Time-sensitive Information Flow Control in Timed Event-B

Chunyan Mu and Shengchao Qin
School of Computing, Teesside University, UK
Email: c.mu@tees.ac.uk, s.qin@tees.ac.uk

***Abstract*—Protecting confidential data in today's computing environments is an important problem. Information flow control can help to avoid information leakage and violations introduced by executing the software applications. In software development cycle, it is important to handle security related issues from the beginning specifications at the level of abstract. Mu [1] investigated the problem of preserving information flow security in the Event-B specification models. A typed Event-B model was presented to enforce information flow security and to prevent *direct* flows introduced by the system. However, in practice, *timing behaviours* of programs can also introduce a covert flow. The problem of run-time flow monitoring and controlling must also be addressed. This paper investigates information flow control in the Event-B specification language with timing constructs. We present a timed Event-B system by introducing timers and relevant time constraints into the system events. We suggest a time-sensitive flow security condition for the timed Event-B systems, and present a type system to close the covert channels of timing flows for the system by ensuring the security condition. We then investigate how to refine timed events during the stepwise refinement modelling to satisfy the security condition.**

***Keywords*-flow, security, time, Event-B, refinement.**

I. Introduction

Security threats are everywhere in today's digital life. We often access software applications from untrusted sources, provide private and sensitive data and also have to grant them access permission over the internet. By communicating and observing the running of the software application, the attacker can inflict damage such as stealing information stored in the affected systems. Protecting confidential data in computing environments is an important and difficult problem. Information flow control aims to avoid information leakage and violations introduced by executing the programs, and defend attacks from the application level. Software applications are usually modelled using higher level specification languages before being implemented in programming languages. In the software development cycle, it is important to handle security related issues from the very beginning, *e.g.* when dealing with specifications at the level of abstract. Mu [1] investigated the problem of preserving information flow security in Event-B specification models and during the procedure of refinement. A typed Event-B model was presented to enforce information flow security and prevent explicit and implicit flow introduced by the system. Information flow control considered in [1] focused on the interference among variables with different security levels when executing the model events but without considering information release via timing side channels. Timing behaviours of programs can also introduce a covert flow and bring serious threats to the security of software systems. By analysing the execution time of a software system, an attacker may be able to deduce information about the secret components. Therefore, run-time information flow monitoring and controlling should also be addressed. This paper aims to study *time-sensitive* analysis and control of information flow in the Event-B specification language, built on top of the work in [1].

Consider the execution of a software application downloaded from an untrusted site, attackers can partially observe the execution of the application. Specifically, they can observe a (possibly infinite) sequence of timed observable events, *i.e.,* when does an event take place and what is the value of the observable outputs of that event. By observing the timing and communicating behaviours of the system, secret information flows to the attacker through direct and side communication channels. Semantic security policies are required to ensure the observable time-sensitive behaviours to be independent of secret information manipulated by the program, and enforcement mechanisms are required to guarantee the programs satisfy the policies.

In this paper, we model the system in an extended finite event system for this goal. We extend Event-B models with branch and timing constructs. The extended Event-B model can be used to model real-time systems and analyse time-sensitive information flow through the system. The executions of the model can be viewed as a set of infinite sequences of timed events. Each execution trace is attached with a sequence of times such that the time of the occurrence of the i^{th} event of the trace is recorded as the i^{th} element of the time sequence. Specifically, we present the semantics of the timed Event-B specification language, to support automated reasoning about timing flow sensitive systems. By incorporating timers and timing constraints into Event-B machines, the timed machines can then accept time event sequences in which each event is associated with the time of its occurrence. A timed Event-B system can then capture interesting run-time properties for timing flow analysis and control.

Our main contributions are summarised as follows. First, we present the language of Event-B models with timing and branch constructs, and introduce the relevant semantic

security conditions for the purpose of timing flow analysis and control. Second, we present a type system for the timed Event-B models to ensure the flow security condition. Third, we extend the refinement rules to preserve timing flow security under stepwise refinement.

The rest of the paper is organised as follows. Section II reviews the core Event-B models with an additional branch construct to facilitate flow analysis. Section III introduces the timed Event-B language, and presents the timing flow security condition. Section IV studies time-sensitive information flow analysis in the timed Event-B systems, and presents a type system to ensure the security condition. In Section V, we investigate the problem of preserving timing security properties and timing constructs under refinement. Section VI presents related work, followed by some concluding remarks.

II. Event B with branch construct

Event-B [2] is a formal language for flexible system development via stepwise refinement. The framework supports mechanised proofs and steps for refining, splitting existing events and introducing new events. All such steps are accompanied with mechanised proof obligations which make the verification process efficient. An Event-B model includes two main parts: a context specifies the static part and a machine specifies the dynamic part of the system.

A machine specification normally consists of a list of model variables denoted by $\mathbf{v}$, state invariants $I(\mathbf{v})$, and a set of model events EVT:

```
Machine  M
var v
invariant I(v)
EVT  init,  evt0,  ...
end
```

where $\mathbf{v}$ defines the state variables of the system, $I(\mathbf{v})$ defines the global specification of the state variables, and EVT describes all possible updates to the machine state. A machine `M` is accompanied with consistency proof obligations such that the events preserve the invariant.

Specifically, an event is a "guarded command" consisting of a *guard* $G(\mathbf{v})$ over the variables, and a generalised substitution action (the body) $A(\mathbf{v})$. We consider the events in the following form:

$$\text{EVT} \triangleq \textbf{when } G(\mathbf{v}) \textbf{ then } A(\mathbf{v})$$

where the action (body) $A(\mathbf{v})$ is used to update the state of the machine. The effect of the body defines a "before-after predicate" $\Phi(\mathbf{v}, \mathbf{v}')$ describing state updates of variables upon event execution, in terms of the relationship between the variable values before ($\mathbf{v}$) and after ($\mathbf{v}'$) the action has occurred. The execution of the body ensures that the predicate $\Phi(\mathbf{v}, \mathbf{v}')$ is met.

To reason about time-sensitive flow analysis, we define the branch construct with the restriction using the choice operator []. The branch construct defines a choice between a pair of mutually exclusive events:

$$\text{EVT} \triangleq \textbf{when } G(\mathbf{v}) \textbf{ then } A(\mathbf{v}) \ [] \ \textbf{when } \neg G(\mathbf{v}) \textbf{ then } A'(\mathbf{v})$$

The branch specification can be considered as an *if statement* in the common sequential language:

$$\textbf{if } G(\mathbf{v}) \textbf{ then } A(\mathbf{v}) \textbf{ else } A'(\mathbf{v})$$

The substitution actions of the core Event-B model include:

$$A ::= \texttt{skip} \ \Big| \ \mathbf{x} := \mathbf{e} \ \Big| \ \mathbf{x} :\in S \ \Big| \ \mathbf{z} : |P \ \Big| \ \mathbf{x}, \mathbf{y} := \mathbf{e}, \mathbf{f}$$

The event actions can be viewed as transformation functions which update the model state. Specifically, `skip` denotes the empty set of actions for an event, the state is unchanged under the `skip` action; $\mathbf{x} := \mathbf{e}$ denotes the assignment, *i.e.,* state is updated by replacing free occurrences of $\mathbf{x}$ by $\mathbf{e}$: where $\mathbf{x} \subseteq$ VAR is a sequence of variables, and $\mathbf{e}$ denotes a number of set-theoretic expressions corresponding to each of the variables in $\mathbf{x}$; $\mathbf{x} :\in S$ denotes that we update the state by arbitrarily choosing values from the set S for the variables in $\mathbf{x}$, *i.e.,* $\mathbf{x}$ becomes a member of S; $\mathbf{z} : |P$ denotes that we update the state by arbitrarily choosing values for the variables in $\mathbf{z}$ that satisfy the predicate P, *i.e.,* $\mathbf{x}$ becomes such that the predicate P holds; $\mathbf{x}, \mathbf{y} := \mathbf{e}, \mathbf{f}$ denotes a concurrent assignment of the values $\mathbf{e}$ and $\mathbf{f}$ to the variable sequences $\mathbf{x}$ and $\mathbf{y}$ respectively.

III. Timed Event B system and flow security condition

A. The language

We extend Event B machines with timing aspects to support automated analysis, reasoning about, and eventually controlling time-sensitive information flows. We consider the form of the evaluation judgements for actions as: $\langle [\![A]\!], \Sigma\rangle \xrightarrow{\iota} \Sigma'$ where Σ is the state space associating model variables with values, ι is the time taken to make this state transition. Specifically, time expression ι can be any of the following regarding to the action taken: $\iota_{\mathbf{e}}$ denote the time taken to evaluate expression $\mathbf{e}$; $\iota_{[\![:=]\!]}$ denote the time taken to make a single assignment; $\iota_{[\![:\in]\!]}$ denote the time taken to randomly choose values from a set; $\iota_{[\![:|]\!]}$ denote the time taken to randomly choose values satisfying a predicate; ι_G denote the time taken to judge a guard G; we also use ι_A and ι_E to denote the time taken to perform action A and event E in general. We therefore incorporate a notion of execution time into the semantics. Table I presents the rules for the evaluation of model events.

B. Timed event sequences

We model the system in an extended finite event system with timing constructs. The executions of the model can be viewed as a set of infinite sequences of events. Each execution trace is attached with a sequence of times such that the time occurred by the i^{th} event of the trace is recorded

(Event)
$$\frac{\Sigma \vdash G \Downarrow \texttt{true}}{\langle \texttt{when } G \texttt{ then } A, \Sigma\rangle \xrightarrow{\iota_G+\iota_A} \Sigma[A]} \qquad \frac{\Sigma \vdash G \Downarrow \texttt{false}}{\langle \texttt{when } G \texttt{ then } A, \Sigma\rangle \xrightarrow{\iota_G} \Sigma}$$

(BranchEvent)
$$\frac{\Sigma \vdash G \Downarrow \texttt{true}}{\langle \textbf{when } G(\mathbf{v}) \textbf{ then } A(\mathbf{v}) \ [] \textbf{ when } \neg G(\mathbf{v}) \textbf{ then } A'(\mathbf{v}), \Sigma\rangle \xrightarrow{\iota_G+\iota_A} \Sigma[A]}$$

$$\frac{\Sigma \vdash G \Downarrow \texttt{false}}{\langle \textbf{when } G(\mathbf{v}) \textbf{ then } A(\mathbf{v}) \ [] \textbf{ when } \neg G(\mathbf{v}) \textbf{ then } A'(\mathbf{v}), \Sigma\rangle \xrightarrow{\iota_G+\iota_{A'}} \Sigma[A']}$$

(Skip)
$$\frac{}{\langle \texttt{skip}, \Sigma\rangle \xrightarrow{0} \Sigma}$$

(Assignment)
$$\frac{\Sigma \vdash \mathbf{e} \Downarrow \mathbf{w}}{\langle \mathbf{x} := \mathbf{e}, \Sigma\rangle \xrightarrow{\iota_{\mathbf{e}}+\iota_{[:=]}\cdot|\mathbf{x}|+\iota_{\mathbf{x}}} \Sigma[\mathbf{x} = \mathbf{w}]}$$

(ChoiceFromSet)
$$\frac{\Sigma \vdash S \Downarrow \{s_i \mid s_i \in S, 1 \le i \le |\mathbf{x}|\}}{\langle \mathbf{x} :\in S, \Sigma\rangle \xrightarrow{\iota_{[:\in]}+\iota_{[:=]}\cdot|\mathbf{x}|+\iota_{\mathbf{x}}} \Sigma[\mathbf{x} = \{s_i \mid 1 \le i \le |\mathbf{x}|\}]}$$

(ChoiceByPredicate)
$$\frac{\Sigma \vdash P \Downarrow \{x_i \mid x_i \models P, 1 \le i \le |\mathbf{z}|\}}{\langle \mathbf{z} : |P, \Sigma\rangle \xrightarrow{\iota_{[:|]}+\iota_{[:=]}\cdot|\mathbf{z}|+\iota_{\mathbf{z}}} \Sigma[\mathbf{z} = \{x_i \mid 1 \le i \le |\mathbf{z}|\}]}$$

(MultipleAction)
$$\frac{\Sigma \vdash \mathbf{e_1} \Downarrow \mathbf{w_1}, \mathbf{e_2} \Downarrow \mathbf{w_2}}{\langle \mathbf{x}, \mathbf{y} := \mathbf{e_1}, \mathbf{e_2}; \Sigma\rangle \xrightarrow{\iota_{[:=]}\cdot|\mathbf{x}|+\iota_{[:=]}\cdot|\mathbf{y}|+\iota_{\mathbf{e_1}}+\iota_{\mathbf{e_2}}+\iota_{\mathbf{x}}+\iota_{\mathbf{y}}} \Sigma[\mathbf{x} = \mathbf{w_1}, \mathbf{y} = \mathbf{w_2}]}$$

Table I
SEMANTICS: RULES FOR THE EVALUATION OF TIMED MODEL EVENT.

as the i^{th} element of the time sequence. By incorporating timers and timing constraints into Event B machines, the timed machines can then accept time event sequences in which each event is associated with the time taken and thus the time of its occurrence. A timed Event-B system can then capture interesting flow properties introduced by timing channels.

A timed event sequence is a pair $(\mathbf{e}, \mathbf{c})$ where $\mathbf{e} = e_1 e_2 \ldots$ is an infinite event sequence, $\mathbf{c} = c_0 c_1 c_2 \ldots$ is a time sequence, and e_i occurs at clock c_i, *i.e.,* e_i takes the time of $c_{i+1} - c_i$, where $c_0 = \alpha$ denotes that $\mathbf{e}$ starts at time α. With each event, we associate the guard with time constraints, and require that the event is taken only if the current values of the clocks satisfy this constraint:

$$\textsc{Evt} \triangleq \textbf{when } G(\mathbf{v}, \mathbf{c}) \textbf{ then } A(\mathbf{v}).$$

The definition for a timed event sequence is as follows.

Definition 1 (Timed event sequence): A timed event sequence is defined as: $\rho = (M, \mathbf{c})$, where $M = e_1 e_2 \ldots e_n$, $\mathbf{c} = c_1 \ldots$, and $c_i = c(e_i)$ $(i \geq 1)$ denotes the time instant e_i occurs.

C. Time-sensitive flow security condition

To reason about flow analysis, we assign security levels to model objects. Given a timed Event-B model, all entities are built from a set of timed events, each of which is defined in terms of substitution actions on model variables and recordings on event timers. Each variable is thus assigned with a security level. The powerset of model variables therefore forms a complete lattice $\mathcal{L}$, where the partial ordering is regarding to the security levels of the model variables: $\forall v_1, v_2 \in \textsc{Var},\ v_1 \preceq v_2$ *iff* $\tau_1 \sqsubseteq_{\texttt{sec}} \tau_2$, where $\preceq$ denotes the partial ordering on VAR, and $\tau_1 = \tau(v_1), \tau_2 = \tau(v_2) \in \mathcal{L}$ denote the security levels of variables v_1 and v_2 respectively. Execution of model events causes interference between model variables and thus a flow through the security lattice. A model is considered *secure* if all the flows in the model satisfy its flow policy. This section studies the *timing flow policy* that a secure model should satisfy.

First let us look at a simple example to see how timing behaviours can introduce leakage.

Example 1: Consider a simple Event-B model as follows.

variables $high, low_1, low_2$
invariants $low_1, low_2 \in \mathbb{Z},\ high \in \{-1, 0, 1\}$
initialisation $low_1 := -1 \quad low_2 := -1 \quad high := -1$
events
$\textsc{Init} \triangleq$ (**when** `true` **then** $low_1, low_2 := 0, 0$)
$\textsc{Evt}_1 \triangleq$ (**when** $(low_1 == 0)$ **then** $high :\in \{0, 1\}$)
$\textsc{Evt}_2 \triangleq$ (**when** $(high == 1)$
then $low_1, low_2 := low_1 + 2, low_1$)
[](**when** $(high == 0)$
then $low_1 := 2$)

Assume that $low_1, low_2 \preceq high$. Note that low_1 (low_2) will always be 2 (0) at the end and no secret data is leaked by observing the low output. However the program may leak the value of $high$ via its timing behaviour since the execution

time is dependent on the value of $high$ from the observer's view.

To capture time-sensitive flow leakage, we define the security condition for timed Event-B models based on equivalence relations from the observer's view.

Definition 2 (L-equivalent state space): We say state space Σ_1 and Σ_2 are L-equivalent, written as $\Sigma_1 =_L \Sigma_2$ iff

$$\textsc{Dom}(\Sigma_1) = \textsc{Dom}(\Sigma_2) \ \wedge \ \forall x \in \textsc{Dom}(\Sigma_1).\tau(x) \leq L \Rightarrow \Sigma_1(x) = \Sigma_2(x).$$

Definition 3 (L-bisimulation $\sim_L$): Consider two timed event sequences:

$$\rho_1 = (M_1, \mathbf{c}_1) = \langle E_{10}, (\Sigma_{10}, c_{10})\rangle \rightarrow \ldots,$$
$$\rho_2 = (M_2, \mathbf{c}_2) = \langle E_{20}, (\Sigma_{20}, c_{20})\rangle \rightarrow \ldots,$$

$\forall \Sigma_{10}, \Sigma_{20}$ such that $\Sigma_{10} =_L \Sigma_{20}$, and $c_{10} = c_{20}$, we say $\rho_1 \sim_L \rho_2$ if:

$$\forall j \in \{1, \ldots\}.(\Sigma_{1j} =_L \Sigma_{2j}) \wedge (c_{1j} = c_{2j}).$$

Definition 4 (Timing flow security condition $\phi_{\mathtt{sec}}$): A model M is considered timing flow secure, written as $M \models \phi_{\mathtt{sec}}$, if for a given public security level L:

$$\forall \rho_1, \rho_2 \in (M, \mathbf{c}).\rho_1 \sim_L \rho_2.$$

Consider again the timed events in Example 1. There are two possible event sequences regarding to the input value of $high$.

$\rho_1 = \langle \textsc{Init}, (\Sigma(low_1, low_2 = -1, high = -1), c_{10} = 0)\rangle \rightarrow$
$\langle \textsc{Evt}_1, (\Sigma(low_1, low_2 = 0, high = -1), c_{11} = 2 \times \iota_{[\![:=]\!]} + \iota_{low_1} + \iota_{low_2})\rangle \rightarrow$
$\langle \textsc{Evt}_2, (\Sigma(low_1, low_2 = 0, high = 1), c_{12} = c_{11} + \iota_{[\![:\in]\!]} + \iota_{high})\rangle \rightarrow$
$(\Sigma(low_1 = 2, low_2 = 0, high = 1),$
$c_{13} = c_{12} + 2 \times \iota_{[\![:=]\!]} + 2 \times \iota_{low_1} + \iota_{(low_1 + 2)})$

$\rho_2 = \langle \textsc{Init}, (\Sigma(low_1, low_2 = -1, high = -1), c_{20} = 0)\rangle \rightarrow$
$\langle \textsc{Evt}_1, (\Sigma(low_1, low_2 = 0, high = -1), c_{21} = 2 \times \iota_{[\![:=]\!]} + \iota_{low_1} + \iota_{low_2})\rangle \rightarrow$
$\langle \textsc{Evt}_2, (\Sigma(low_1, low_2 = 0, high = 0), c_{22} = c_{21} + \iota_{[\![:\in]\!]} + \iota_{high})\rangle \rightarrow$
$(\Sigma(low_1 = 2, low_2 = 0, high = 0), c_{23} = c_{22} + \iota_{[\![:=]\!]})$

Note that the model leaks information via its timing channel, since the amount of the time taken is dependent on the value of $high$.

IV. Type system for flow control in timed event B systems

This section presents a type system that closes any covert channels of timing information flows for the model by ensuring the security condition.

A. Type checking for information flow control

Event actions may cause information to flow among variables. Secure information flow is described by a secure information flow predicate using typing rules. Let $\mathcal{L}$ be the finite flow lattice. $\tau \subseteq \mathcal{L}$ denotes a sequence of security levels in $\mathcal{L}$ related to a sequence $\mathbf{x}$ of variables with the same length: $|\tau| = |\mathbf{x}|$. The security typing environment is considered as: $\Gamma : \textsc{Var} \rightarrow \mathcal{L}$. In a model event, for an action A, judgements have the form: $\tau \vdash \Gamma\{A(\mathbf{x})\}\Gamma'$, where the type τ denotes the counter level of the variable sequence $\mathbf{x}$ regarding to the action $A(\mathbf{x})$ being executed to eliminate implicit flows from the *guard*, Γ and Γ' describe the security levels of the identifiers which hold before and after the execution of A. The derivation rules of model events enforce that only variables with types greater than or equal to τ are allowed to be updated by action A.

Model events can influence each other and introduce information flows. In addition to the dependence relationship discussed in [1], the *branch specification* might introduce *implicit* and *timing* flows when branch on high data (Example 1). To close such flows, we need to ensure that the external observer cannot deduce which branch is taken, *i.e.,* $A \sim_L A'$ for the observable security level L.

Table II presents security typing rules for specifying time-sensitive secure information flow predicates of events, as an extension of the type system presented in [1]. Notation $\Gamma \vdash \mathbf{e} : \mathbf{t}$ denotes that under the type environment Γ, expression $\mathbf{e}$ has type $\mathbf{t}$. The type of an expression (including guard expression) is defined by taking the least upper bound of the types of its free variables:

$$\Gamma \vdash \mathbf{e} : \mathbf{t} \ \ \textit{iff} \ \ \mathbf{t} = \bigsqcup_{\mathbf{v} \in \mathtt{fv}(\mathbf{e})} \mathbf{\Gamma}(\mathbf{v})$$

A flow secure program should only safely branch on sensitive data if the public observer is not able to determine which branch was taken. Rule **TBranchEvent1** specifies the case that guard G contains higher level data. For this case, we need to ensure that $A \sim_L A'$.

Theorem 1 (Soundness of the type system): A timed Event-B model M is secure if the events satisfy the security properties, which are guarded by the typing rules defined in Table II:

$$\tau \vdash_{\mathcal{L}} \Gamma\{M\}\Gamma' \Rightarrow M \models \phi_{\mathtt{sec}}$$

Proof: We need to prove that, for M s.t. $\tau \vdash_{\mathcal{L}} \Gamma\{M\}\Gamma'$:

$$\forall \rho_1, \rho_2 \in (M, \mathbf{c}).\rho_1 \sim_L \rho_2$$

regarding to the security condition defined in Definition 4, where ρ_1, ρ_2 are any two timed event executions of M, say:

$$\rho_i = \langle E_{i0}, (\Sigma_{i0}, c_{i0})\rangle \rightarrow \cdots \rightarrow (\Sigma_{in}, c_{in})$$

and for $i = 1, 2$, $E_{i0} \ldots E_{in-1}$ is a sequence of events of M. By Definition 3, this is equivalent to prove that:

$$\forall j \in \{0, \ldots, n\}.(\Sigma_{1j} =_L \Sigma_{2j}) \ \wedge \ (c_{1j} = c_{2j}).$$

By Definition 2, we write the equivalent semantic security relation as:

$$\tau \models_{\mathcal{L}} \Gamma\{E_{ij}\}\Gamma',$$

(TSub) $$\frac{\tau_1 \vdash \Gamma_1\{\textsc{Evt}\}\Gamma_1' \quad \tau_2 \sqsubseteq_{\mathsf{sec}} \tau_1,\ \Gamma_2 \sqsubseteq_{\mathsf{sec}} \Gamma_1,\ \Gamma_1' \sqsubseteq_{\mathsf{sec}} \Gamma_2'}{\tau_2 \vdash \Gamma_2\{\textsc{Evt}\}\Gamma_2'}$$

(TEvent) $$\frac{\Gamma \vdash G : t \quad \mathbf{t} \sqcup \tau \vdash \Gamma\{A\}\Gamma'}{\tau \vdash \Gamma\{\texttt{when } G \texttt{ then } A\}\Gamma'}$$

(TBranchEvent1) $$\frac{\Gamma \vdash G : \mathbf{t} \sqsupset \tau \quad \mathbf{t} \vdash \Gamma\{A\}\Gamma' \quad \mathbf{t} \vdash \Gamma\{A'\}\Gamma'}{\tau \vdash \Gamma\{\texttt{when } G \texttt{ then } A; \texttt{when } \neg G \texttt{ then } A'\}\Gamma'} \quad A \sim_L A'$$

(TBranchEvent2) $$\frac{\Gamma \vdash G : \mathbf{t} \sqsubseteq \tau \quad \mathbf{t} \vdash \Gamma\{A\}\Gamma' \quad \mathbf{t} \vdash \Gamma\{A'\}\Gamma'}{\tau \vdash \Gamma\{\texttt{when } G \texttt{ then } A; \texttt{when } \neg G \texttt{ then } A'\}\Gamma'}$$

(TDepEvts) $$\frac{\Gamma_0\{\textsc{Evt}_1\}\Gamma_1 \quad \Gamma_1\{\textsc{Evt}_2\}\Gamma_2 \quad \ldots \quad \Gamma_{n-1}\{\textsc{Evt}_n\}\Gamma_n}{\Gamma\{\textsc{Evt}_1 \rhd \textsc{Evt}_2 \rhd \cdots \rhd \textsc{Evt}_n\}\Gamma_n}$$

(TSkip) $$\frac{}{\tau \vdash \Gamma\{\texttt{skip}\}\Gamma}$$

(TAssign) $$\frac{\Gamma \vdash \mathbf{e} : \tau'}{\tau \vdash \Gamma\{\mathbf{x} := \mathbf{e}\}\Gamma'(\mathbf{x} \mapsto \tau \sqcup \tau')}$$

(TChoiceFromSet) $$\frac{t = \bigsqcup_{s \in S} \Gamma(s) \quad \tau' = \mathbf{t} \wedge |\tau'| = |\mathbf{x}|}{\tau \vdash \Gamma\{\mathbf{x} :\in S\}\Gamma'(\mathbf{x} \mapsto \tau \sqcup \tau')}$$

(TChoiceByPredicate) $$\frac{t = \bigsqcup_{v \in \mathtt{fv}(P)} \Gamma(v) \quad \Gamma \vdash P : \tau' = \mathbf{t}}{\tau \vdash \Gamma\{\mathbf{x} : |P\}\Gamma'(\mathbf{x} \mapsto \tau \sqcup \tau')}$$

(TMultipleAction) $$\frac{\Gamma \vdash :\tau_1 \quad \Gamma \vdash \mathbf{f} : \tau_2}{\tau \vdash \Gamma\{\mathbf{x}, \mathbf{y} := \mathbf{e}, \mathbf{f}\}\Gamma'(\mathbf{x} \mapsto \tau \sqcup \tau_1, \mathbf{y} \mapsto \tau \sqcup \tau_2)} \quad (\{\mathbf{x}\} \cap \{\mathbf{y}\} = \emptyset)$$

$$\frac{\Gamma \vdash \mathbf{e} : \tau_1 \quad \Gamma \vdash \mathbf{f} : \tau_2}{\begin{array}{ll}\tau \vdash & \Gamma\{\mathbf{x}, \mathbf{y} := \mathbf{e}, \mathbf{f}\} \\ & \Gamma'(\{v_i\} \mapsto \tau(\{v_i\}) \sqcup \tau_1(\{v_i\}) \sqcup \tau_2(\{v_i\}), \\ & \quad \mathbf{x} \setminus \{v_i\} \mapsto \tau(\mathbf{x} \setminus \{v_i\}) \sqcup \tau_1(x \setminus \{v_i\}), \\ & \quad \mathbf{y} \setminus \{v_i\} \mapsto \tau(\mathbf{y} \setminus \{v_i\}) \sqcup \tau_2(\mathbf{y} \setminus \{v_i\}))\end{array}} \quad (\{\mathbf{x}\} \cap \{\mathbf{y}\} = \{v_i\})\ \textit{for}\ 1 \leq i \leq n$$

Table II
TYPING RULES FOR MODEL EVENTS WITH TIMING-SENSITIVE INFORMATION FLOW CONTROL

which holds *iff*: $\forall \Sigma_{ij}, \Sigma_{ij+1}$, $x \in \textsc{Var}$, $\tau(x) \leq L$, $i = 1, 2$ and for $\Sigma_{10} =_L \Sigma_{20}$, if:

$$\langle E_{ij}, (\Sigma_{ij}, c_{ij})\rangle \Downarrow (\Sigma_{ij+1}, c_{ij+1})$$

then:

$$(\Gamma'(x) \sqsubseteq \Gamma(x) \vee \Sigma_{1j+1}(x) = \Sigma_{2j+1}(x)) \wedge (c_{1j+1} = c_{2j+1}).$$

The proof is then concluded by induction on the depth of the typing derivation. ■

V. TIMING FLOW CONTROL UNDER REFINEMENT

In this section, we investigate how to check that the specification is consistent with the flow policy and that the security properties are preserved under refinement. Specifically, we are interested in preserving the security condition under vertical refinement between a higher level abstract machine and a relatively concrete one.

A system specification model M' is a refinement of a specification model M if and only if $M' \sqsubseteq_{\mathtt{ref}} M$. Intuitively, M' is more accurate or less abstract than M. M is secure if the events satisfy the security properties which are guarded by the typing rules defined in Table II. However, by the definition of refinement there is no guarantee that a refinement of M will preserve the security properties since the refined model might introduce new or merge existing events. Therefore, secure information flow properties are not always preserved by refinement. The reason for this is that the secure flow properties and the relevant typing rules depend on the semantics of events and the type environment, which cannot guarantee that the refinement transformation preserve the security properties of interest. Even the security requirement can be viewed as security predicates, some of the events and their actions that satisfy the rules may be removed or merged during the refinement. Therefore, proving the security property at one abstract level is not enough in general. Relevant security properties must be proven again at the concrete level or ensured via the refinement transformation to guarantee that these additional behaviours introduced via refinement do not violate security flow policy. In this section, we study the problem of preserving security properties under refinement transformation. We extend the secure refinement rules of

events presented in [1] in order to prove that a refinement is both timing flow-sensitive secure and correct.

Refinement provides a way to construct stronger invariants and add details in a way to enrich it step by step. This is generally achieved by extending the list of state variables, by merging and refining existing events, and by adding new events. Specifically, assume the lower level model denoted by $M_{\mathcal{C}}$ has a collection of state variables denoted by $\mathbf{v}_{\mathcal{C}}$, which is distinct from the collection of variables denoted by $\mathbf{v}_{\mathcal{A}}$ in the abstract model denoted by $M_{\mathcal{A}}$. The abstract variables $\mathbf{v}_{\mathcal{A}}$ and the concrete ones $\mathbf{v}_{\mathcal{C}}$ are linked together by means of *gluing invariant* $J(\mathbf{v}_{\mathcal{A}}, \mathbf{v}_{\mathcal{C}})$: it "glues" the state of the concrete model $M_{\mathcal{C}}$ to that of its abstraction $M_{\mathcal{A}}$. For any variable in abstract (concrete) machine if there is no glued variable in the relevant concrete (abstract) machine, we introduce a virtual variable denoted by $\perp$ to represent it, *i.e.,* we extend the set of the variables such that: $\text{VAR}^*_{\mathcal{A}} = \text{VAR}_{\mathcal{A}} \cup \{\perp\}$, and $\text{VAR}^*_{\mathcal{C}} = \text{VAR}_{\mathcal{C}} \cup \{\perp\}$. For our purpose of security concern, we introduce virtual variables called $\perp$ to denote the glued variable which is empty in the abstract (concrete) machine regarding to gluing variable in the relevant concrete (abstract) machine. Furthermore, a variant V is introduced to prevent executions "non-terminating": V has to be decreased by every *convergent event* and must not be increased by *anticipated events*. Assume the type environments for abstract event and concrete event are denoted as $\Gamma_{\mathcal{A}}$ and $\Gamma_{\mathcal{C}}$ respectively. We extend the refinement laws [1] to incorporate rules *w.r.t. time-sensitive* security properties in Table III. For clear cases, we use $\Psi(\mathbf{v}, \mathbf{v}')$ to denote the before and after states of variable(s) $\mathbf{v}$ instead of writing $\Psi(\Sigma(\mathbf{v}), \Sigma'(\mathbf{v}))$.

Assume abstract event $E_{\mathcal{A}}$ with guard $G_{\mathcal{A}}(\mathbf{v}_{\mathcal{A}}, \mathbf{c}_{\mathcal{A}})$ and before-and-after predicate $\Psi_{\mathcal{A}}(\mathbf{v}_{\mathcal{A}}, \mathbf{v}'_{\mathcal{A}})$ in $M_{\mathcal{A}}$ is refined to a relevant concrete event $E_{\mathcal{C}}$ with guard $G_{\mathcal{C}}(\mathbf{v}_{\mathcal{C}}, \mathbf{c}_{\mathcal{C}})$ and before-and-after predicate $\Psi_{\mathcal{C}}(\mathbf{v}_{\mathcal{C}}, \mathbf{v}'_{\mathcal{C}})$ in $M_{\mathcal{C}}$. Let $\tau_{\mathcal{A}} \subseteq \mathcal{L}_{\mathcal{A}}$ be a sequence of security levels of the sequence of variables $\mathbf{v}_{\mathcal{A}}$ and $\tau_{\mathcal{C}} \subseteq \mathcal{L}_{\mathcal{C}}$ be a sequence of security levels of the sequence of variables $\mathbf{v}_{\mathcal{C}}$. We say $\tau_{\mathcal{C}} \sqsubseteq_{\mathtt{sec}} \tau_{\mathcal{A}}$, *iff*:

$$\forall v_a \in \mathbf{v}_{\mathcal{A}},\ v_c \in \mathbf{v}_{\mathcal{C}},\ J(v_a, v_c) \in J(\mathbf{v}_{\mathcal{A}}, \mathbf{v}_{\mathcal{C}}) \Rightarrow \tau_c \sqsubseteq_{\mathtt{sec}} \tau_a.$$

Specifically, **BranchEvtSec-REF** specifies the secure refinement for branch events. The rule ensures that the actions in true branch and false branch are L-bisimilar to each other, and thus the external observer cannot determine which branch is taken when branching on high data. We briefly review the rest of the refinement rules discussed in [1] for consistence. **SubSec-REF** ensures that for glued variables, the security level of variables in the concrete event will not be higher than that of their glued variables in the abstract event. **FisSec-REF** ensures that the refined event is feasible and there is no unauthorised flow introduced by the before-after predicate. **GrdSec-REF** and **InvSec-REF** ensure that the concrete event is a correctly refined event regarding to the abstract one and the security level of the corresponding variables will not be higher than that of the glued ones in the abstract event. **MergeSec-REF** specifies the secure refinement for merging existing events. Several abstract events can be merged being refined to a single concrete event. New events might be introduced in a refinement. **NewEvSec-REF** ensures that adding new event during the refinement will not introduce secure information flow and will be correct. As usual, each new event refines an implicit `skip` event which satisfies the security typing rules, and the non-divergence rule.

VI. Related Work

This paper studied the problem of time-sensitive secure information flow control in Event-B specifications. The notion of secure information flow specifies the security requirements of the system where there should be no information flow from the sensitive data to the observer. Denning and Denning [3] first use program analysis to investigate if the information flow properties of a program satisfy a specified multi-level security policy. The most commonly used flow police is *non-interference* [4]. A program satisfies the non-interference property if its low security outputs do not depend on the high security inputs. Note that under the view of non-interference property, the program is viewed as a function from input to output. Therefore, information can still be leaked to the external observer through timing channels.

Security type systems have been substantially used to formulate the analysis of secure information flow and enforce the non-interference property in programs. Hunt and Sands [5] present a type system for information flow control in a while language. In [5], sensitive information is stored in programming variables, the powerset of program variables thus forms the universal lattice. Their flow-sensitive type system is defined by a family of inference systems which is forced to satisfy a simple non-interference property. Their further work [6] shows how flow-sensitive multi-level security typing can be achieved in polynomial time. However, no timing leakage has been considered in their works.

There are a number of works dealing with time-sensitive notions of secure information flow for programming languages by using security type systems. Volpano and Smith [7] propose a type-based approach that prevents information flow leakage by enforcing the non-interference police. Timing leaks to external observers are closed by enforcing both branching and looping conditions to be independent of sensitive data in sequential programs. They then investigate flow security in a multi-threaded language by enforcing the same condition to close the flow leakage through termination behaviour [8]. Furthermore, the same authors have also studied the issue of timing leakages through the probabilistic behaviours scheduling among concurrent threads in [9]. Sabelfeld and Sands [10] explore external timing-sensitive security conditions for multi-

(SubSec-REF)	$I(\mathbf{v}_\mathcal{A}) \ \wedge \ J(\mathbf{v}_\mathcal{A}, \mathbf{v}_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash \mathbf{v}_\mathcal{A} : \tau_\mathcal{A}) \ \wedge \ (\Gamma_\mathcal{C} \vdash \mathbf{v}_\mathcal{C} : \tau_\mathcal{C}) \implies \tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A}$
(FisSec-REF)	$I(\mathbf{v}_\mathcal{A}) \ \wedge \ J(\mathbf{v}_\mathcal{A}, \mathbf{v}_\mathcal{C}) \ \wedge \ G_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{c}_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{C} \vdash \mathbf{v}_\mathcal{C} : \tau_\mathcal{C}) \ \wedge \ (\Gamma'_\mathcal{C} \vdash \mathbf{v}'_\mathcal{C} : \tau'_\mathcal{C}) \implies \exists \mathbf{v}'_\mathcal{C}.\Psi_\mathcal{C}(\mathbf{v}_\mathcal{C}, v'_\mathcal{C}) \ \wedge \ (\tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{C})$
(GrdSec-REF)	$I(\mathbf{v}_\mathcal{A}) \ \wedge \ J(\mathbf{v}_\mathcal{A}, v_\mathcal{C}) \ \wedge \ G_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{c}_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{C} \vdash G_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{c}_\mathcal{C}) : \tau_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash G_\mathcal{A}(\mathbf{v}_\mathcal{A}, \mathbf{c}_\mathcal{A}) : \tau_\mathcal{A}) \implies G_\mathcal{A}(\mathbf{v}_\mathcal{A}, \mathbf{c}_\mathcal{A}) \ \wedge \ (\tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A})$
(InvSec-REF)	$I(\mathbf{v}_\mathcal{A}) \ \wedge \ J(\mathbf{v}_\mathcal{A}, \mathbf{v}_\mathcal{C}) \ \wedge \ G_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{c}_\mathcal{C}) \ \wedge \ \Psi_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{v}'_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{C} \vdash \mathbf{v}_\mathcal{C} : \tau_\mathcal{C}, \Gamma'_\mathcal{C} \vdash \mathbf{v}'_\mathcal{C} : \tau'_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash \mathbf{v}_\mathcal{A} : \tau_\mathcal{A}, \Gamma'_\mathcal{A} \vdash \mathbf{v}'_\mathcal{A} : \tau'_\mathcal{A})$ $\implies \exists \mathbf{v}'_\mathcal{A}.(\Psi_\mathcal{A}(\mathbf{v}_\mathcal{A}, \mathbf{v}'_\mathcal{A}) \wedge J(\mathbf{v}'_\mathcal{A}, \mathbf{v}'_\mathcal{C})) \ \wedge \ (\tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A}) \ \wedge \ (\tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau'_\mathcal{A} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A})$
(Branch1Sec-REF)	$(E \triangleq \texttt{when}\ G(\mathbf{v}, \mathbf{c})\ \texttt{then}\ A(\mathbf{v})\ []\ \texttt{when}\ \neg G(\mathbf{v}, \mathbf{c})\ \texttt{then}\ A'(\mathbf{v})\ \texttt{end}) \ \wedge \ I(\mathbf{v}_\mathcal{A}) \ \wedge \ J(\mathbf{v}_\mathcal{A}, \mathbf{v}_\mathcal{C}) \ \wedge \ G_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{c}_\mathcal{C}) \ \wedge$ $(\Gamma_\mathcal{C} \vdash \mathbf{v}_\mathcal{C} : \tau_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash \mathbf{v}_\mathcal{A} : \tau_\mathcal{A}) \ \wedge \ (\Gamma_\mathcal{C} \vdash G_\mathcal{C} : t_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash G_\mathcal{A} : t_\mathcal{A}) \ \wedge \ t_\mathcal{A} \sqsupseteq_{\mathsf{sec}} \tau_\mathcal{A} \ \wedge \ t_\mathcal{C} \sqsupseteq_{\mathsf{sec}} \tau_\mathcal{C}$ $\implies G_\mathcal{A}(\mathbf{v}_\mathcal{A}, \mathbf{c}_\mathcal{A}) \ \wedge \ (A_\mathcal{C}(\mathbf{v}_\mathcal{C}) \sim_L A'_\mathcal{C}(\mathbf{v}_\mathcal{C})) \ \wedge \ (A_\mathcal{A}(\mathbf{v}_\mathcal{A}) \sim_L A'_\mathcal{A}(\mathbf{v}_\mathcal{A})) \ \wedge \ (\tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A})$
(Branch2Sec-REF)	$(E \triangleq \texttt{when}\ G(\mathbf{v}, \mathbf{c})\ \texttt{then}\ A(\mathbf{v})\ []\ \texttt{when}\ \neg G(\mathbf{v}, \mathbf{c})\ \texttt{then}\ A'(\mathbf{v})\ \texttt{end}) \ \wedge \ I(\mathbf{v}_\mathcal{A}) \ \wedge \ J(\mathbf{v}_\mathcal{A}, \mathbf{v}_\mathcal{C}) \ \wedge \ \neg G_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{c}_\mathcal{C}) \ \wedge$ $(\Gamma_\mathcal{C} \vdash \mathbf{v}_\mathcal{C} : \tau_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash \mathbf{v}_\mathcal{A} : \tau_\mathcal{A}) \ \wedge \ (\Gamma_\mathcal{C} \vdash G_\mathcal{C} : t_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{A} \vdash G_\mathcal{A} : t_\mathcal{A}) \ \wedge \ t_\mathcal{A} \sqsupseteq_{\mathsf{sec}} \tau_\mathcal{A} \ \wedge \ t_\mathcal{C} \sqsupseteq_{\mathsf{sec}} \tau_\mathcal{C}$ $\implies \neg G_\mathcal{A}(\mathbf{v}_\mathcal{A}, \mathbf{c}_\mathcal{A}) \ \wedge \ (A_\mathcal{C}(\mathbf{v}_\mathcal{C}) \sim_L A'_\mathcal{C}(\mathbf{v}_\mathcal{C})) \ \wedge \ (A_\mathcal{A}(\mathbf{v}_\mathcal{A}) \sim_L A'_\mathcal{A}(\mathbf{v}_\mathcal{A})) \ \wedge \ (\tau_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{A})$
(MergeSec-REF)	$(E \triangleq \texttt{when}\ G(\mathbf{v}, \mathbf{c})\ \texttt{then}\ S(\mathbf{v})\ \texttt{end}) \ \wedge \ (F \triangleq \texttt{when}\ H(\mathbf{v}, \mathbf{c})\ \texttt{then}\ S(\mathbf{v})\ \texttt{end}) \ \wedge \ (\tau \vdash \Gamma\{E\}\Gamma') \ \wedge \ (\tau \vdash \Gamma\{F\}\Gamma'')$ $\implies EF \triangleq (\texttt{when}\ G(\mathbf{v}, \mathbf{c}) \vee H(\mathbf{v}, \mathbf{c})\ \texttt{then}\ S(v)\ \texttt{end}) \ \wedge \ (\tau \vdash \Gamma \ \{EF\} \ \Gamma' \sqcup \Gamma'') \ \wedge (\Gamma' \sqcup \Gamma'' \sqsubseteq_{\mathsf{sec}} \Gamma)$
(NewEvtSec-REF)	$I(\mathbf{v}_\mathcal{A}) \ \wedge \ J(\mathbf{v}_\mathcal{A}, \mathbf{v}_\mathcal{C}) \ \wedge \ G_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{c}_\mathcal{C}) \ \wedge \ \Psi_\mathcal{C}(\mathbf{v}_\mathcal{C}, \mathbf{v}'_\mathcal{C}) \ \wedge \ (\Gamma_\mathcal{C} \vdash \mathbf{v}_\mathcal{C} : \tau_\mathcal{C}) \ \wedge \ (\Gamma'_\mathcal{C} \vdash \mathbf{v}'_\mathcal{C} : \tau'_\mathcal{C})$ $\implies J(\mathbf{v}_\mathcal{A}, \mathbf{v}'_\mathcal{C}) \ \wedge \ (V(\mathbf{v}'_\mathcal{C}) \in \mathbb{N}) \wedge (V(\mathbf{v}'_\mathcal{C}) < V(\mathbf{v}_\mathcal{C})) \ \wedge \ (\tau'_\mathcal{C} \sqsubseteq_{\mathsf{sec}} \tau_\mathcal{C})$

Table III
TIME-SENSITIVE SECURE REFINEMENT RULES FOR MODEL EVENTS.

threaded languages, and enforce the timing behaviour of a program to be independent of secrets by using padding techniques. Russo et. al [11] propose a method to track and close internal timing channels in multi-threaded programs by doing transformation. Through this channel, the information is leaked when the secrets affect the timing behaviour of a thread. Intuitively, the internal timing leaks are introduced by low assignment after high conditionals. To close this channel, a `fork` is introduced by the transformation when the branches depend on high data. Agat [12] studies the problem of detecting and removing timing leaks. A type system is proposed to prevent the time-sensitive information leakage, and a transformation is performed to remove the timing leaks from programs regarding to a bisimulation based semantic security condition. The programs satisfying the security condition do not leak any secret information directly, indirectly, via timing channels, or by termination behaviour to the external observers. By padding the branching commands, timing leaks is removed in the transformed program. Hendin and Sands [13] extend Agat's work for the treatment of an object oriented language. Barthe et. al [14] introduce a program transformation to prevent timing and termination leaks in a sequential object-oriented language via transaction mechanisms.

In addition to type-based treatments, there are also other attempts to deal with secure information flow control for programs. Rustan, Leino and Joshi [15] introduce a low level counter variable to record execution time for the purpose of reasoning about covert flows involving timing behaviour. This enforcement is very strong and over pessimistic since the counter variable need to be updated after each command and all high branching commands will be considered introducing implicit flows. Köpf and Basin [16] study a parameterised notion of information flow security for the analysis of timing side channels in synchronous hardware circuits. The synchronous system is modelled as an automaton with output. The flow security notion ($R_I R_O$-secure) is based on the idea that the observational equivalence of states is a partial equivalence relation (PER), *i.e.,* the system can be formalised as the PER model of secure information flow [17]. The quantity of timing sensitive information leakage of the system is approximated by counting distinguishable behaviours. Hammer and Snelting [18] present an approach for information flow control in program analysis based on program dependence graphs (PDG). Based on [18], [19] extends the PDG-based flow analysis by incorporating refinement techniques via path conditions to improve the precision of the flow control. Such PDG-based information flow control is more precise but more expensive than type-based approaches and there is no timing flow treatment.

In a software development cycle, it is important to ensure security from the very beginning at the specification level. However, there has been no treatment for flow analysis and control in abstract specification languages and security preservation under refinement in the above works. There have been a number of works addressing flow analysis in Event-B, but without considering timing leakage [20], [21], [1]. Mu [1] presents a type system for flow control in Event-B models. Iliasov [20] handles the interference between ordered events introduced by a set of conditions formulated on a machine for flow control. Bendisposto *et. al* [21] derive a flow graph structure from an Event-B model specification to

manage the information about dependence and independence of events for flow analysis. Comparing with these works, we have presented a framework to reason about *time* sensitive flow control in *specification* language and relevant relations of the stepwise *refinement* transformations.

VII. Conclusions

This paper investigates the problem of preserving time-sensitive information flow security in timed Event-B specification models and in the process of refinement. We have introduced a framework for reasoning about the secure flow property in Event-B and under refinement. We extend the Event-B system with timing constructs, and present a security type system for the timed Event-B model with time-sensitive information flow control. We then extend the refinement rules to preserve secure information flow properties under abstraction refinement.

We have presented a general framework for formally reasoning about preserving time-sensitive flow secrecy properties in a specification language and the process of abstraction refinements. We believe this is a promising starting point for a comprehensive formal treatment of information flow security in specification languages and stepwise secure refinements. There are a large number of programming languages, which can be classified in terms of the paradigms they support, such as imperative, object-oriented and real-time languages. For future work, we propose to develop combined theories for formal treatment of information security issues in different paradigms, and to explore the relationships between them.

References

[1] C. Mu, "On information flow control in event-b and refinement," in *Seventh International Symposium on Theoretical Aspects of Software Engineering, TASE 2013, 1-3 July 2013, Birmingham, UK*, 2013, pp. 225–232.

[2] J. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in Event-B," *STTT*, vol. 12, no. 6, pp. 447–466, 2010.

[3] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," *Commun. ACM*, vol. 20, no. 7, pp. 504–513, 1977.

[4] J. Goguen and J. Meseguer, "Security policies and security models," in *S & P*, 1982, pp. 11–20.

[5] S. Hunt and D. Sands, "On flow-sensitive security types," in *POPL*. ACM Press, January 2006, pp. 79–90.

[6] ——, "From exponential to polynomial-time security typing via principal types," in *ESOP*, 2011, pp. 297–316.

[7] D. M. Volpano and G. Smith, "Eliminating covert flows with minimum typings," in *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, 1997, pp. 156–169.

[8] G. Smith and D. M. Volpano, "Secure information flow in a multi-threaded imperative language," in *POPL '98, Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, CA, USA, January 19-21, 1998*, 1998, pp. 355–364.

[9] D. M. Volpano and G. Smith, "Probabilistic noninterference in a concurrent language," in *Proceedings of the 11th IEEE Computer Security Foundations Workshop, Rockport, Massachusetts, USA, June 9-11, 1998*, 1998, pp. 34–43.

[10] A. Sabelfeld and D. Sands, "Probabilistic noninterference for multi-threaded programs," in *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00, Cambridge, England, UK, July 3-5, 2000*, 2000, pp. 200–214.

[11] A. Russo, J. Hughes, D. A. Naumann, and A. Sabelfeld, "Closing internal timing channels by transformation," in *Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues, 11th Asian Computing Science Conference, Tokyo, Japan, December 6-8, 2006, Revised Selected Papers*, 2006, pp. 120–135.

[12] J. Agat, "Transforming out timing leaks," in *POPL 2000, Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Boston, Massachusetts, USA, January 19-21, 2000*, 2000, pp. 40–53.

[13] D. Hedin and D. Sands, "Timing aware information flow security for a javacard-like bytecode," *Electr. Notes Theor. Comput. Sci.*, vol. 141, no. 1, pp. 163–182, 2005.

[14] G. Barthe, T. Rezk, and M. Warnier, "Preventing timing leaks through transactional branching instructions," *Electr. Notes Theor. Comput. Sci.*, vol. 153, no. 2, pp. 33–55, 2006.

[15] K. R. M. Leino and R. Joshi, "A semantic approach to secure information flow," in *Mathematics of Program Construction, MPC'98, Marstrand, Sweden, June 15-17, 1998, Proceedings*, 1998, pp. 254–271.

[16] B. Köpf and D. A. Basin, "Timing-sensitive information flow analysis for synchronous systems," in *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, 2006, pp. 243–262.

[17] A. Sabelfeld and D. Sands, "A per model of secure information flow in sequential programs," *Higher-Order and Symbolic Computation*, vol. 14, no. 1, pp. 59–91, 2001.

[18] C. Hammer and G. Snelting, "Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs," *Int. J. Inf. Sec.*, vol. 8, no. 6, pp. 399–422, 2009.

[19] M. Taghdiri, G. Snelting, and C. Sinz, "Information flow analysis via path condition refinement," in *FAST*, 2010, pp. 65–79.

[20] A. Iliasov, "On event-b and control flow," School of Computing Science, Newcastle University, Tech. Rep. CS-TR-1159, 2009.

[21] J. Bendisposto and M. Leuschel, "Automatic flow analysis for event-b," in *FASE*, 2011, pp. 50–64.