Analysis and Run-time Verification of Dynamic Security Policies

H. Janicke, F. Siewe, K. Jones, A. Cau and H. Zedan

{heljanic, fsiewe, kij, acau, hzedan}@dmu.ac.uk
Software Technology Research Laboratory,
Gateway House, De Montfort University,
Leicester LE1 9BH

Abstract. Ensuring the confidentiality, integrity and availability of information is the key issue in the battle for information superiority and thus is a decisive factor in modern warfare. Security policies and security mechanisms govern the access to information and other resources. Their correct specification, i.e. denial of potentially dangerous access and adherence to all established need-to-know requirements, is critical. In this paper we present a security model that allows to express dynamic access control policies that can change on time or events. A simple agent system, simulating a platoon, is used to show the need and the advantages of our policy model. The paper finally presents how existing tool-support can be used for the analysis and verification of policies.

1 Introduction

Fast and reliable access to information is becoming one of the major factors that decide on the success of a military operation. Modern technologies, such as airborne sensors and satellite imaging, provide more detailed and accurate data about the physical domain than ever before. The amount of information helps to lower uncertainty whilst new technologies to communicate information, help to develop shared situational awareness.

The sheer load of information also has its drawbacks. Commanders that need to make decisions quickly might not have the capability to analyse and comprehend the provided information in time. Decision making processes are therefore supported by systems that are capable of analysing, filtering, combining and presenting information that is relevant to the scenario. Such a system can be seen as a Multi Agent System in which Software Agents represent the information sources and processors that assist in human decision making [1,2,3].

The agent system that is providing, processing and communicating the information, together with the information itself, forms the information domain. It is information that is becoming increasingly important with the rise of Network Centric Warfare, and concerns about its availability, confidentiality and integrity are predominant factors that decide on the success of military operations [4]. These security requirements are traditionally expressed in security policies. Security policies describe properties that the underlying system must implement to be secure. This is usually ensured by adequate security mechanisms, that enforce the policy on the system. Security policies are an invaluable asset to any organisation, especially, when they are based on a sound model, that can be used for the analysis and proof of properties.

Security policies in general deal with all classes of security requirements. We restrict ourself here to those concerned with access control. Most access control models that are available today [5,6], are of a relatively static nature and make it difficult to express access control requirements that are dependent on time or the occurance of events. These temporal aspects of access control are becoming more important the more flexible ways of communicating information become. Especially in the military domain the value of tactical information, and therefore its protection requirements, will be highly dependent on time (e.g. time to mission start) and events (e.g. adversary action).

Other work [7,8] has recognised the need for more expressive security policies, to capture the temporal dimension of access control. Although widely recognised, these models lack compositionality. By compositionality we mean that the overall security policy can be composed out of smaller policies that capture specific requirements and that can be verified individually. The advantage of the access control model that is used in our work is that it does not only allow to express parallel, but also sequential composition, which allows to express changes in the policy dependent on time and events. The security model has a sound foundation in Interval Temporal Logic which has been successfully used for functional and temporal system specifications [9] and is now extended to express security properties [10].

The security model and the tool-support for the analysis is part of SANTA development framework, that is concerned with the development of secure Multi Agent Systems. In this framework agents are controlled by security policies, that express security requirements such as authorisation, delegation and obligation. SANTA is unifying, i.e. it allows to express functional, temporal and security requirements within the same formal framework.

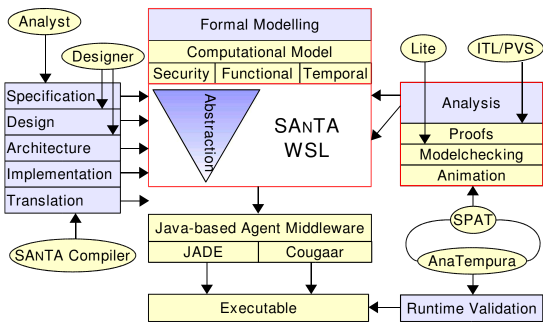


Fig. 1. The SANTA Framework

Beside the sound model, SANTA comprises linguistic support that allows the top-down development of Multi Agent Systems together with their security requirements. The importance of addressing security in the beginning and throughout the development has been widely recognised [11], but there is still a lack of methodology and tool-support. The SANTA framework, depicted in Fig. 1, tries to rectify this situation and draws the focus on security requirements and their interplay with functional and temporal aspects. Starting from a formal specification, the abstract design of the agent system is written in the wide-spectrum language SANTA-WSL, that allows to express abstract specification (as Interval Temporal Logic formulae) and concrete implementation within the same language. SANTA-WSL is close to the popular Java programming language, and contains additional constructs for agent and policy specification. These additions make it easier for an agent-system developer to implement application level security requirements. The SANTA-WSL translator is then used to translate the SANTA-WSL program into an appropriate agent middle-ware. Programs in SANTA-WSL will have a formal semantics and can be analysed using a variety of tools that comprise the SANTA toolkit. This paper will not discuss the development approach itself, but show how access-control policies can be composed to cater for dynamic aspects of security requirements.

Dynamic access control policies are more expressive, but also more difficult to comprehend and analyse. In this paper we will present the prototype of the Security Policy Analysis Tool (SPAT) using a small case study, that illustrates some of the temporal aspects of access-control. The tool will be used to animate the security policy, to show how access control decisions change over time and by events. It allows the analysis of information flow and can provide information on which policy rule is responsible for a concrete access control decision.

The rest of the paper is organised as follows. Section 2 provides a short informal introduction to the underlying logic and the security model. Section 3 then describes a simplified scenario and shows the formalisation of security requirements. Section 4 describes the tool-support for the visualisation and verification of security policies. The final section then concludes and outlines future work.

2 A Dynamic Security Model

This section will provide an overview of how access control requirements can be expressed in our model using small motivating examples. Due to space limitations we do not provide the formal semantics of the model in this paper, but refer the interested reader to [10]. The security model is based on Interval Temporal Logic (ITL), which provides the sound foundation that is necessary in the development of critical systems. We first provide a short informal introduction to ITL, and go then on to introduce our security policies in a small scenario.

2.1 Interval Temporal Logic

Interval Temporal Logic (ITL) is a flexible notation for both propositional and first order reasoning about periods of time found in descriptions of hardware and software systems. It can handle both sequential and parallel composition unlike most temporal logics [12] since assumption/commitment paradigm and a set of compositional guidelines [13] are applied in ITL. There is a very powerful and practical compositional proof system for ITL [12]. That is, much of the proof of a system specified in ITL can be decomposed into proofs of its parts. It offers powerful and extensible specification and proof techniques for reasoning about properties involving safety, liveness and timeliness.

Syntax and Semantics The key notion of ITL is an *interval*. An interval σ is considered to be a (in)finite sequence of states $\sigma_0,\ \sigma_1 \ldots$, where a state σ_i is a mapping from the set of variables Var to the set of values Val. The length $|\sigma|$ of an interval $\sigma_0 \ldots \sigma_n$ is equal to n (one less than the number of states in the interval, i.e., a one state interval has length 0). The syntax of ITL is defined below.

Expressions

$$e ::= \mu \mid a \mid A \mid g(exp_1, \ldots, exp_n) \mid \imath a\colon f$$

Formulae

$$f ::= p(e_1, \ldots, e_n) \mid \neg f \mid f_1 \wedge f_2 \mid \ \forall v \bullet f \mid \mathsf{skip} \mid f_1 \,;\, f_2 \mid f^*$$

Where μ is an integer value, a is a static variable (doesn't change within an interval), A is a state variable (can change within an interval), v a static or state variable, g is a function symbol and p is a predicate symbol.

The informal semantics of the most interesting constructs are as follows:

- skip: unit interval (length 1, i.e., an interval of two states).
- $f_1 \,;\, f_2$: holds if the interval can be decomposed ("chopped") into a prefix and suffix interval, such that f_1 holds over the prefix and f_2 over the suffix, or if the interval is infinite and f_1 holds for that interval. Note the last state of the interval over which f_1 holds is shared with the interval over which f_2 holds. This is illustrated in Figure 2.
- f^*: holds if the interval is decomposable into a finite number of intervals such that for each of them f holds, or the interval is infinite and can be decomposed into an infinite number of finite intervals for which f holds. Figure 2 illustrates the chopstar operator.

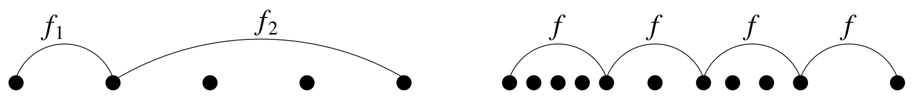


Fig. 2. Chop and Chopstar

Derived constructs Following is a list of some derived constructs which are useful for the specification of systems:

- $\mathsf{finite} \mathrel{\widehat{=}} \neg(true \,;\, false)$: finite interval, i.e., any interval of finite length.
- $\Diamond f \mathrel{\widehat{=}} \mathsf{finite} \,;\, f$: sometimes f, i.e., any interval such that f holds over a suffix of that interval. Example: $\Diamond X \neq 1 \mathrel{\widehat{=}} \mathsf{finite} \,;\, X \neq 1$: Any interval such that there exists a state in which X is not equal to 1.
- $\Box f \mathrel{\widehat{=}} \neg\Diamond\neg f$: always f, i.e., any interval such that f for all suffixes of that interval. Example: $\Box X = 1 \mathrel{\widehat{=}} \neg(\mathsf{finite} \,;\, X \neq 1)$: Any interval such that the value of X is equal to 1 in all states of that interval.
- $\mathsf{fin}\ f \mathrel{\widehat{=}} \Box(\mathsf{empty} \supset f)$: final state, i.e., any interval such that f holds in the final state of that interval.

2.2 Security Policies

Access control policies are expressed in terms of subjects, objects and actions. Subjects represent active entities, such as users and processes, that can be authenticated within the system. We denote the set of all subjects by S. The system state is represented by objects. Objects can only be modified by the execution of actions on request of authenticated subjects. We denote the set of all objects by O, the set of all actions by A. The access control policy determines whether a subject is allowed to perform an action on an object, or not.

In the context of a Multi Agent System, each agent is seen as both, subject and object. As a subject, it is a uniquely identifiable process that acts on behalf of another agent or user. As an object, it encapsulates its state. In our case contains information about the physical domain, such as images, positions or other tactical information, that requires protection.

Traditionally access control policies are defined in terms of rules that capture access control requirements [14]. The general form of a rule is:

$$\mathit{premise} \longrightarrow \mathit{consequence}$$

The premise of a rule determines when the rule *fires* and the consequence of the rule determines the outcome of the rule, for example an access control decision. We follow this approach, but allow the premise of a rule to express a behaviour rather than a predicate. The intuition is that an authorisation can be dependent on the history of execution rather than only the currently observable state. This allows to express history dependent authorisations such as the Chinese Wall Policy [15]. The following example shows such a rule:

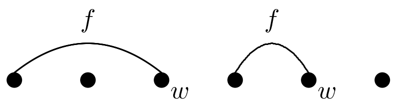
$$\begin{pmatrix} \forall s \in S, o \in O, a \in A \cdot \\ \Diamond do(s,o,a) \wedge clientinfo(c,o) \wedge \\ sepconcern(c,c') \wedge clientinfo(c',o') \end{pmatrix} \mapsto autho^{-}(s,o',a) \qquad (1)$$

Where the predicates have the following meaning:

- $do(s,o,a)$: Subject s performs action a on object o.
- $clientinfo(c,o)$: Object o belongs to client c.
- $sepconcern(c,c')$: Client c and client c' are in a separation of concern relationship.

The rule given in Eq. 1 then states that when a subject has at some point in time accessed information of a client, the same subject cannot (negative authorisation denoted by $autho^{-}$) access information about a client that is in a separation of concern relationship.

The informal semantics of operator $\mapsto$ (*Followed By*), that is used in the rules is: Whenever f holds for a subinterval, w holds in the last state of that subinterval. This is depicted in the figure below.



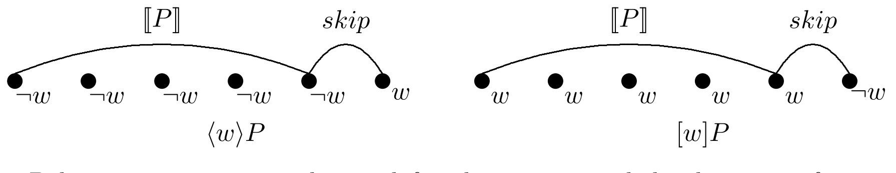
The right-hand side of a rule in the security model contains either the variable *autho*, $autho^{+}$ or $autho^{-}$. This allows to express hybrid access control policies, in which both positive authorisation ($autho^{+}$) and negative authorisation ($autho^{-}$) can be expressed. In case of conflicts, i.e. a subject has both positive and negative authorisation, a conflict resolution rule (*autho*) determines the actual access decision. Eq. 2 shows a conflict resolution rule, stating that a negative authorisation takes precedence over a positive authorisation.

$$autho^{+}(S,O,A) \wedge \neg autho^{-}(S,O,A) \mapsto autho(S,O,A) \tag{2}$$

We denote universally quantified variables by uppercase letters. Rules form the basis of our access control model. A simple policy can be seen as a set of these rules, where the intuition is that all rules apply simultaneously.

To capture the dynamics of certain security requirements and to allow the incremental development of security policies, policies can be composed using a rich set of operators. The following depicts a selection of operators with their informal semantics.

– $[\![P\,;Q]\!] \mathrel{\widehat{=}} [\![P]\!]\,;[\![Q]\!]$: Sequential composition of two policies. The system is first governed by policy P and then by policy Q.
– $[\![P \| Q]\!] \mathrel{\widehat{=}} [\![P]\!] \wedge [\![Q]\!]$: Parallel composition of two policies. The system is governed by policy P and Q at the same time.
– $[\![\langle w\rangle P]\!] \mathrel{\widehat{=}} [\![[\neg w]P]\!]$: The system is governed by policy P unless w holds. The state formula w can here indicate the happening of an event.
– $[\![[w]P]\!] \mathrel{\widehat{=}} ([\![P]\!] \wedge \Box w) \vee ((([\![P]\!] \wedge \Box w)\,;\mathsf{skip}) \wedge \mathsf{fin}\ \neg w) \vee (\mathsf{empty} \wedge \neg w)$: The system is governed by policy P as long as w holds.



Policy composition can be used for the incremental development of security policies. The advantage of this approach is that small policies are easier to comprehend and verify. The compositional operators can then be used for the integration of the overall system policy. We presented only a selection of operators that are used in the following case-study. The policy model provides a wider range of operators, for example to allow the dynamic addition/ deletion of rules or to select different policies according to the happening of an event or a time-out, whichever is first.

3 Case Study

We present a small simplified scenario, that shows the use of our dynamic policy model. We will use this scenario in the subsequent section, to illustrate the functionality of our analysis toolkit.

Scenario *A platoon is navigating an area, where long range communication is limited due to environmental conditions. The platoon consists of several small units and a command unit that carries a long distance transmitter. The communication within the platoon is enabled using short distance radio links. The quality of service of the long distance transmission is highly dependent on the environment the platoon is navigating. Dependent on the command units position there may be significant drops in the communication bandwidth or even areas where communication is not possible at all. The command unit is used to analyse and control the mission. It is constantly relaying mission related information back to the base and provides a relay service to the other members of the platoon. The access to the relay service is controlled by a policy with the following requirements.*

1. All members of the platoon are allowed to relay information.
2. If the bandwidth is dropping below 50% then units that have not been involved in combat action within the last 2 time-units are denied to relay information.
3. If the bandwidth drops below 20% only the command unit can relay tactical and strategic information.
4. If the command unit is under attack, the units that are not in its direct proximity are denied to relay messages, regardless of the available bandwidth.

In the following we will formalise the requirements individually as rules and then show how the rules can be composed to reflect the overall requirement specification. We formalise the first requirement as in Eq. 3

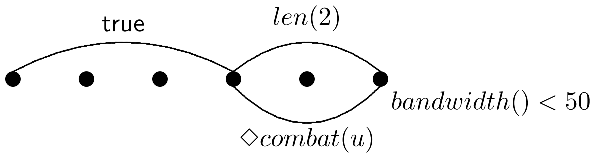
$$member(U, platoon) \wedge command(CMD, platoon) \mapsto autho^{+}(U, CMD, relay \tag{3}$$

Where *member* represents the membership relation between units and the platoon, and *command* the command unit relation. If U is a member of *platoon* and CMD is the command unit of the *platoon* it follows that U is authorised to *relay* information via the command unit CMD.

Whilst the first requirement uses only static information, such as the membership, the second requirement includes a temporal aspect. This can be formalised, as in Eq. 4.

$$\begin{pmatrix} \mathsf{fin}\,(bandwidth() < 50) \wedge member(U, platoon) \wedge \\ \neg(\mathsf{finite}\,;(\Diamond combat(U) \wedge len(2))) \wedge \\ command(CMD, platoon) \end{pmatrix} \mapsto autho^{-}(U, CMD, relay) \quad (4)$$

If the interval cannot be decomposed into a prefix interval and a suffix interval of length 2, in which sometimes $combat(U)$ holds, and if the bandwidth is in the last state of the interval below 50% then the unit is explicitly denied to relay information via the command unit. The informal semantics of this rule is depicted below.



The formalisation of requirement 3 follows the same lines as requirement 2, without the temporal aspect.

$$\begin{pmatrix} bandwidth() < 20 \\ \wedge\ member(U, platoon) \wedge \\ command(CMD, platoon) \end{pmatrix} \mapsto autho^{-}(U, CMD, relay) \quad (5)$$

Requirement 4 finally defines, that if the command unit is under attack, units that are not in its proximity are denied to relay information, regardless of the bandwidth requirements stated in requirements 2 and 3.

$$\begin{pmatrix} combat(CMD) \wedge \\ command(CMD, platoon) \wedge \\ \neg near(CMD, U) \end{pmatrix} \mapsto autho^{-}(U, CMD, relay) \quad (6)$$

The rule in Eq. 6 expresses the requirement. Additionally it states that the rule overrides the requirements 2 and 3. This can be seen as a dynamic change in the security policy, dependent on the event that the command unit is engaged in combat.

The whole policy is expressed as a hybrid policy in which denials take precedence over allowances (Eq. 2). Policy rule 3 holds independently of policy changes. This means in general all members of the platoon have the authorisation to relay information. We distinguish now between two cases:

a) The command unit is not engaged in combat.
b) The command unit is engaged in combat.

The policy for case *a)* consists of rules 4 and 5, stating that access is limited according to the available bandwidth. It is applied *unless* the command unit is engaged in combat. On this event the policy changes (sequential composition, ;) to case *b)* defined by rule 6, stating that units in the proximity can relay information. Case *b)* is applied as long as the command unit is under attack, and is then followed by case *a)*. The composed policy is given in Eq. 7.

$$\{Eq.\ 2, Eq.\ 3\} \| (\langle combat(CMD) \rangle \{Eq.\ 4, Eq.\ 5\}\ ;\ [combat(CMD)]\{Eq.\ 6\})^* \qquad (7)$$

The advantage of this approach is that access requirements that are dependent on time and events, can be expressed at a higher abstraction level, without the need to explicitly encode the conditions in the premise of the rule. This leads to rules and policies that are easier to comprehend. Using policy composition, the security administrator can then decide on the time and event relations between different policies.

The case-study demonstrates the use of dynamically changing policies, and does show how requirements to control access to resources are specified in general. The model allows to express more traditional security concepts like multi-level security and role-based access control, via the introduction of appropriate predicates. Its compositionality then allows to combine different policies and to reason about properties of the composition.

The semantic model of the security policies allows the formal analysis of the security specification and can be used to prove properties about the specification. In the following we will show how these security policies can be expressed in Tempura, an executable subset of ITL and present tool-support, that assists in the analysis of the given security policy.

4 Analysis and Run-time Verification

An important reason of choosing ITL is the availability of an executable subset of the logic, known as Tempura [16]. A formula is executable if *i.* it is deterministic, *ii.* the length of the corresponding interval is known and *iii.* the values of the variables (of the formula) are known throughout the corresponding interval. The Tempura interpreter takes a Tempura formula and constructs the corresponding sequence of states, i.e., interval. For more technical details of the interpreter, we refer the reader to [16] which is available from the ITL home-page [17]. The use of ITL, together with its subset of Tempura, offers the benefits of traditional proof methods balanced with the speed and convenience of computer-based testing through execution and simulation. The entire process can remain in one powerful logical and compositional framework.

4.1 Expressing Access Control Policies in Tempura

Executable Temporal Logics have been used for the high-level specification of Multi Agent Systems for a considerable time [18]. Advantages of Tempura are that both parallel and sequential composition is expressible, and that it can closely resemble well known programming language constructs. Tempura has been previously applied to hardware verification and the analysis of time-critical systems [9].

This allows us to model the behaviour of the agent system at a high level and shows how the security policy controls the access to system resources. An access control rule can be expressed in Tempura as follows:

```
define rule1(AuthoP) = { keep {
  forall s < noSubjects :
    forall o < noObjects :
      (member(s,platoon) and command(o,platoon)) implies AuthoP(s,o,relay)=true
}}.
```

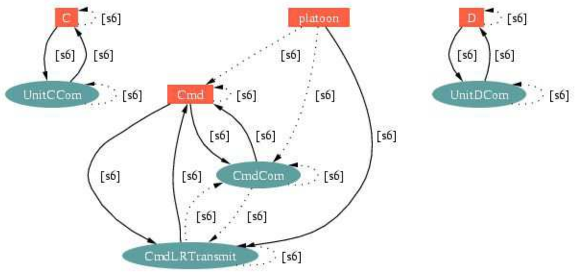
Where the predicates `member`, `command` model the relations-ships as in Eq.3. The rules can then be combined using parallel and or sequential composition. The complete policy is shown in the listing below.

```
define policy(AuthoP, AuthoN, Autho) = {
  rule1(AuthoP) and denialtakesprecedence(AuthoP, AuthoN, Autho) and
  ( (halt(combat(cmd)) and rule3(AuthoN) and rule4(AuthoN)) ;
    (halt(not combat(cmd)) and rule5(AuthoN)))
}.
```

The Tempura program representing the system simulation and the policy description is then executed by the Tempura interpreter. The code emits information about current access-control decision in each step of the execution to the graphical analysis tool.

4.2 SPAT

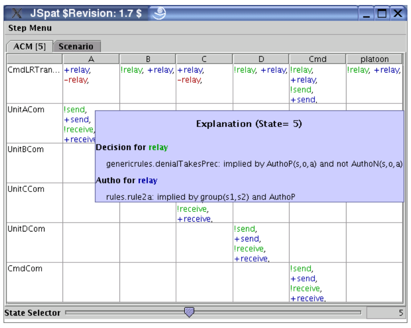
The Security Policy Analysis Tool (SPAT) is used to analyse the behaviour of dynamically changing policies. The graphical front-end can display the access control matrix for all states in the simulation, and it provides interactive filtering mechanisms that make it easier to obtain the required information. Access control information can be displayed in form of an access control matrix, in form of access control lists, or capabilities. The tool also supports the visualisation of delegation and access control decisions, which are not demonstrated in the presented scenario.



Information Flow Transitive Closure

Especially interesting is the analysis of permissible information flow. By permissible information flow we mean such flows that are allowed by the access control policy. This is a valuable aid in debugging the policy, because *a)* unwanted information flows can easily be detected and *b)* restrictions in the policy that violate any *need-to-know* requirements can easily be seen. The figure above depicts the permissible information flow in state 6 of the simulation.

The prototype, that is currently under development, can provide an explanation which rules caused an authorisation or denial. This allows to trace back the rule that lead to an unwanted authorisation and helps in the design of security policies, that match the informal requirement. The figure below depicts the scenario simulation and shows an example explanation.



	A	B	C	D	Cmd	platoon
CmdLRTran...	+relay, -relay,	!relay, +relay,	+relay, -relay,	!relay, +relay,	!relay, +relay, !send, +send,	!relay, +relay,
UnitACom	!send, +send, !receive, +receive					
UnitBCom						
UnitCCom			!receive, +receive,			
UnitDCom				!send, +send, !receive, +receive,		
CmdCom					!send, +send, !receive, +receive,	

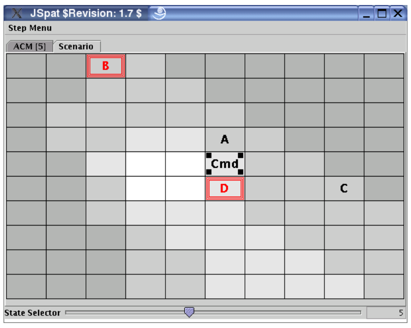
Explanation (State= 5)

Decision for relay

genericrules.denialTakesPrec: implied by AuthoP(s,o,a) and not AuthoN(s,o,a)

Autho for relay

rules.rule2a: implied by group(s1,s2) and AuthoP



The access control matrix (left picture), together with the explanation component and the graphical visualisation are generic components, that can be used for the analysis of arbitrary security policies. The scenario representation (right picture) is dependent on the scenario itself, but SPAT provides mechanisms for the development of such components and their integration.

5 Conclusion and Future Work

We illustrated the need for dynamically changing security policies using a small military scenario. We presented the security model that underlies the SANTA framework and showed how security policies can be incrementally developed. Unlike most other models, our model allows capturing temporal aspects in both, the premise of authorisation rules and through policy composition. The Security Policy Analysis Tool can then be used to animate and visualise the developed policies, to ensure that the formalisation captured the initial requirements. The tool is especially useful for the analysis of permissible information flow. It allows to write the access control policy tight enough to prohibit malicious behaviour and still ensure that all *need to know* requirements are fullfilled.

Future work will concentrate on the enhancement of the tool support for both the analysis and the linguistic support. In the analysis part we enhance the tracability of access control decisions and increase functionality to filter the visualised information. We also plan to enhance the Tempura interpreter, to allow the animation of a wider class of security policies together with the agent system specification. In the linguistic part we develop the wide-spectrum language SANTA-WSL in which both security and functional aspect can be expressed in a uniform, accessible language at all levels of abstraction.

References

1. Thomas E. Potok, A.S.L., Phillips, L., Pollock, R.: Suitability of agent technology or military command and control in the future combat system environment. In: Proceeding 8th ICCRTS, National Defence University. (2003) 1
2. F.T. Sheldon, T.P., Kavi, K.: Multi-agent system case studies in command and control, information fusion and data management. In: Journal of Informatica. Volume 28., Solvene Society Informatica (2004) pp 78–89 1
3. Bharadwaj, R.: Secure middleware for situation-aware naval c2 and combat systems. In: In Proceedings 9th International Workshop on Future Trends of Distributed Comput ing Systems (FTDCS 2003). (2003) 1
4. Alberts, D.S.: Understanding information age warfare. CCRP publication series, DoD, US (2001) 1
5. Jajodia, S., Samarati, P., Subrahmanian, V.S., Bertino, E.: A unified framework for enforcing multiple access control policies. ACM transaction on Database Systems **26** (2001) 214–260 1
6. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems **15** (1993) 1–29 1
7. Steve Barker, P.J.S.: Flexible access control specification with constraint logic programming. ACM Transactions on Information and System Security **6** (2003) 1
8. Bertino, E., Bonatti, P.A., Ferrari, E.: Trbac: A temporal role-based access control model. ACM Trans. Inf. Syst. Secur. **4** (2001) 191–233 1
9. A. Cau, C. Czarnecki, H.Z.: Designing a provably correct robot control system using a lean formal method. In: Proceedings of FTRTFT'98, LNCS 1486. (1998) pp 123–132 1, 4.1
10. Siewe, F., Cau, A., Zedan, H.: A compositional framework for access control policies enforcement. In: Proceedings of the ACM workshop on Formal Methods in Security Engineering: From Specifications to Code. (2003) 1, 2
11. Eckert, C.: Matching security to application needs. In: IFIP TC11 11TH INTERNATIONAL CONFERENCE ON INFORMATION SECURITY. (1995) 237 –254 1
12. Moszkowski, B.: Some very compositional temporal properties. In Olderog, E.R., ed.: Programming Concepts, Methods and Calculi. Volume A-56 of IFIP Transactions., IFIP, Elsevier Science B.V. (North–Holland) (1994) 307–326 2.1
13. Zedan, H., Cau, A., Zhou, S.: A calculus for evolution. In: Proc. of The Fifth International Conference on Computer Science and Informatics (CS&I'2000). (2000) 2.1
14. Woo, T.Y.C., Lam, S.S.: Authorization in distributed systems: A formal approach. In: Proceedings of the 13th IEEE Symposium on Research in security and Privacy, Oakland, California, May 4-6 (1992) 33–50 2.2
15. Brewer, D., Nash, M.: The chinese wall policy. In: IEEE Symposium on Research in Security and Privacy. (1989) 206–214 2.2
16. Moszkowski, B.: Executing Temporal Logic Programs. Cambridge University Press, England (1986) 4
17. Cau, A., Moszkowski, B., Zedan, H.: The ITL homepage. http://www.cse.dmu.ac.uk/ cau/itlhomepage/index.html (2002) 4
18. Fisher, M.: A survey of concurrent METATEM – the language and its applications. In Gabbay, D.M., Ohlbach, H.J., eds.: Temporal Logic - Proceedings of the First Intemational Conference (LNAI Volume 827), Springer-Verlag: Heidelberg, Germany (1994) 480–505 4.1