# Wikiprint Book

**Title: Modeling Language**

**Subject: ThirdEye - logic-based-modelling**

**Version: 15**

**Date: 09/15/14 11:40:09**

## Table of Contents

## Modeling Language

The objective of the modeling process is to enable users to analyze data at higher levels of abstraction and thus at a level closer to their understanding of system operation. The modeling language is at the core of the semantic analysis process and provides semantically relevant abstractions to capture relationships like causality, ordering, concurrency, exclusions, combinations and dependencies between high-level system operations. Specifically, it provides the following features:

1. Enables analysis over multi-type, multi-variate, timestamped data.
2. Expresses a wide variety of "interesting" relationships relevant to networked and distributed systems.
3. Enables analysis over higher-level abstractions.
4. Enables composing abstractions into higher-level abstractions.

The following ideas are central to modeling:

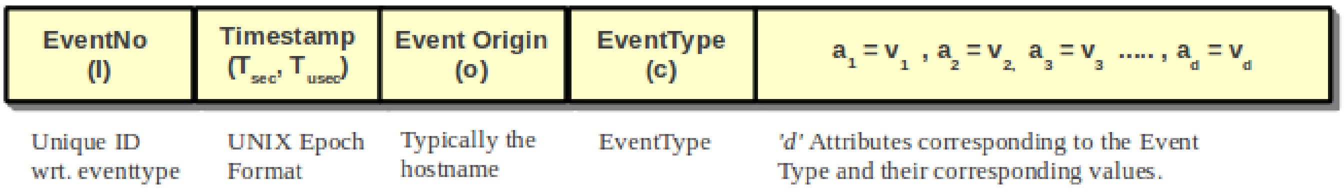
- Events - Uniform representation of raw data.
- Behavior - Fundamental abstractions for analysis.
- Relationships - Enable semantic level analysis.
- Behavior Models - Assertions about overall behavior of a system or process.

---

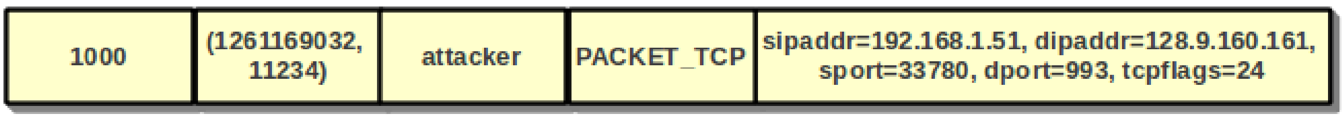
### *Events*

An ***event*** is a uniform representation of low-level of multi-type, multi-variate, timestamped data. Data normalization plugins are responsible for converting low-level data of any type into uniform events. This normalization allows the framework to operate independent of the type of raw data.

Events are identified by an event number and an event type, are timestamped and carry event details as attribute-value pairs. A generic representation of an event is shown below.


EventNo (I)
Timestamp (T_sec, T_usec)
Event Origin (o)
EventType (c)
a_1 = v_1 , a_2 = v_2, a_3 = v_3 ..... , a_d = v_d
Unique ID wrt. eventtype
UNIX Epoch Format
Typically the hostname
EventType
'd' Attributes corresponding to the Event Type and their corresponding values.

(a) Event Format


1000
(1261169032, 11234)
attacker
PACKET_TCP
sipaddr=192.168.1.51, dipaddr=128.9.160.161, sport=33780, dport=993, tcpflags=24

(b) Example PACKET_TCP event (w/ partial attributes shown)

Events are stored in a database. Every event type has a separate table of the corresponding name; the columns of the tables correspond to the event attributes and each row describes an event of that event type. For example, the following is a sample database containing events of type **PACKET_TCP**.

```
sqlite> select * from PACKET_TCP;
eventno     eventtype   timestamp   timestampusec  origin      sipaddr       dipaddr        sport
----------  ----------  ----------  -------------  ----------  ------------  -------------  ----------
1           PACKET_TCP  1267192686  584044         localhost   192.168.3.65  188.72.243.72  1032
2           PACKET_TCP  1267192686  693493         localhost   188.72.243.7  192.168.3.65   80
...
<output snipped ... another 1100 events displayed>
```

Note that the framework itself does not require any specific event types or event attributes to be defined. The plugins can choose to define any event type and any attributes for events.

---

### ***Behavior*: Fundamental modeling abstractions**

The key differentiator of the semantic approach is the introduction of the notion of behavior - a sequence or a group of related events - as a fundamental abstraction for analysis and reasoning over data. Behaviors can be composed using a rich set of operators to express semantically-relevant relationships. Behaviors capture system semantics and operators allow expressing sophisticated relationships between behaviors. This enables users to reason over data at higher levels and closer to their understanding of system operation.

For example, consider a network packet trace contains TCP packets.

- Each individual packet is normalized **PACKET_TCP** event with each event attribute corresponding to a field in the packet.
- A grouping of related **PACKET_TCP** events - related due to dependencies between their attributes and due to causal relationships - can be called a **TCP_FLOW** behavior.
- A higher-level behavior like **CONCURRENT_TCP_FLOWS** can now be formed by relating **TCP_FLOW** behavior.
- and so on...

### *Behavior Models (or Models)*

A ***Model*** is a high-level assertion about the overall behavior of the system. Additionally, such models can be composed to form higher-level models.

For example, see the example of the TCP connection setup and TCP connection teardown models from the knowledge base. Individually, each model captures the possible behaviors of TCP during connection start and termination. A new model for TCP flow - capturing a complete TCP Flow - can now be created by simply composing the above two models.

### *Relationships*

The language allows expressing the following kinds of relationships to capture the core characteristics of networked and distributed systems:

| No. | Relationship | Example |
|---|---|---|
| 1. | Causal relationships between behaviors. | A file being opened only if a user is authorized |
| 2. | Partial or total ordering | In-order or out-of-order arrival of packets |
| 3. | Dynamic changes over time | Traffic between client and server drops after an attack on the server |
| 4. | Concurrency of operations | Simultaneous web client sessions |
| 5. | Multiple possible behaviors | A polymorphic worm behavior may vary on each execution |
| 6. | Synchronous or asynchronous operations | Some operations need to complete within a specific time whereas others need not |
| 7. | Value dependencies between operations | A TCP flow is valid only if the attribute?values contained in the individual packets are related to each other |
| 8. | Invariant operations | Some operations may always hold true |
| 9. | Eventual operations | Some operations happen in the course of time |
| 10. | Exclusivity and combination of operations | Two or more operations are always expected to happen but not the third one |

## What cannot be modeled in the current release?

The current modeling language abstractions cannot be used for situations like:

- Modeling probabilistic behavior.
- Modeling packet distributions.
- Modeling complex dependencies between event attributes like algebraic dependencies.
- Recursive behaviors.

These abstractions and more are in the pipeline to be included in the future releases.

---