S_{EHDB},\gamma^C_{EHDB},2,2>,pol_{EHDB}>} \\ & \quad \langle \texttt{Bob, PrivateNotes, bobtext} \rangle \end{aligned}$$

where we omit the specific values of some parts of the annotation.

Assume our Doctor Hansen now has 2 processes, one for reading Alice's information and the other for reading Bob's. We could redefine it as:

$$\begin{aligned} NetHansen' = \quad & Hansen ::^{<<100,0,0,\gamma^O_{Hansen}>,pol_{staff}>} \\ & \quad \mathbf{read}(\texttt{Bob, PrivateNotes}, !content)@EHDB.\ P_1 \\ & \quad | \\ & \quad \mathbf{read}(\texttt{Alice, CarePlan}, !content)@EHDB.\ P_2 \end{aligned}$$

where we also omit the specific values of some parts of the annotation, and we assume Doctor Hansen has a very high clearance (γ^S_{Hansen} is 100), but he decided to log into the system with the lowest one (γ^C_{Hansen} is 0).

Now, the evolution of the processes could follow the next fashion:

$$\begin{aligned} & NetData' \,\|\, NetHansen' \\ \rightarrow \quad & \\ & NetData' \,\| \\ & \quad Hansen ::^{<<100,0,2,\gamma^O_{Hansen}>,pol_{staff}>} P_1\theta_1 \,\| \\ & \quad Hansen ::^{<<100,0,0,\gamma^O_{Hansen}>,pol_{staff}>} \\ & \qquad \mathbf{read}(\texttt{Alice, CarePlan}, !content)@EHDB.\ P_2 \\ \rightarrow \quad & \\ & NetData' \,\| \\ & \quad Hansen ::^{<<100,0,2,\gamma^O_{Hansen}>,pol_{staff}>} P_1\theta_1 \,\| \\ & \quad Hansen ::^{<<100,0,1,\gamma^O_{Hansen}>,pol_{staff}>} P_2\theta_2 \end{aligned}$$

where we omit the superscript for the transition relation $\rightarrow$ and also the details of the substitutions θ_1 and θ_2. Indeed, what we want to emphasise is that as soon as one of the processes in the location performs some action, its historic component could be increased. Moreover, if that happens, then if the process location has more than one process, it automatically "splits" into 2 separate locations, one holding each of the processes[12]. This certainly

[12]Furthermore, notice the different parallel composition operator.

$$
\begin{array}{rcl}
check(\,\alpha, \overrightarrow{\alpha}\,;\,\alpha', \overrightarrow{\alpha'}\,) & = & check(\,\overrightarrow{\alpha}\,;\,\overrightarrow{\alpha'}\,) \circ do(\alpha;\alpha') \\
check(\,\epsilon\,;\,\epsilon\,) & = & id \\
check(\,\cdot\,;\,\cdot\,) & = & \mathsf{fail} \text{ otherwise} \\
 & & \\
do(u\,;\,l) & = & [u \mapsto l] \\
do(!u\,;\,!u') & = & [u \mapsto u'] \\
do(X\,;\,P) & = & [X \mapsto P] \\
do(\_\,;\,l) & = & id \\
do(\_\,;\,!u) & = & id \\
do(l\,;\,l) & = & id \\
do(c\,;\,c) & = & id \\
do(\cdot\,;\,\cdot) & = & \mathsf{fail} \text{ otherwise}
\end{array}
$$

Table 8: Checking Formals to Actuals **AspectKB+**.

occurred in the first transition, but not in the second one, since the location had just one process at that point.

It is worth noting that this does not help at all, unless we have some security policies that use those values to prevent some possible transitions. For instance, we could have a policy that forbids writing to certain locations if the historic component is greater than a certain value, because this might mean that a reading of some high information has been done.

In the following, to keep illustrating the next developments throughout the paper, we will continue using our previous running example (i.e. the one without history), unless stated otherwise.

4.3 Meaning of Policies and Granting Access

In the "where" lines of each semantic rule from Table 4, there is a check that tells whether the interaction should be allowed. For this purpose, the policies of both locations taking part in the interaction are combined using the Belnap operator $\oplus$, and the result of the evaluation by the operator $[\![.]\!]$, defined below, is passed to the function grant.

The function grant is defined by $\mathsf{grant}(p) = p \leq_k \mathbf{tt}$, for all p in **Four**. The aim of granting access whenever the result is less than or equal to $\mathbf{tt}$ is to do so not only when both policies agree, but also when some of the policies lack some decision, because this would mean that it does not actually forbid the interaction. This is related to the use of $\oplus$ to combine the policies, and

$$[\![[rec\ \underline{\text{if}}\ \ cut : cond]\!](l :: a\,.\,P, lst_s, lst_t) \quad =$$
$$\left(\begin{array}{l} \text{case } check(\,extract(cut)\,;\,extract(l :: a\,.\,P))\ \text{of} \\ \qquad \mathsf{fail}: \quad \bot \\ \qquad \theta: \quad \left\{\begin{array}{ll} [\![((rec\ \theta)\theta')]\!] & \text{if } [\![(cond\ \theta)]\!] \\ \bot & \text{if } \neg[\![(cond\ \theta)]\!] \end{array}\right. \end{array}\right)$$

where $\theta' = [S_s \mapsto \gamma_S, C_s \mapsto \gamma_C, O_t \mapsto \gamma_O, H_s \mapsto \gamma_{Hs}, H_t \mapsto \gamma_{Ht}]$
and where $< \gamma_S, \gamma_C, \gamma_{Hs}, - >= lst_s$ and $< -, -, \gamma_{Ht}, \gamma_O >= lst_t$

$$\begin{array}{lcl} [\![\neg pol]\!](act, st1, st2) & = & \neg([\![pol]\!](act, st1, st2)) \\ [\![pol_1\ \phi\ pol_2]\!](act, st1, st2) & = & ([\![pol_1]\!](act, st1, st2))\ \phi\ ([\![pol_2]\!](act, st1, st2)), \\ & & (\phi \in \{\oplus, \otimes, \Rightarrow, >, \wedge, \vee\}) \\ [\![\mathbf{true}]\!](act, st1, st2) & = & \mathbb{tt} \\ [\![\mathbf{false}]\!](act, st1, st2) & = & \mathbb{ff} \end{array}$$

Table 9: Meaning of Policies in **Pol** for **AspectKB+**.

the aim is that whenever the policies are contradictory, we conservatively forbid the interaction, as discussed in Section 3.1. Indeed, we deny access as long as some policy has evidence that the interaction should be denied. This is achieved because the evaluation by $[\![.]\!]$ gives $\top \in \mathbf{Four}$.

The evaluation function $[\![.]\!]$ (Table 9) is defined inductively on the structure of the (infix) policy, which belongs to the syntactic category **Pol** from Table 2. The inductive cases are in the second and third lines. The base cases are when the policy is just a constant (**true** or **false**, last 2 lines) and when it is just an Aspect (i.e. it belongs to **Asp**, first line).

In this latter case, the first (postfix) parameter, a *specific* action with continuation[13], is checked against the *cut* of the Aspect, a *generic* action with continuation, using the function *check*, defined in Table 8 and explained below, to see if there is a substitution θ that unifies both actions. This is achieved using a function *extract*, which gives the list of literals occurring in an action with continuation in a way that, for instance, $extract(\ell :: \mathbf{out}(\ell_1^t, \cdots, \ell_n^t)@\ell'.X) = [\ell, \mathbf{out}, \ell_1^t, \cdots, \ell_n^t, \ell', X]$. We omit the formal definition of *extract*, which could easily be done by pattern matching the components of the given parameter and pushing them into a list. The second (postfix) parameter (consisting of the localised states of the involved locations, according to the rules in Table 4) is used to produce another

[13]Due to the rules in Table 4, this action will always be the actual action taking place.

substitution (θ') that is also used (together with θ) to determine the result.

While producing the substitution θ', we take from the target-location localised state the values identifying the *classification* (γ_t^O) of the location and the *historic* annotation (γ_t^H). Meanwhile, the values taken from the process location are the ones identifying the *clearance* (γ_s^S) and the *current level* (γ_s^C) of the location and the *historic* annotation (γ_s^H).

The function *check* (Table 8) determines whether there is a substitution θ that can be performed in the *cut* that matches the specific action with continuation given as parameter. Actually, it receives a list of literals, and it just pattern matches them one by one (using function *do*) in order to find a unification. We just ignore (i.e. return *id*) the positions where the *cut* ignored the literal (written '-') or it mentions the very same name as in the actual action. For the other positions, we just map the variable from the *cut* to the literal occurring in the actual action.

It should be noticed that, while the θ substitutes according to some checking performed between the *cut* and the actual action, the θ' substitutes according to the localised states, and to access these one should describe recommendations *rec* using the names prescribed by the syntax of Table 2, in the $v \in \mathbf{Lev}$ meta-variable.

Finally, θ and θ' are used to substitute in *cond* and in *rec* to give the final result of $[\![.]\!]$. This is achieved using the usual two-valued meaning $[\![cond]\!]$, which could be adapted to a four-valued meaning $[\![rec]\!]$. We do not define formally $[\![.]\!]$, but it can easily be done inductively on the structure of the elements of the syntactic categories **Cond** and **Rec** from Table 2.

Example In the first example of Section 4.2, the path shows that Nurse Olsen gets the Private Notes about Bob, first receiving them from Doctor Hansen and then reading them by herself. Let us assume we do not actually want that Private Notes from any patient can be obtained by any Nurse. Then, we could prevent Nurses from reading Private Notes directly by defining pol_{EHDB} as follows:

$$pol_{EHDB} = \left[\begin{array}{c} \mathbf{test}(\texttt{Doctor}, \#u)@ROLES \\ \underline{\text{if}} \;\; \#u :: \mathbf{read}(-, \texttt{PrivateNotes}, -)@EHDB : \\ \mathbf{true} \end{array} \right]$$

Certainly, this policy would avoid the last transition in the example path, but the Nurse can still get the information from the Doctor. To prevent this,

we could define pol_{staff} as follows:

$$pol_{staff} = \left[\begin{array}{c} \mathbf{test}(\texttt{Doctor}, \#target)@ROLES \\ \underline{\text{if}} \;\; \#u :: \mathbf{out}(-, \texttt{PrivateNotes}, -)@\#target : \\ \neg(\#target = EHDB) \end{array} \right]$$

Now, the only possible path (recall that although we give just 1 path in the example of Section 4.2, we also mention there are 2 more) is the following:

$$\begin{array}{l} NetData \;||\; NetRoles \;||\; NetHansen \;||\; NetOlsen \\ \rightarrow^{Hansen:\texttt{r(Bob, PrivateNotes, bobtext)}@EHDB} \\ NetData \;||\; NetRoles \;||\; NetOlsen \;|| \\ \quad Hansen ::^{pol_{defaultDr}} \mathbf{out}(\texttt{Bob, PrivateNotes, bobtext})@Olsen.\mathbf{0} \end{array}$$

4.4 Capturing BLP in AspectKB+

Having developed our formal framework, we shall show how the extended BLP policy of Section 2.3 can be elegantly captured. We shall also show that we can easily decide which cases of the example in Section 1.1 are secure and which are not, without losing any precision, unlike the information-flow approach.

Recall that **AspectKB+** is a process calculus and, even though in the original formulation of BLP the compliance with the policy is checked against the states, here we can just check the transitions. Then, to avoid insecure states, we will check if a transition might take us to such a state, therefore avoiding the transition. Also recall that, when describing Aspects, we are able to use the five syntactic constants from the syntactic category **Lev**, which will later be substituted by the evaluation function $[\![.]\!]$. So by basically using these features we aim to capture the BLP policy.

The First Aspects. Let us focus first on the **ss-property**, which prescribes that a subject cannot read an object that has higher security level than itself. The actions that can read information from other locations are the **read** and the **in** actions. So the Aspects that capture the **ss-property** are the following:

$$\left[\; S_s \geq O_t \; \underline{\text{if}} \;\; l_s :: \mathbf{read}(-)@l_t.P : \mathbf{true} \; \right] \tag{4}$$

$$\left[\; S_s \geq O_t \; \underline{\text{if}} \;\; l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true} \; \right] \tag{5}$$

Note that each Aspect is trapping a particular action, without caring about the parameters and with a trivial applicability condition. Whenever some of these Aspects trap an action, the recommendation will be considered, granting access only if the security level of the subject is not lower than that of the object, since the two names S_s and O_t will then be replaced by the corresponding security levels of the actual interacting locations, thanks to Tables 4 and 9.

For the **⋆-property.1**, which prescribes that a subject cannot write any object that has lower security level than the level where the subject is currently in, we have to follow a similar approach. Considering that the write actions are the **out** and the **in** (since deleting data is a form of write, because some implicit information could be communicated), the Aspects are as follows:

$$\left[\ O_t \geq C_s \ \underline{\text{if}}\ \ l_s :: \mathbf{out}(-)@l_t.P : \mathbf{true}\ \right] \tag{6}$$

$$\left[\ O_t \geq C_s \ \underline{\text{if}}\ \ l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true}\ \right] \tag{7}$$

Whenever some of these Aspects trap an action, the recommendation will grant access only if the security level of the object is not lower than the one the subject is currently logged in (note the use of C_s instead of S_s).

The ⋆-property.2. Now let us consider the **⋆-property.2**, which was basically the one that initiated the proposal made in this paper, due to the difficulty of capturing it precisely in a distributed setting. Note that the semantics of **AspectKB+** will keep track of the least upper bound of the security levels of the objects read by a particular subject location, because it updates it whenever the subject reads something that is not lower than the current value. A similar observation can be made for the object locations.

Let us consider a subject location, which might have read some high information as long as its security level allows it (otherwise either Aspect (4) or (5) would have denied it). Any subsequent write to a low location must be denied, and in principle either Aspect (6) or (7) might decide this, unless the subject is logged into the system with a low security level. In any case, using the localised state that we have in the subject location, and making use of the H_s syntactic constant provided by the syntax for expressing Aspects, we define the following Aspects:

$$\left[\ O_t \geq H_s \ \underline{\text{if}}\ \ l_s :: \mathbf{out}(-)@l_t.P : \mathbf{true}\ \right] \tag{8}$$

$$\left[\ O_t \geq H_s \ \underline{\text{if}}\ \ l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true}\ \right] \tag{9}$$

These can be understood in a similar way to Aspects (6) and (7), with the difference being that they are considering the localised state of the subject location, instead of the level at which the subject is logged into the system.

Analogous considerations can be made for an object location, and we can define the following Aspects to finish capturing the whole BLP policy:

$$\left[\ S_s \geq H_t \ \underline{\text{if}}\ \ l_s :: \mathbf{read}(-)@l_t.P : \mathbf{true}\ \right] \tag{10}$$

$$\left[\ S_s \geq H_t \ \underline{\text{if}}\ \ l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true}\ \right] \tag{11}$$

Combining the Aspects. After defining these eight Aspects, the idea is to combine and attach them to every location, so every time an interaction is to take place, the semantics will consider all the Aspects before allowing the interaction to happen.

Since the BLP model says that a state is secure if *both* properties are satisfied, then we need to make sure that none of the Aspects representing the properties detects a possible insecure interaction, as that would mean that at least *some* of the properties are not satisfied. For capturing this situation, the Belnap operator that must be used to combine the Aspects while attaching them to the locations is again $\oplus$.

Now we are ready to state our first Proposition:

Proposition 1 *If a distributed system could become insecure in the sense of Section 2.3 after performing an interaction, then some of the Aspects from (4) to (11) would deny the interaction.*

Proof: It should be clear that if a system could become insecure in the sense of the **ss-property** (the clearance of the subject trying to read some data is lower than the security level of the data itself), then either Aspect (4) or (5) would deny the interaction. Similarly, in the sense of the **$\star$-property.1** then either Aspect (6) or (7) would do so.

We shall then focus on the **$\star$-property.2**. If a system could become insecure in the sense of Equation (2) when the interaction is performed, then

we know that the following predicate, which is of course the complement of the one expressing security in Equation (2), would become **tt**:

$$\begin{aligned}\exists(s,o,a)\in B: \quad & (\ (a=\texttt{read}\wedge(\neg f_H(s)\geq f_O(o)\\ & \qquad\vee\neg f_H(s)\geq f_H(o)\\ & \qquad\vee\neg f_S(s)\geq f_H(o)))\ \vee\\ & (a=\texttt{write}\wedge(\neg f_H(o)\geq f_C(s)\\ & \qquad\vee\neg f_H(o)\geq f_H(s)\\ & \qquad\vee\neg f_O(o)\geq f_H(s)))\ )\end{aligned}$$

For the first disjunct, assuming the action is a **read** (resp. **in** in our case), then the first (resp. second) rule in the Semantics of Table 4 would take care of making both $f_H(s) \geq f_O(o)$ and $f_H(s) \geq f_H(o)$ satisfied, by updating the historic component on the subject location. Therefore, $\neg f_S(s) \geq f_H(o)$ should become satisfied in order to satisfy this disjunct. Since the historic component cannot be changed for the object location according to the action we are assuming (**read**/**in**), the value of $f_H(o)$ should be the same just before the interaction, and therefore Aspect (10) (resp. (11)) would avoid the interaction. This shows the first disjunct cannot become **tt**.

For the second disjunct, the case of **out** action follows the same reasoning and then Aspect (8) would avoid the interaction, preventing the second disjunct from becoming **tt**. There is a small extra argument in the case of **in** (which in our case it is also a write operation). Indeed, in such case the historic component *can* indeed be changed. Therefore, we may think that $\neg f_O(o) \geq f_H(s)$ could be satisfied if the interaction actually takes place. However, on one side, if it is already satisfied before the interaction, then Aspect (9) would deny the interaction. On the other side, if $f_O(o) \geq f_H(s)$ before the interaction, then the interaction could take place but, after applying the least upper bound done by rule $[Rule - in]$, $f_H(s)$ would become *equal* to $f_O(o)$ (but never greater than). This concludes the proof. □

For the converse proposition, we need to make an extra observation, discussed in the following Subsection.

4.4.1 Initialising the Historic Value

The Aspects just defined will check, among other values, the historic component γ^H attached to each location, and that value will be kept updated by the Semantics. However, initially, one must give a particular value for the component. The chosen value will not affect the correctness of the Aspects detecting insecure interactions, but in order to fulfil our requirement of not

losing any precision while doing so (unlike the information-flow approach) the value should be $\perp \in L$[14]. This follows the suggestion of Equation (1) and the observation just after it. Now we are ready to state the converse of Proposition 1:

Proposition 2 *If some of the Aspects from (4) to (11) deny an interaction, then the hypothetical resulting global state, in the event the interaction was actually allowed, would be insecure in the sense of Section 2.3.*

Sketch of the proof: Let us call the target location of the interaction $t*$ and the source $s*$. Assume Aspect (10) denies the interaction (for the other, the reasoning is similar). Then we know that $\neg f_S(s*) \geq f_H(t*)$ is satisfied. But since $f_H(t*)$ equals $\perp \in L$ at the beginning, this means that $f_H(t*)$ was increased by the Semantics due to some past action. If the Aspect is not present and so the interaction is allowed, then there will be a tuple $(s*, t*, a)$ where $a =$ `read` and where $f_S(s*) < f_H(t*)$, not satisfying Equation (2). Moreover, any possible past action that increased $f_H(t*)$ must have been an **out** action (only rule $[Rule - out]$ can do it). This means that in the target location $t*$ there must be some data (written with such **out** action) with security level greater than or equal to $f_C(s)$ and to $f_H(s)$ (for any subject s that has written to $t*$). Therefore, allowing a subject with clearance equal to $f_S(s*)$ to read from that location is clearly insecure. This concludes this sketch of proof.

We can now easily verify that the three examples of Figure 1 are precisely captured. It is particularly important to take into account what could happen after the process in location E writes to location D (Figure 1c). For the process in D to be actually influenced by this, it must explicitly read the data, since the semantics of **AspectKB+** will put it in another "virtual" location, with a higher historic component. So if the process is influenced, then at t3 the Aspects (actually Aspect (8)) will prevent the write to C, otherwise the write will be allowed.

4.5 A Small Example

Let us now consider a very small example to show how to combine looking to the future and to the past. Assume an airline has a database containing

[14]We mention in Section 2.3 that the initial value must be lower than or equal to the fixed security level, but that was just for the correctness of the BLP extension. In Section 4.4.1 we are going one step further: looking for precision.

information about the passengers. The historic component of the database location is initialised to $\bot \in L$ so any process could read from it, but after some data is written, only some processes could do so, according to the security level of the data written. The Aspect that prescribes this is:

$$\left[\begin{array}{c} clearance_u \geq history_{AirlineDB} \\ \underline{\text{if}}\ \ u :: \mathbf{read}(\texttt{pass}, -)@AirlineDB.P : \mathbf{true} \end{array}\right] \tag{12}$$

As can be seen, (12) is a special case of Aspect (10), but it is written like this to emphasise the example.

One of the process locations that will not be allowed to read data from the database due to Aspect (12) is the Government, whose clearance should not be enough to satisfy the *rec* of the Aspect. Indeed, the historic component of the database should be high enough since the data written in there might be sensitive for the passengers.

However, in times of heightened security due to probable threats, the Government should be able to audit the passengers, therefore it is necessary to allow the Government to read the database. At all events, the read should be allowed as long as the Government, later, will not give the passengers' data to the Press, to keep satisfying the right to privacy of the passengers. The following Aspect prescribes this:

$$\left[\begin{array}{c} \neg(\mathbf{out}(data)@PressRelease\ \mathbf{occurs\text{-}in}\ P) \\ \underline{\text{if}}\ \ Government :: \mathbf{read}(\texttt{pass}, data)@AirlineDB.P : \\ \mathbf{test}(threatlevel, \texttt{high})@AirlineDB \end{array}\right] \tag{13}$$

This is just a little Aspect that illustrates how looking to the future is achieved[15]. In the presence of Aspect (13), the Government will be allowed to perform the read action, as long as there is a tuple $< threatlevel, \texttt{high} >$ in the Airline database (i.e. the Airline was already notified of the heightened security situation), and also as long as the Government process trying to read the data will not leak the data to the Press in the future.

However, one of the conditions is set in the *cond* of the Aspect whereas the other is in the *rec*. The reason is related to the fact that Aspect (13) is a temporary one, and the aim is to combine it with Aspect (12). Moreover, the combination should be done in a way that the Government should actually be allowed to read the database, although the pre-existing Aspect (Aspect (12)) might deny this. Therefore, the operator needed for combining the two

[15] In [34] there are many more realistic examples that look to the future, in the Electronic Health Records domain.

Aspects is the priority $>$, and then the whole security policy for the Airline database would be (13) $>$ (12)[16].

With this combination, if the process location is the Government and the heightened security situation was declared, then Aspect (13) will be considered. Otherwise, either the action will not be trapped by the Aspect (if the process location is not the Government) or the condition *cond* will be **ff** (if the threat level is not high), resulting in both cases a $\perp \in$ **Four** for Aspect (13), considering then the Aspect (12).

This example, even though it is very small, clearly shows three features of our framework:

- The use of Aspects for security allows us to temporarily modify a distributed system without having to dig into the business logic of the processes.
- The use of the four-valued Belnap Logic allows us to easily combine policies, providing flexibility for the Aspect-oriented framework.
- The combination of looking to the past and to the future provides even more flexibility, giving the power to express exactly what is intended, to satisfy precisely some properties.

While the first two features where already present in the **AspectKB** framework (and in particular the first one is widely used in the Aspect-orientation community), the third one is a very powerful add-on provided by the new **AspectKB+** framework.

5 Validating the Framework

Having defined the language for describing networks and localised security policies over them, we shall proceed to devise a technique for analysing the networks actually described using this language.

What we expect to have is a Logic for expressing the desired global security property of our network, and a way to check if the property is actually met by the network, considering the existing localised policies that we have attached. We approach the problem by defining a variant of the temporal logic ACTL [28], referred in this paper as ACTLv ('v' for variant).

[16]Note that, by using this policy with the priority, the Aspect (13) could even remain there, instead of just being a temporary one, since it will be ignored in most of the cases, as long as the tuple $< threatlevel, \texttt{high} >$ is removed after the situation is normalised.

$Obl \in$ **Obligations**	Obl	::=	$AG_{\{labs\}}Pred$
$labs \in$ **Lab**	$labs$	::=	$\ell(w_s) : \mathbf{c}(\overrightarrow{\ell^t}) \,@\, \ell \ (w_t)$
$\mathbf{c} \in$ **Cap**	$\mathbf{c}$	::=	$\mathbf{o} \mid \mathbf{i} \mid \mathbf{r}$
$Pred \in$ **Predicates**	$Pred$	::=	$\mathbf{true} \mid \mathbf{false} \mid \neg Pred \mid Pred \vee Pred$ $\mid Pred \wedge Pred \mid \forall x : Pred \mid \exists x : Pred \mid bp$
$bp \in$ **BasicPredicates**	bp	::=	$\ell_a = \ell_b \mid \mathbf{test}(\overrightarrow{\ell_a})@\ell_b \mid \mathbf{test}'(\overrightarrow{\ell_a})@\ell_b \mid \Gamma_1 \geq \Gamma_2$
$\Gamma \in L$	Γ	::=	$x \mid \gamma$

Table 10: ACTLv Syntax – How to express obligations.

5.1 Defining the Logic

We expect to describe useful desired global security properties. In [32], it is shown that the properties that can be enforced at runtime by access control methods are *safety* [3] properties, and these are related with the G modality. In a process calculus as the one we are dealing with, the interactions among locations are those that need to be monitored and controlled. In fact, it is when the interactions happen that some information may go from one location to another. These issues lead us to define an action-based temporal logic, focusing on the AG modality. We will then check the possible threats arising by every interaction, assuring the stated reached after the interaction is secure.

Syntax The formal syntax of our Logic is given in Table 10. We shall express an obligation (something we want the network to satisfy) as an AG formula, meaning we want the property to be satisfied always, and in all possible paths the system might run into. Some set of transitions (lab) are to be given as subscript to the formula, and a state predicate ($Pred$) as the main checkable entity. This latter can be a combination of smaller state predicates or the simplest ones (bp) comparing two values or testing the occurrence of some value in some location. For this last issue, the state of the tested location may be the one just before the transition (if no prime symbol is added) or the one just after the transition (if the prime symbol $-'-$ is added). Finally, we could also do some comparison between any pair of security level values matched in the transition, or constant values.

$$
\begin{array}{l}
N_0 \models_{Obl} AG_{\{labs\}} Pred \\
\text{iff} \\
\nexists \text{ path } N_0 \rightarrow^* N_i \rightarrow^{l_s(w_s):\mathbf{c}(\overrightarrow{l})@l_t(w_t)} N_{i+1} : \\
\quad (\exists \theta : \quad check'(extract'(labs\,\theta), extract'(l_s(w_s) : \mathbf{c}(\overrightarrow{l})@l_t(w_t))) \\
\qquad \wedge \quad (N_i, N_{i+1}) \models^{\theta}_{Pr} \neg Pred\,)
\end{array}
$$

Table 11: ACTLv Semantics - Satisfaction relation $\models_{Obl}$.

Semantics The formal Semantics of the Logic are divided into three satisfaction relations, one for each syntactic category (**Obligations**, **Predicates** and **BasicPredicates**) defined in Table 10.

The first satisfaction relation gives semantics for the obligation formula and it is given in Table 11. It says that for an obligation to be satisfied there must not be any path that reaches some network N_i after an arbitrary number of steps (the $\rightarrow^*$), and in which the next step meets the following condition: the step can be matched[17] by some substitution done in the *labs* of the obligation but the two involved networks do not satisfy the $Pred$ of the obligation. This intuitively means that for every reachable transition relevant to the *labs* of the obligation, the two networks related by the transition should comply with the expected predicate $Pred$.

The satisfaction relation $\models_{Pr}$ is defined in Table 12. The rules should be straightforwardly understood, the only detail that will be explained is in the three last rules. For the last one, if the predicate is a basic one, then the satisfaction relation $\models_{bp}$ is used. For the previous two, we have to make an extra substitution in the predicate for evaluating it, and the values that we might take are all those that occur in the involved nets, and for that purpose the auxiliary function Loc and Vec are defined in the same Table.

The satisfaction relation for basic predicates $\models_{bp}$ is given in Table 13. The rules are straightforward, checking the equality in the first case, interpreting the **test** in the next two (distinguishing whether the **test** aims to check the net just before or just after the transition), and checking the security levels in the last one. In the four cases, the substitution is actually

[17] The use of prime symbol -′- in the $check'$ and $extract'$ functions, denotes that they are actually not the same functions used in Section 4, but a slight modification of them. Moreover, in Section 4 they operate over elements of **Cut**, whereas in Section 5 they operate over elements of **Lab**. We omit their formal definition.

$$
\begin{array}{llll}
(N_1,N_2) & \models^{\theta}_{Pr} \ \mathbf{true} & \text{iff} & \mathbf{tt} \\
(N_1,N_2) & \models^{\theta}_{Pr} \ \mathbf{false} & \text{iff} & \mathbf{ff} \\
(N_1,N_2) & \models^{\theta}_{Pr} \ \neg Pred & \text{iff} & (N_1,N_2) \not\models^{\theta}_{Pr} Pred \\
(N_1,N_2) & \models^{\theta}_{Pr} \ Pred_1 \vee Pred_2 & \text{iff} & (N_1,N_2) \models^{\theta}_{Pr} Pred_1 \vee (N_1,N_2) \models^{\theta}_{Pr} Pred_1 \\
(N_1,N_2) & \models^{\theta}_{Pr} \ Pred_1 \wedge Pred_2 & \text{iff} & (N_1,N_2) \models^{\theta}_{Pr} Pred_1 \wedge (N_1,N_2) \models^{\theta}_{Pr} Pred_1 \\
(N_1,N_2) & \models^{\theta}_{Pr} \ \forall x : Pred & \text{iff} & \forall l \in Loc(N_1) \cup Loc(N_2) : (N_1,N_2) \models^{\theta[l/x]}_{Pr} Pred \\
(N_1,N_2) & \models^{\theta}_{Pr} \ \exists x : Pred & \text{iff} & \exists l \in Loc(N_1) \cup Loc(N_2) : (N_1,N_2) \models^{\theta[l/x]}_{Pr} Pred \\
(N_1,N_2) & \models^{\theta}_{Pr} \ bp & \text{iff} & (N_1,N_2) \models^{\theta}_{bp} bp
\end{array}
$$

$$
\begin{array}{rcl}
Loc(l_1 :: \langle \overrightarrow{l_2} \rangle) & = & \{l_1\} \cup Vec(\overrightarrow{l_2}) \\
Loc(l_s :: a(\overrightarrow{l_d})@l_t.P) & = & \{l_s, l_t\} \cup Vec(\overrightarrow{l_d}) \cup Loc(P) \\
Loc(N_1 \mid\mid N_2) & = & Loc(N_1) \cup Loc(N_2)
\end{array}
$$

$$
\begin{array}{rcl}
Vec(\alpha, \overrightarrow{\alpha'}) & = & \{\alpha\} \cup Vec(\overrightarrow{\alpha'}) \\
Vec(\epsilon) & = & \emptyset
\end{array}
$$

Table 12: ACTLv Semantics – Satisfaction relation $\models_{Pr}$ and auxiliary functions Loc and Vec.

performed while checking the corresponding condition.

The way the **test** is interpreted depends on the structure of the net, and the structural inductive definition of it is given in Table 14.

Example Property: the BLP policy If the BLP policies are those that aim to be validated, one can code them using this Logic. For instance, and considering the extension from Section 2.3, the **⋆-property.2** could be partially (just the first conjunct of Equation (2)) expressed by:

$$AG_{\{l_s(<<s,-,h_1,->,->):\mathbf{r}(-)@l_t(<-,-,h_2,o>,->)\}} h_1 \geq o \wedge s \geq h_2 \wedge h_1 \geq h_2$$

We ignore the security levels, and also the policies, that are not relevant for the predicate we are defining. Given an **AspectKB+** network, the Semantics of Table 4 induces an LTS, over which the Semantics of Table 11 can be interpreted. While doing so, the security levels not ignored in the *labs* part of our property will be matched (using the function *check′*) to actual security levels whenever a transition from the LTS is relevant to our *labs*. Then, the three comparisons could be actually dealt with using the other satisfaction relations from Tables 12 and 13.

$$
\begin{array}{llll}
(N_1, N_2) & \models^{\theta}_{bp} \ \ell_a = \ell_b & \text{iff} & (\ell_a\theta) = (\ell_b\theta) \\
(N_1, N_2) & \models^{\theta}_{bp} \ \mathbf{test}(\ell_a)@\ell_b & \text{iff} & [\![\mathbf{test}(\ell_a\theta)@(\ell_b\theta), N_1]\!] \\
(N_1, N_2) & \models^{\theta}_{bp} \ \mathbf{test}'(\ell_a)@\ell_b & \text{iff} & [\![\mathbf{test}(\ell_a\theta)@(\ell_b\theta), N_2]\!] \\
(N_1, N_2) & \models^{\theta}_{bp} \ \Gamma_1 \geq \Gamma_2 & \text{iff} & (\Gamma_1\theta) \geq (\Gamma_1\theta)
\end{array}
$$

Table 13: ACTLv Semantics – Satisfaction relation $\models_{bp}$.

$$
\begin{array}{lcl}
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, \quad N_1 \mid\mid N_2 \quad]\!] & = & [\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_1]\!] \vee [\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_2]\!] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, \quad l ::^{pol} P \quad]\!] & = & \mathbf{ff} \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l_3 ::^{pol} \langle\, \overrightarrow{l_4} \,\rangle]\!] & = & (l_2 = l_3 \wedge \overrightarrow{l_1} = \overrightarrow{l_4})
\end{array}
$$

Table 14: ACTLv Semantics – Interpretation of **test**.

Indeed, if the specific network given is one that contains the Aspects discussed in Section 4.4, then the network will satisfy the ACTLv property from this example (and some others that could encode the second conjunct of Equation (2) and also the **ss-property** and the $\star$**-property.1**).

Structurally Congruent Nets Produce Same Results One expected property of the Semantics defined in the current Section is that if it is given two different **AspectKB+** networks, but which are actually structurally congruent, then the result should be the same. Indeed, otherwise it would mean that the result depends on how the network is described and not on which components it has, which in the end are the ones that make the network run.

Certainly, it could be proven using structural induction that:

$$\frac{N_1 \equiv N_2}{N_1 \models_{Obl} Obl \iff N_2 \models_{Obl} Obl}$$

We omit such proof here due to lack of space, and because it does not actually help in what we want to do next.

5.2 Model Checking without Exploring the Entire State Space

Given an **AspectKB+** network, the LTS induced by the semantics might be infinite. Certainly, the language is Turing-complete [27]. Thus, performing

a model checking over the entire language is undecidable in general.

Let us concentrate on a finite fragment of **AspectKB+**, which can easily be obtained by ruling out every network that contains the replication operator $*$. With this, we could indeed perform a usual model checking over the induced LTS, but here we will show how we could over-approximate the model checking to make it even faster: statically over the original network.

What we do, is to take each an every action of the network once, and match it (function $check'$) with the *labs* of the expected global property. If the matching succeeds, then the action is relevant for the global property, otherwise it is trivially certified as secure.

For the relevant actions, we know which are the security policies that will govern each of them during the execution of the system. Therefore, we need to check whether the 4-valued combination of the policies' recommendations implies the predicate of the global property. If this is the case, the action can be safely certified as secure. If not, the action might be insecure, but it might also be due to the over-approximation, since we do not know the actual values of the action parameters at runtime, as we are taking the action from the network description and not from the LTS.

6 Conclusion

We have studied the problem of enforcing multilevel security in a distributed system as precisely as possible. An information-flow approach poses the problem of having to "guess" what processes in other locations may do, thereby losing some precision. Therefore, we have extended an existing framework to deal with a notion of localised state, which has given us the power to access information about the past performance of the system, thereby being able to capture the Bell-LaPadula policy with precision.

The resulting framework provides a way to combine policies that look to both the future and the past due to the four-valued Belnap Logic. This gives flexibility to the framework, by capturing precisely what is intended by the security policies. This also gives more power than the previous framework of [20].

Moreover, we have proposed a Logic for reasoning about the distributed systems described within the framework, and shown some properties about it. We have discussed the approach used for model checking the systems against the Logic, in order to overcome the state explosion problem. In future work, we will report the construction of a tool for performing this

model checking, and in particular develop a little more about the different steps discussed in Section 5.2.

With our approach, other history-based security policies can be encoded. For instance, the Chinese Wall [8] security policy would only need a careful design of the lattice L of security levels and of the initial level annotations of the involved locations. Indeed, assume there are 2 competitor companies C1 and C2, and the policy says that a reader R is allowed to read information from at most one of them. We can set to γ_{C1} and γ_{C2} the security level (γ_O) of locations keeping information about C1 and C2 respectively, and to γ_0 the clearance (γ_S) of R. Assuming that $\gamma_0 \geq \gamma_{C1}$ and $\gamma_0 \geq \gamma_{C2}$, and that γ_{C1} and γ_{C2} are incomparable, the policy is satisfied if the following Aspect is present:

$$\left[\ O_t \geq H_s \ \underline{\text{if}}\ \ l_s :: \mathbf{read}(-)@l_t.P : \mathbf{true}\ \right] \tag{14}$$

Certainly, as soon as R reads from one company, say C1, the historical component will be increased by the Semantics. Then, if R tries to read from the other company, say C2, Aspect (14) will deny the interaction.

Acknowledgements

This work was partially funded by the Danish Strategic Research Council (project 2106-06-0028) "Aspects of Security for Citizens" and partially by the EU Integrated Project SENSORIA (contract 016004). We would like to thank Alan Mycroft for his comments on an early version of this paper. Finally, we really appreciated the comments from all the reviewers of the (shorter) workshop-version of this paper and of the preliminary version of this (extended) journal paper.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst.*, 15:706–734, September 1993.

[2] M. Abadi and C. Fournet. Access control based on execution history. In *NDSS*, 2003.

[3] B. Alpern and F. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1986.

[4] O. Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[5] L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *Foundations of Computer Security*, Copenhagen, Denmark, July 2002.

[6] D. E. Bell and L. J. LaPadula. Secure computer systems: mathematical foundations. Technical report, MITRE Corp., 1973.

[7] N. D. Belnap. How a computer should think. In *Contemporary Aspects of Philosophy*, pages 30–56. Oriel Press, 1977.

[8] D. Brewer and M. Nash. The chinese wall security policy. *Security and Privacy, IEEE Symposium on*, 0:206, 1989.

[9] G. Bruns, D. Dantas, and M. Huth. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, FMSE '07, pages 12–21, New York, NY, USA, 2007. ACM.

[10] G. Bruns and M. Huth. Access-control policies via Belnap logic: Effective and efficient composition and analysis. In *CSF08*, pages 163–176. IEEE Computer Society, 2008.

[11] K. M. Chandy. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3:63–75, 1985.

[12] M. J. Fischer, N. D. Griffeth, and N. A. Lynch. Global states of a distributed system. *IEEE Transactions on Software Engineering*, 8:198–202, 1982.

[13] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *JOURNAL OF COMPUTER SECURITY*, 3:5–33, 1994.

[14] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *ICALP'00*, pages 354–372, 2000.

[15] C. Fournet and A. Gordon. Stack inspection: theory and variants. In *POPL*, pages 307–318, 2002.

[16] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):96–107, 1992.

[17] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1982.

[18] D. Gollmann. *Computer security*. Wiley, 1999.

[19] K. W. Hamlen and M. Jones. Aspect-oriented in-lined reference monitors. In *PLAS '08: Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*, pages 11–20, New York, NY, USA, 2008. ACM.

[20] C. Hankin, F. Nielson, and H. Riis Nielson. Advice from Belnap policies. In *CSF09*, pages 234–247. IEEE Computer Society, 2009.

[21] C. Hankin, F. Nielson, H. Riis Nielson, and F. Yang. Advice for coordination. In *COORDINATION08, LNCS*, volume 5052, pages 153–168. Springer, 2008.

[22] A. M. Hernandez and F. Nielson. History-sensitive versus future-sensitive approaches to security in distributed systems. In *ICE2010 - 3rd Interaction and Concurrency Experience - EPTCS*, volume 38, pages 29–43, 2010.

[23] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP97, LNCS*, volume 1241, pages 220–242. Springer, 1997.

[24] B. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8:18–24, January 1974.

[25] J. McCune, S. Berger, R. Caceres, T. Jaeger, and R. Sailer. Shamon: A system for distributed mandatory access control. *ACSAC*, 2006.

[26] R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: A kernel language for agents interaction and mobility. *IEEE Trans. on Soft. Engineering*, 24(5):315–330, 1998.

[27] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of klaim-based calculi. *Theor. Comput. Sci.*, 356:387–421, May 2006.

[28] R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. In *Proceedings of the LITP spring school on theoretical computer science on Semantics of systems of concurrent processes*, pages 407–419, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[29] A. Sabelfeld and H. Mantel. Static confidentiality enforcement for distributed programs. In Manuel Hermenegildo and Germán Puebla, editors, *Static Analysis*, volume 2477 of *Lecture Notes in Computer Science*, pages 376–394. Springer Berlin / Heidelberg, 2002.

[30] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.

[31] R. S. Sandhu. Lattice-based access control models. *Computer*, 26:9–19, 1993.

[32] F. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[33] T. Woo and S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.

[34] F. Yang, C. Hankin, F. Nielson, and H. Riis Nielson. Aspects-oriented access control of tuple spaces. *Manuscript submitted to a journal*, 2010.