Secure Requirements Elicitation Through Triggered Message Sequence Charts

Arnab Ray, Bikram Sengupta, and Rance Cleaveland

[1] Department of Computer Science SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
arnabray@cs.sunysb.edu
[2] IBM India Research Labaratory,
Block-1, Indian Institute of Technology, Hauz Khas,
New Delhi-110016, India
bsengupt@in.ibm.com
[3] Department of Computer Science SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
rance@cs.sunysb.edu

Abstract. This paper argues for performing information-flow-based security analysis in the first phase of the software development life cycle itself ie in the requirements elicitation phase. Message Sequence Charts (MSC)s have been widely accepted as a formal scenario-based visual notation for writing down requirements. In this paper, we discuss a method for checking if a TMSC (Triggered Message Sequence Chart), a recently propsed enhancement to classical MSCs, satisifes one of the most important information flow properties namely non-interference.

1 Introduction

With our increased reliance on computer systems in all aspects of life, protecting the confidentiality of information being manipulated has become an increasingly important research problem To be confident that a system is secure with respect to confidentiality, it should be rigorously analyzed as a whole to check if it enforces good confidentiality practices. The aim of the analysis should be to demonstrate that the information controlled by a confidentiality policy cannot leak out to a location where that policy is being violated. These policies which govern the movement of information through the system are called *information flow policies*.

Information flow is traditionally checked through run-time monitoring of systems [4], or by static analysis of the source-code [2] [3]. But these are post-implementation approaches—finding spurious information flow at this stage may result in the entire system being sent back to the drawing board for possible redesign and reimplementation. Model-based approaches for information-flow analysis that check for information leaks at the design phase [7], [5] developed out of the need to isolate security bugs as early as possible in the development life-cycle with the broader aim of reducing the cost that would otherwise be incurred implementing an insecurely designed system.

What should be noted however is that models are not the earliest artifacts in the software development process. Before models are constructed by the design team, the

client or customer typically provides requirements which encapsulate her demands from the final system. Traditionally these requirements have been textually represented in a natural language like English. But in modern software engineering, requirements are expressed in precise formalisms like Message Sequence Charts (MSC) [6] which are now accepted as standard notation for systems by the International Telecommunications Union (ITU).

In practice, the software design phase consists of an iterative process with developer's designs being refined by client's requirements and vise versa. A system which is developed based on "sensitive-information leaky" requirements is bound to be insecure no matter how well designed the ultimate system is. An implementation that meets these faulty requirements will be doomed to be information-unsafe. The solution to such a problem would not be found in better implementations or in better models but in more secure requirements. Thus any software engineering solution for secure information-flow that does not analyze requirement elicitation formalisms will always be incomplete. In this paper, we endeavor to provide methods to formally analyze information flow violations on requirements expressed in terms of a recently proposed variant of MSCs called Triggered Message Sequence Chart (TMSC)s [11] which enrich scenarios with the notions of conditional and partial behavior so as to facilitate early stage requirements modeling.

The paper is organized as follows. Section 2 provides the background for the concepts introduced, Section 3 illustrates an example of information-flow analysis on TMSCs while Section 4 shows how TMSCs may be characterized as ready sets. Section 5 illustrates the formal way of doing non-interference analysis on TMSCs and Sections 6 and 7 introduce TMSC expressions and investigates whether application of TMSC operators preserve NI. The last section concludes the paper.

2 Background

A *process* is a tuple $\langle S, A, \rightarrow, s_I \rangle$ where: S is a set of *states*; A is a *action set* consisting of named-actions and the internal transition τ; $\longrightarrow \subseteq S \times A \times S$ is the *transition relation*, and $s_I \in S$ is the *start state*.
A set X of actions is a ready set of a state p in process P ie $Ready(p) = X$ if the state p offers X to the environment. In figure 1 we see an example of a process with the ready set of state 1: $Ready(1) = \{\{a, b\}\}$
Non-interference (NI) is an information flow property which states that if there are two privilege levels for all the actions of a process –HI and LO it is impossible for an observer who can observe only LO events to deduce any information about HI events. Alternatively this can be interpreted as: for any system trace (sequence of visible actions) consisting of HI(H) and LO(L) events there is a second trace consisting of the same subsequence of LO events as the first trace but with no HI events. Since both these traces are legal traces of the system, an observer (who can look at only LO actions) would not know whether the first trace occurred or the second. Hence he would obtain no information about the HI events from his observation.

If P be a process with its action set partitioned into two sets L and H (corresponding to low and high privilege actions respectively) and tr be a trace then P/tr denotes

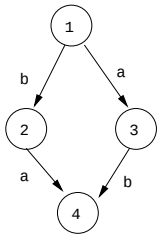

Fig. 1. Process and Ready Set

the process state after executing trace tr and $traces(P)$ denote all the traces of P. Then:
$ReadySets(P/tr)$= $\{Ready(p) \mid$ for all traces tr from s_I to $p\}$
$ReadySets_L(P)$ denotes $ReadySets(P)$ restricted to the L alphabet.

In [8] a formulation for NI is given as:
$\forall tr \in traces(P).ReadySets_L(S/tr) = ReadySets_L(S/tr \uparrow L))$

$tr \uparrow L$ projects tr down to the event set L ie purges out all the H actions.
[5] provides several characterizations for NI using a process model and ready sets of which we consider one above.

Fig. 2. Two TMSCs M_1 and M_2

Triggered Message Sequence Charts Triggered Message Sequence Charts (TMSC)s [11], like MSCs [1] describe system scenarios in terms of the atomic actions (message sends and receives and local actions) that each parallel instance may engage in. However unlike traditional MSCs each TMSC instance's action sequences are partitioned into two subsequences: a *trigger* and an *action*. A TMSC scenario stipulates that in any system execution, if the sequence of events performed by an instance constitutes the trigger, then the subsequent behavior of the instance must include the sequence of events that constitute its action. In an implementation, an instance is not *required* to display the

behavior performed by the trigger, but if it does so, its subsequent behavior is limited by the action.

Graphically we denote TMSCs as shown in Figure 2. The partitioning of the sequences of events into the trigger and action sequences is indicated by the horizontal line running through the instances of the MSC. For each instance, the sequence of events above the line constitute the trigger while that below constitutes the action.

In Figure 2 we have two TMSCs M_1 and M_2. Each TMSC has 2 instances called W and R respectively. Another novelty of TMSCs, in comparison with MSCs, is that a scenario may be partial. This is depicted by the presence/absence of a bar at the foot of an instance in a TMSC. The presence of such a bar, as in TMSC M_2, indicates that the instance has to terminate on reaching that point, whereas the absence, as in M_1, indicates that there are no constraints on the subsequent behavior of the instance. As such, the scenario is partial, and may be extended later on.

M_1 trigger consists of a message from W to R labeled by a and its action consists of a message from R to W labeled by b. M_2 has an empty trigger followed by the same exchange of messages present in M_1.

3 Information flow in TMSCs

The information-flow we shall be concerned with in this paper is *Non-interference* or NI for short. We defined non-interference in terms of processes in the previous section. Over here we lift that definition to TMSCs and use it to define the concept of information-safety on requirements.

Let us associate one of 2 security levels (HI or LO) with each message in a TMSC. The intuition behind NI is as follows. Actions having a LO security level may be observed by anyone but not HI actions. For a system to have the non-interference property, it should be impossible for a user observing LO actions to deduce anything about the occurrence or non-occurrence of HI actions. In other words, for an insecure system even though the attacker cannot directly observe HI actions, she may still be able to deduce information about HI actions from observing LO actions only. Thus if a HI action is preceded (or followed) by a LO action, it should also be possible for the HI action to occur without being preceded (or followed) by the LO action. If that be the case, then even if the LO action is observed no information is leaked about the HI action as the HI action can also occur without the LO action.

In Figure 2, let a be a HI action and b be a LO action. In M_1 , a triggers b. However, this does not mean that b cannot occur without the occurrence of a. It just means that if a happens then b has to happen. Hence if any observer observes b she cannot deduce that a definitely happened as b may follow or may not follow a. Hence NI holds for M_1. However M_2 has an empty trigger which means it is always true. This implies that the action *always* has to happen and since the action imposes a total ordering on a and b, an observer who observes b (the LO action) will know for sure that a (the HI action) happened. Information will thus flow in a spurious fashion from HI to LO which means that M_2 does not satisfy NI and consequently is "unsafe".

This example shows the necessity of checking information-flow properties like NI on requirements. As is evident, despite the fact that M_1 and M_2 are requirements very

similar to the other one exhibits information-leak while the other does not. This motivates us to look at formal ways of analyzing TMSCs for NI. In Section 2 we saw how NI can be formulated in terms of ready sets on processes. Using that fact, if we may provide a ready set characterization of TMCSs we can obtain a method for checking if a TMSC satisfies the property of NI.

4 From TMSCs to Ready Sets

We will now show how individual TMSCs may be equipped with a ready-set semantics, which makes information-flow analysis on requirements expressed as TMSCs, feasible in practice. The basic idea is that triggers are handled via non-determinism: a TMSC is essentially treated as a non-deterministic choice of all behaviors violating the trigger together with those in which the trigger is satisfied and progresses made on performing the action; and if the action does not terminate the behavior of the instance (i.e. the scenario is partial), then the subsequent allowed behavior is again given by the non-deterministic choice of all possible behaviors.

The formal semantics of single TMSCs is described in detail in [11]. This definition translates a TMSC to an *acceptance tree*. In the TMSC setting, the main difference between the acceptance set semantics of [11] and the ready set semantics we show here, is that the former needs to satisfy a closure property called *saturation*, which simplified the definition of the refinement relation needed in [11], but which is not relevant in the information-flow context. Otherwise, the technical development needed for both the acceptance set and ready set semantics is exactly the same and we will not repeat the account in [11] in its entirety here. Rather, we will present only the relevant definitions for the ready set semantics here, and the interested reader is referred to [11] for details.

In the following, we consider 3 kinds of events: $\mathsf{out}(I_i, I_j, m)$ corresponds to I_i sending m to I_j. I_j will receive m by performing $\mathsf{in}(I_i, I_j, m)$. Also, if I_j is waiting to receive m from I_i but m is yet to be sent, then this will be indicated by the *potential event* $\mathsf{wait}(\mathsf{in}(I_i, I_j, m))$. We will denote by $\mathbb{I}$, $\mathbb{E}$ and $\mathbb{R}$, the set of all instances, all send and receive events, and all receive events respectively.

We assume all events in $\mathbb{E}$ to have a security level associated with them, where the security level has the domain $\{HI, LO\}$. We define the function $sec\_level : \mathbb{E} \longrightarrow \{HI, LO\}$ which maps every event to a security level.

We first define how to associate a ready set with an instance I_i in a TMSC M. The ready set construction differs from the traditional one for LTSs given previously in that it is given relative to a set of "enabled inputs". An instance can only emit an input event if another instance has emitted the corresponding output; otherwise, this input event is not enabled. To capture this behavior, we introduce an additional parameter, $eR \subseteq \mathbb{R}$, into the ready-set definition. An input event in eR is deemed enabled; otherwise, it is defined to be disabled. We also need the following operation on languages. Let $L \subseteq A^*$ and $w \in A^*$. Then the *next set*, $next(L, w) \subseteq A$, of L after w is given by: $next(L, w) = \{a \in A \mid \exists w' \in L.\ w \cdot a \preceq w'\}$. Finally, we define the *nondeterminism set* of $E \subseteq \mathbb{E}$ and enabled inputs $eR \subseteq \mathbb{R}$ as follows.

$$\begin{aligned} ND(E, eR) = (\{\{e\} \mid e \in E \wedge (e \in \mathbb{R} \Rightarrow e \in eR)\} \\ \cup \{\{\mathsf{wait}(r)\} \mid r \in ((E \cap \mathbb{R}) - eR)\}) \end{aligned}$$

$ND(E, eR)$ represents the ready set of a system that can nondeterministically decide to perform any event in E that is *enabled*, where any output or local event, and any input in eR, is enabled, or wait for any input event in E that is not yet enabled.

Definition 1. *Let $I_i \in \mathcal{I}$, $w \in \mathbb{E}^*$ and $eR \subseteq \mathbb{R}_{\{I_i\}}$. Then $ReadySets(I_i, M, w, eR)$ is defined as follows.*

$$ReadySets(I_i, M, w, eR) = \begin{cases} \emptyset \\ \quad \text{if } w \notin L_M(I_i) \\ ND(\mathit{next}(L_M(I_i), w), eR) \\ \quad \text{otherwise} \end{cases}$$

We will first explain some of the notation used above. The *language*, $L_M(I_i)$, of an instance I_i records the possible sequences of events the instance might generate as it executes. Intuitively, if a sequence does not "satisfy" the trigger of I_i, then it will be admitted as a sequence. Otherwise, it will be constrained to "satisfy" the action. $ReadySets(I_i, M, w, eR)$ is the ready set of instance I_i in TMSC M after w. The first clause above handles the case when I_i is incapable of performing w, while the second one computes the ready set based on events that are possible "next" after w, together with any potential receive events that are not enabled.

We assume that any such instance, whose behavior is not explicitly described in M, has empty trigger and action in M and must terminate.

Definition 2. *Let $I_i \in \mathbb{I} - \mathcal{I}$. Then $ReadySet(I_i, M, w, eR)$ is defined as follows.*

$$ReadySet(I_i, M, w, eR) = \begin{cases} \{\{\mathit{end}(I_i)\}\} & \text{if } w = \epsilon \\ \{\emptyset\} & \text{if } w = \mathsf{end}(I_i) \\ \emptyset & \text{otherwise} \end{cases}$$

Thus, any instance $I_i \in \mathbb{I} - \mathcal{I}$ can only perform the event $\mathsf{end}(I_i)$ in M, after which it terminates.

Interpreting TMSCs. We now define the ready set $ReadySets(M, w)$ as follows.

Definition 3. *The ready set $ReadySets(M, w)$, of M after $w \in \mathbb{E}^*$ is defined as:*

$$ReadySets(M, w) = \begin{cases} \emptyset \\ \quad \text{if } w \text{ is not well-balanced} \\ \bowtie_{I_i \in \mathbb{I}} ReadySets(I_i, M, w\lfloor I_i, e\mathcal{R}(w)) \\ \quad \text{otherwise} \end{cases}$$

We call w well-balanced if every receive event is preceded by a corresponding send. The projection, $w\lfloor I_i$ returns I_i's contribution in w, i.e. the longest subsequence of w containing only events in which I_i is active. Also, the receive event $\mathsf{in}(I_i, I_j, m)$ is called *enabled* by w if $|w|_{\mathsf{out}(I_i,I_j,m)} > |w|_{\mathsf{in}(I_i,I_j,m)}$. We use $e\mathcal{R}(w)$ to stand for all receive events enabled by w. Finally, the $\bowtie$ operator takes the pairwise union of a set of ready sets. Intuitively, we first compute the local ready set of each instance I_i after the execution $w\lfloor I_i$, and then combine the local states across all the instances in all possible ways, to generate the global system configurations.

5 Example of Readyset analysis of TMSCs

In section 2, we showed how to characterize TMSCs in terms of ready sets. In order to show non-interference, we need to show that for all traces tr of a TMSC M, $ReadySets_L(M/tr) = ReadySets_L(M/tr \uparrow L))$ where L or LO is obtained from applying the function sec_level on its set of actions.

To illustrate our approach, we take the two TMSCs M_1 and M_2 we had in our initial example where a is a HI action and b is a LO action.

As mentioned before, a TMSC can be looked upon as a non-deterministic choice of all behaviors violating the trigger together with those in which the trigger is satisfied and progress is made on performing the trigger. As a result, for M_1 if the trigger is not satisfied (ie a is not sent from W to R) there are a myriad number of things it can do, some of them being:the sending out and reception of b alone, termination of W at the very beginning, termination of R at the very beginning, the sending out of an event while the receiver is still waiting for it. In other words, for a conditional scenario like M_1 there are many ways in which it may be satisfied.

Revisiting our previous example from the stand-point of the readyset characterization of NI we can see that we get the same result we obtained from intuition. That is in M_2 if the trace considered is $w = out(W, R, a).in(R, W, a).out(R, W, b)$ then readyset of M_2 after the perfomance of this trace projected on the set of low actions ie $RD_L(M_2/w) = \{\{in(W, R, b)\}\}$ is not equal to $RD_L(M_2/w \uparrow L) = \{\emptyset\}$ ie the set containing the empty trace.

However for the same trace in M_1, $RD_L(M_1/w) = \{\{out(R, W, b), in(W, R, b)\}\}$. Since M_1 is a partial scenario, after the performance of $out(R, W, b)$ the actions it can perform (projected on the low alphabet) may be to either receive the emitted b at W or for R to emit another b. For $RD_L(M_1/w \uparrow L)$ the trigger is not satisfied and the projection of the ready set onto the low alphabet will be $\{\{out(R, W, b), in(W, R, b)\}\}$ ie $RD_L(M_1/w) = RD_L(M_1/w \uparrow L)$

These results tie in with the intuition that a system with more redundancy is likely to be more secure with respect to a behavior obfuscation property like non-interference. A partial scenario like M_1 having more non-determinism than a complete scenario like M_2 has a greater likelihood of being secure.

This leads to an important lesson for requirements and software engineers working together to design a system: while going from partial specifications to more refined ones, it should be remembered that the behaviors which are thrown out during the refinement process may be important in keeping the system's behavior obfuscated from an attacker. Hence, for systems in which security is a concern, there is a need to check for information leaks at every stage of the specification and system refinement process.

6 TMSC Expressions

So far we have been working with single TMSCs that serve as the basic building blocks for structured requirements specifications. An algebra of operators is used to generate larger specifications out of sub-specifications. The resulting terms, which are referred

to as *TMSC expressions*, have the following syntax:

$$
\begin{array}{lll}
S ::= M & & \text{(single TMSC)} \\
\quad | \quad X & & \text{(variable)} \\
\quad | \quad S \parallel S & & \text{(parallel composition)} \\
\quad | \quad S \mp S & & \text{(delayed choice)} \\
\quad | \quad S; S & & \text{(sequential composition)} \\
\quad | \quad recX.S & & \text{(recursive operator)} \\
\quad | \quad S \oplus S & & \text{(internal choice)} \\
\quad | \quad S \wedge S & & \text{(logical and)}
\end{array}
$$

The TMSC language offers a selection of "behavioral" and "logical" operators (as opposed to purely behavioral constructs typically used in MSC specifications) to facilitate a structured approach to *requirements management* whereby composite requirements are generated by interweaving prescriptive and constraint-based requirements. $\parallel$, $\mp$, ; and $recX$ falls into the behavioral category, $\wedge$ is a logical construct, while $\oplus$ falls into both categories. The $\parallel$ operator runs two TMSC expressions in parallel. $S_1 \mp S_2$ represents the "deterministic choice" between S_1 and S_2 while $S_1 \oplus S_2$ represents the nondeterministic choice: a successful refinement can choose either. In this respect $\oplus$ has overtones of logical disjunction. $S_1; S_2$ denotes the *asynchronous* sequential composition [6] of S_1 and S_2. The recursive operator, *rec* allows us to model infinite behavior of processes, where a new execution cycle starts whenever there is a reference within S, to the variable used in the recursive definition (say X). Finally, $S_1 \wedge S_2$ represents the logical conjunction of S_1 and S_2, i.e. it specifies a system that needs to satisfy the requirements expressed by both S_1 and S_2. (A detailed account of the use of the TMSC framework in requirements modeling can be found in [9])

The formal semantics of TMSC expressions is given in [10] by translating them to acceptance trees. Specifically, the TMSC operators are interpreted in terms of acceptance trees, i.e. they take the acceptance trees of the sub-expressions as parameters, and generate the acceptance tree of the larger expression. The ready set semantics of TMSC expressions is defined in an analogous manner, the only difference being we use ready sets in place of acceptance sets, and do not use the saturation operator as in [10]. For more details about the approach, the reader is referred to [10].

7 TMSC Expressions and NI

Now an interesting question is whether NI is preserved by the application of these standard operators. Or in other words if M_1 and M_2 are TMSCs that satisfy NI is it mandatory for M_1 op M_2 also satisfy NI.

The answer to this question is that With the exception of $\oplus$ none of the TMSC operators preserve NI. This means that while building up more complex TMSCs from simpler TMSCs the user has to check NI at each stage of composition as there is no guarantee that just because the components are safe the total system will be safe. Due to shortage of space we demonstrate how $\oplus$ preserves NI and how $\wedge$ does not. Counter-examples of the other operators can be constructed similarly and we plan to provide them in an expanded version of this paper.

Internal Choice The $\oplus$ operator offers non-deterministic choice. Given two TMSC specifications S_1 and S_2, and a sequence of events w, let $RD(S_1/w)$ and $RD(S_2/w)$ be the ready sets of S_1 and S_2 after a trace w then $RD[S_1 \oplus S_2/w] = RD(S_1/w) \cup RD(S_2/w)$ as shown in [10]. If A is a readyset then $A \uparrow L$ denotes the A projected on the low actions.

As a result,

$$\begin{aligned} RD_L(S_1 \oplus S_2/w) &= (RD(S_1/w) \cup RD(S_2/w)) \uparrow L \\ &= RD_L(S_1/w) \cup RD_L(S_2/w) \\ &= RD_L(S_1/w \uparrow L) \cup RD_L(S_2/w \uparrow L) \text{ [Assuming } S_1 \text{ and } S_2 \text{ satisfy NI]} \\ &= (RD(S_1/w \uparrow L) \cup RD(S_2/w \uparrow L)) \uparrow L \\ &= RD_L(S_1 \oplus S_2/w \uparrow L) \end{aligned}$$

This shows that that internal choice operator preserves NI.

Logical And In Figure 3 we have 2 TMSCs S_1 and S_2 and their "logical and" $S = S_1 \wedge S_2$ and each $l_i \in L$ and each $h_i \in H$. Again due to shortage of space we do not provide a formal definition for construction of the logical and of the two processes . The intuition is that at each state of the process S_1 we calculate its ready set and logically "and" it with the ready set of the corresponding state in S_2. For example in the state A for S_1 the ready set is $\{\{l_2, h_1\}\}$ while for S_2 it is $\{\{l_2\}\}$ and so state A in S has its ready set as: $\{\{l_2, h_1\}\} \wedge \{\{l_2\}\} = \emptyset$ ie state A in S has no outgoing transitions.

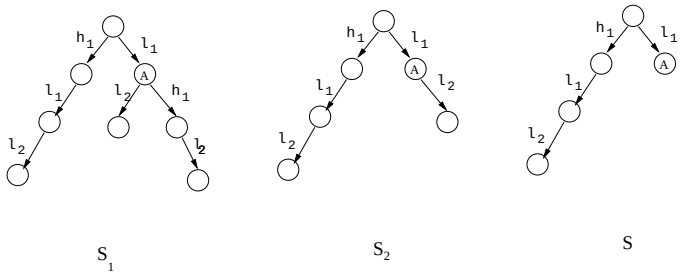

Fig. 3. Two TMSCs S_1 and S_2 and their "logical and" TMSC S

Both S_1 and S_2 satisfy NI because it is evident that by observing l_1 and l_2 we cannot deduce any information about the occurrence or non-occurrence of h_1. However their "logical and" S does not satisfy NI as an observer may observe the occurrence of l_1 and l_2 in sequence and deduce the occurrence of h_1.

The same result may be obtained formally from the ready set characterization of NI. We see that if we consider the trace $w = h_1 l_1$ then $RD_L(S/w) = \{\{l_2\}\}$ while $RD_L(S/w \uparrow L) = \phi$ and hence $RD_L(S/w)$ is not equal to $RD_L(S/w \uparrow L)$

8 Conclusion and Future Work

This paper provides a way in which information-flow analysis can be done on specifications expressed as TMSCs. As we can see that with a ready-set based analysis of non-interference already in place, all that is needed are means to convert specification input formalisms to their respective ready-set characterizations. We have done this for TMSCs in this paper but there are several widely-used formal/informal/semi-formal notations that may be treated similarly. The analysis presented also makes a case for providing a rigorous formal semantics to a modeling/specification language because once one does that, it becomes easier to provide a translation to a form amenable to information flow analysis. With so many system description languages that still have an informal/semi-formal semantics, there is now an added incentive (ie the power to automatedly analyze information flow) to provide them with a formal semantics.

This paper also provides us an important intuition about the application of security analysis at each phase of requirements construction. As we construct more and more refined specifications by the application of operators, we have to keep note of the fact that by potentially removing traces (ie performing refinement) at each stage we may be introducing information leaks due to the removal of redundancy [In the previous section we saw how the $\wedge$ operator removed a trace that led to the violation of NI in S]. This makes it imperative to do NI analysis at each phase of requirements construction.

References

1. Message sequence charts (MSC). *ITU-TS Recommendation Z.120*, 1996.
2. D.E.Denning and P.J.Denning. Certification of programs for secure information flow. *Comm of the ACM*, 20(7):504–513, 1977.
3. D.Wagner. Static analysis and computer security:new techinques for software assurance. *PhD thesis, University of California, Berkeley*, 2000.
4. J.S. Fenton. Information protection systems. *Ph.D thesis, University of Cambridge, England*, 1973.
5. P.Ryan. Mathematical models of computer security–tutorial lectures. *Foundations of Security Analysis and Design*, 2171:1–62, 2001.
6. M.A. Reniers. Message sequence chart: Syntax and semantics. *PhD Thesis, Eindhoven University of Technology*, 1998.
7. R.Focardi, R.Gorrieri, and F.Martinelli. Information flow analysis in a discrete-time process algebra. *IEEE Computer Security Foundations Workshop*, pages 170–184, 2000.
8. Peter Ryan. A csp formulation of non-interference and unwinding. *Presented at CSFW 1990 and published in Cipher*, Winter 1990/91.
9. B. Sengupta and R. Cleaveland. Refinement-based requirements modeling using triggered message sequence charts. *IEEE International Requirements Engineering Conference*, 2003.
10. Bikram Sengupta. Triggered message sequence charts. *Ph.D Thesis, State University of New York, Stony Brook*, 2003.
11. Bikram Sengupta and Rance Cleaveland. Triggered message sequence charts. *Proceedings of ACM SIGSOFT Foundations of Software Engineering*, pages 167–176, 2002.