Monitoring Violations & Threats of Security & Dependability: The SERENITY approach

Prof George Spanoudakis

School of Informatics

City University London

Email: G.Spanoudakis@soi.city.ac.uk

CITY UNIVERSITY LONDON

Lecture objectives

- To introduce the SERENITY approach to dynamic assembly and configuration of S&D solutions and the need for monitoring security and dependability properties at runtime
- To explain the SERENITY approach to monitoring and introduce the SERENITY runtime monitoring framework, called EVEREST
- To provide examples of using EVEREST for runtime monitoring of S&D properties
- To explain advanced features of EVEREST, namely the event diagnosis and the threat detection and reaction mechanisms

Outline

Part I: Overview of the SERENITY framework

- Overview of SERENITY
- S&D patterns
- An example
- Need for monitoring
- The SERENITY infrastructure

Part II: The SERENITY monitoring infrastructure

- The SERENITY monitoring approach
- Monitoring lifecycle
- Monitoring infrastructure

Part III: Specification of monitorable S&D properties

- Specification of monitoring rules
- Examples of monitoring rules

Part IV: Advanced Capabilities

- Monitoring process
- Diagnosis
- Threat detection

Part V: Reaction

- Reaction to monitoring results

Conclusions, Main resources and references

Part I:
Overview of the SERENITY framework

Overview of SERENITY

Aims:

Dynamic

- selection
- (re-) configuration
- integration, and
- deployment

of components that can realise Security and Dependability (S&D) solutions in applications, driven by S&D patterns

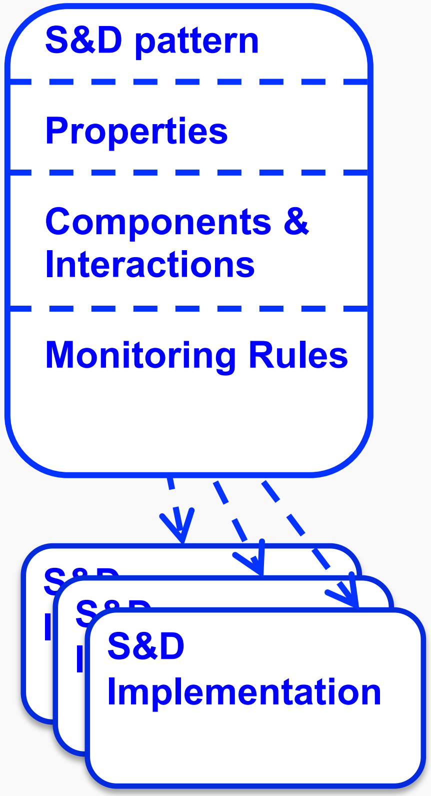
Motivation:

Applications

- Have continually changing S&D requirements
- Often need to operate in changing operational environments and contents
- Interact with dynamically assembled distributed components

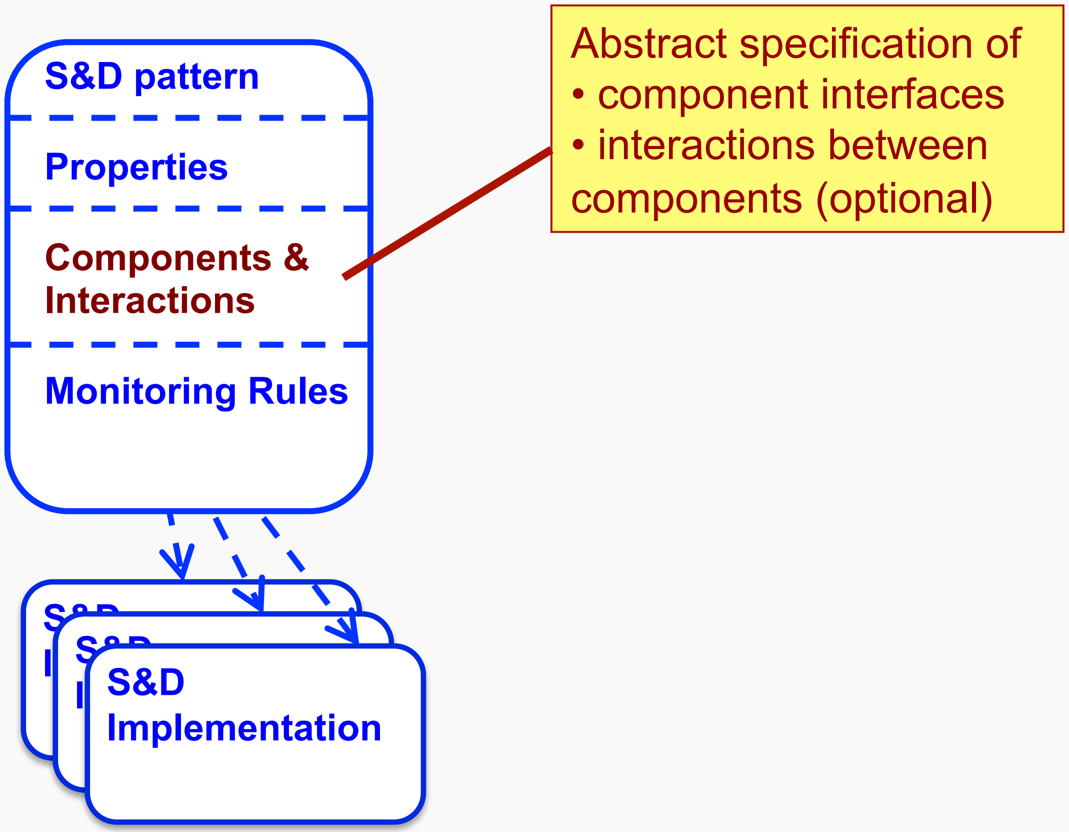
S&D patterns

- Provide an abstract specification of solutions that can be deployed in a system to provide S&D properties and link this specification to alternative concrete implementations



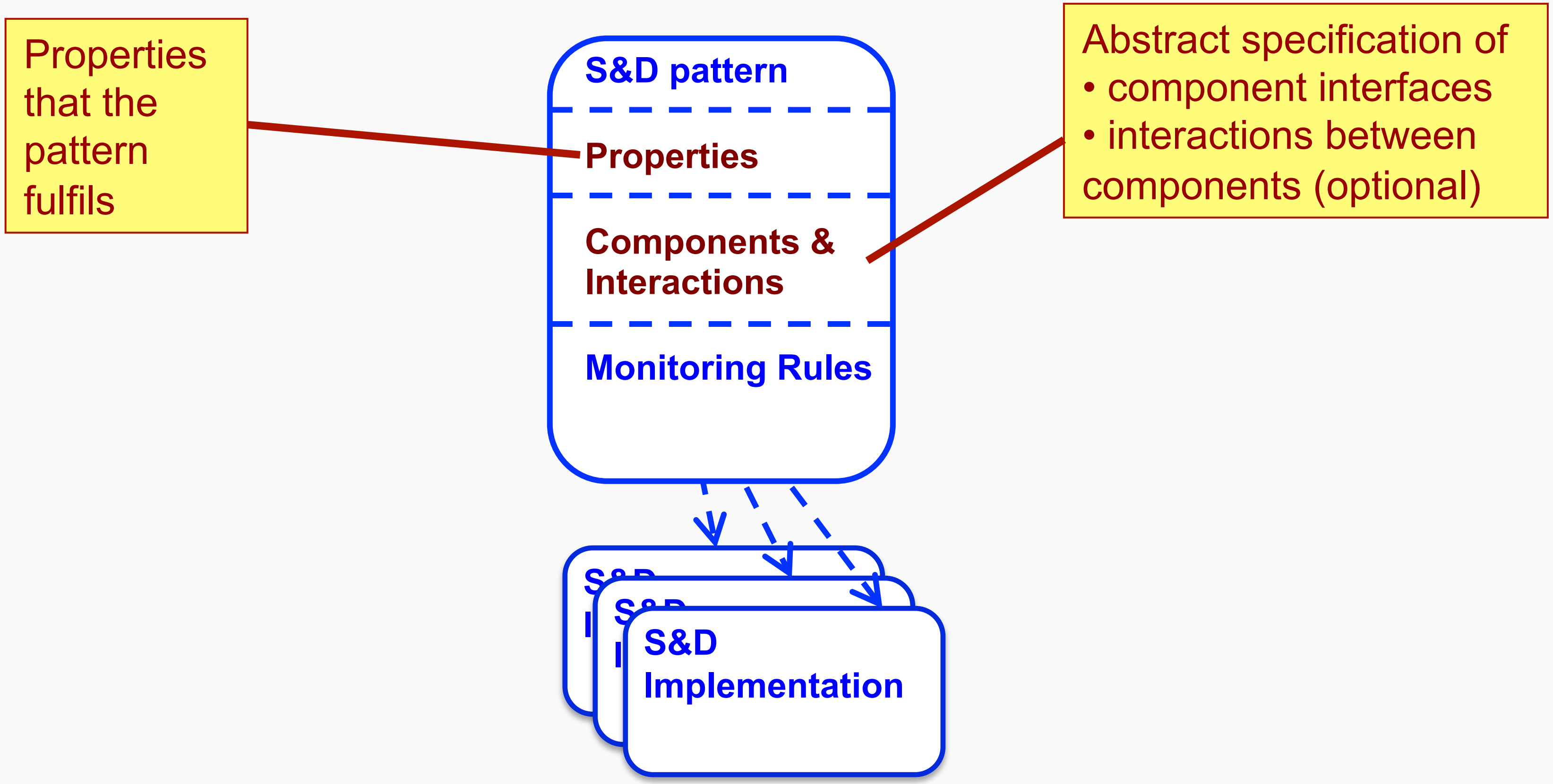
S&D patterns

- Provide an abstract specification of solutions that can be deployed in a system to provide S&D properties and link this specification to alternative concrete implementations



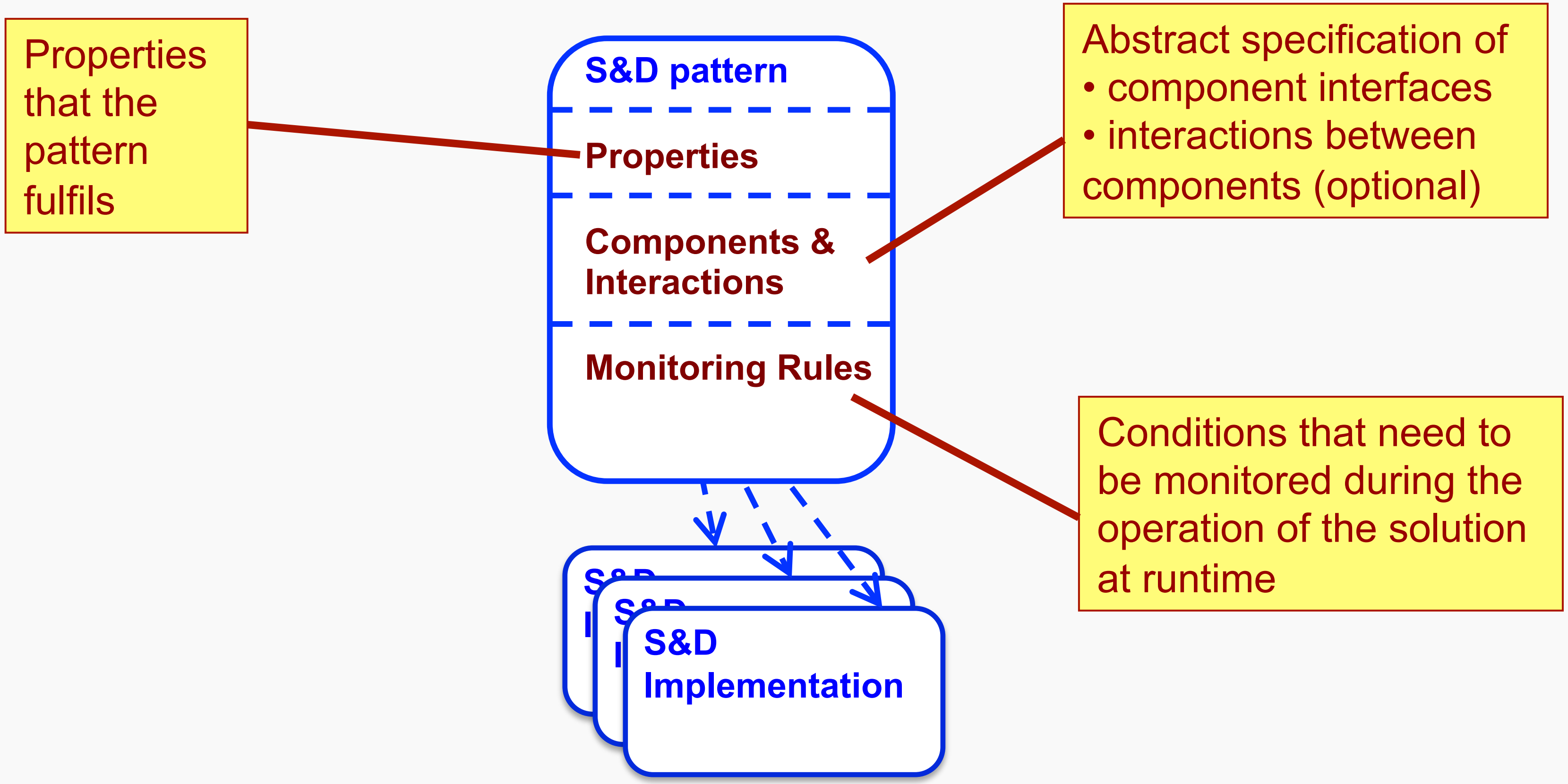
S&D patterns

- Provide an abstract specification of solutions that can be deployed in a system to provide S&D properties and link this specification to alternative concrete implementations



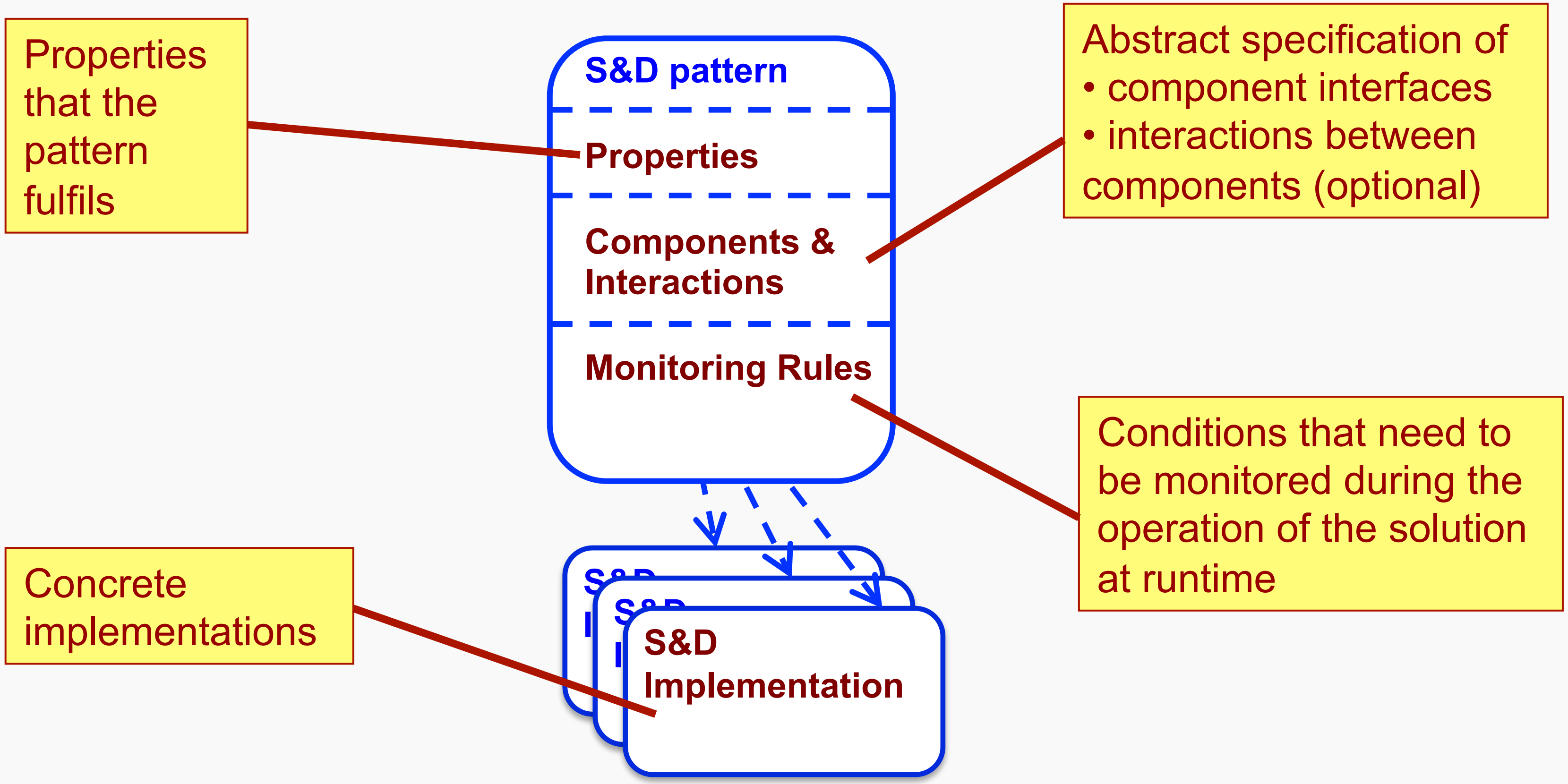
S&D patterns

- Provide an abstract specification of solutions that can be deployed in a system to provide S&D properties and link this specification to alternative concrete implementations



S&D patterns

- Provide an abstract specification of solutions that can be deployed in a system to provide S&D properties and link this specification to alternative concrete implementations

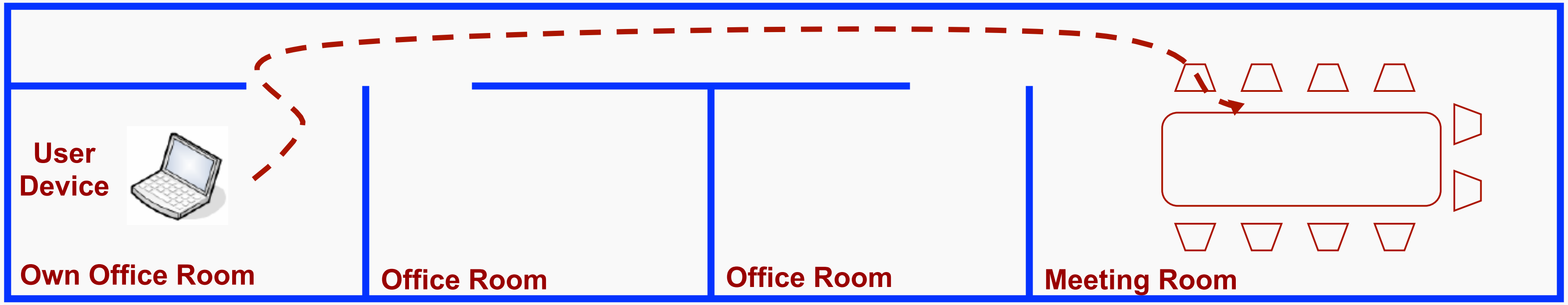


An example: Location based access control

- Access control system providing access to enterprise resources (e.g. printers, Internet access etc) from mobile user devices (PDAs, laptops) (based on [11])
- When a user requests access to a resource, the system may provide it depending on:
  - the credentials of the user,
  - the ability to authenticate the device from which access is requested, and
  - the location of the device

An example: Location based access control

- Access control system providing access to enterprise resources (e.g. printers, Internet access etc) from mobile user devices (PDAs, laptops) (based on [11])
- When a user requests access to a resource, the system may provide it depending on:
  - the credentials of the user,
  - the ability to authenticate the device from which access is requested, and
  - the location of the device



Access to
- Intranet, Internet
- Room's printer
- Printers in common areas

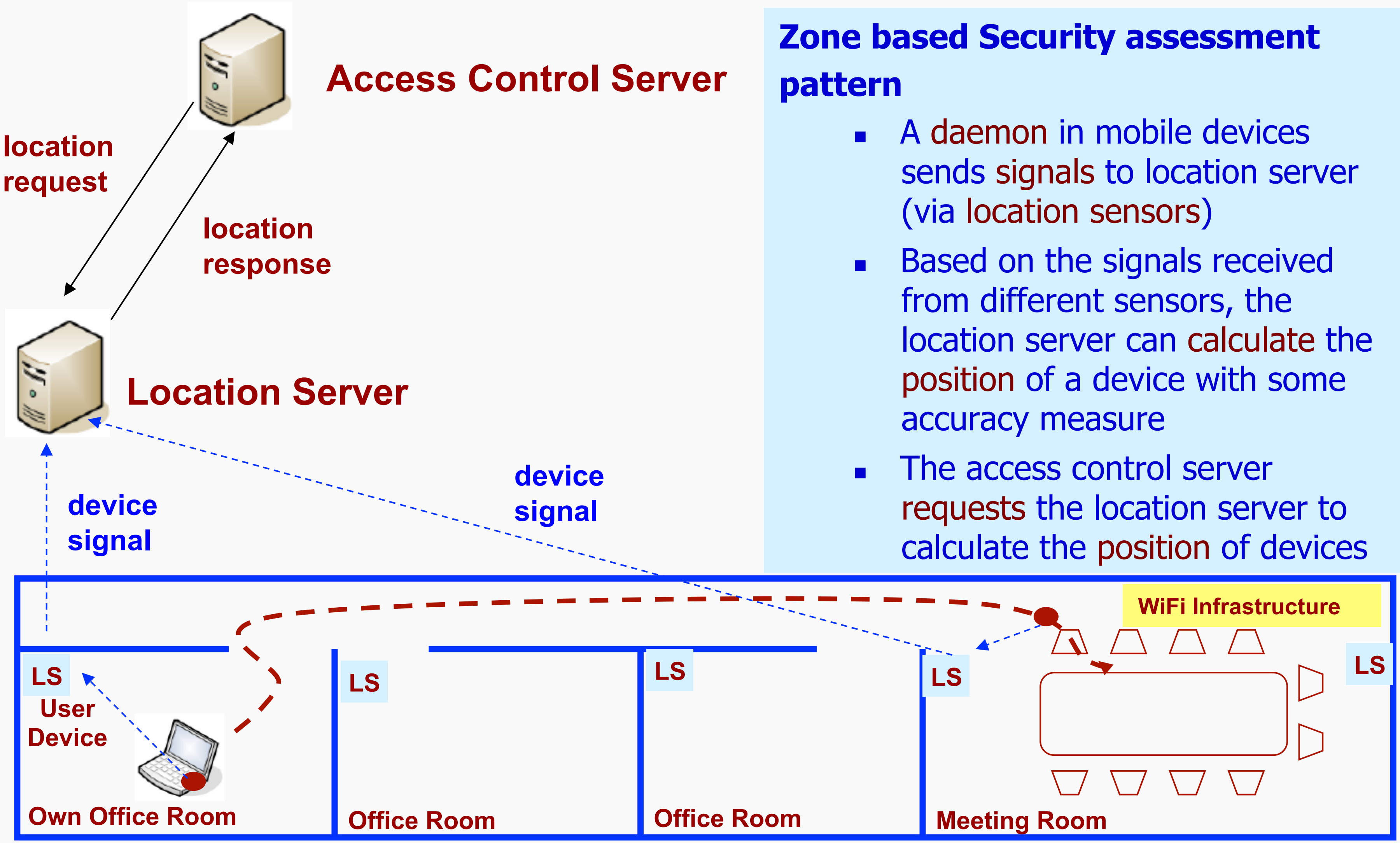
No access to
- printers in other rooms

Provided that both the mobile device and its user have been authenticated

Access to
- Room's printer
- Internet

No access to
- printers in other rooms
- Intranet

An example: device position calculation



Zone based Security assessment pattern

- A daemon in mobile devices sends signals to location server (via location sensors)
- Based on the signals received from different sensors, the location server can calculate the position of a device with some accuracy measure
- The access control server requests the location server to calculate the position of devices

An example: Device location pattern (DLP)

Device Location Pattern

Properties

Location Server: has TPM-based identity

Components

Location Server

locationRequest(devID:ID,loc: Location, acc: Float)

signal(devID: ID)

Monitoring Rules

Need for monitoring

Runtime monitoring of S&D solutions is required in order to

- Check preconditions and invariants required for the correct operation of the solutions
- Verify dynamically that an S&D solution operates according to its specification in all circumstances (static verification and testing cannot provide a full guarantee for this)
- Predict possible violations of conditions and take (if possible) pre-emptive actions

DLP: some monitoring conditions

- Availability of the location server:

  Whenever the access control server makes a request for the location of a device to the location server it must receive a response (or otherwise no access decisions can be made or access will be continually over-restricted)

- Liveness of signal daemons in mobile devices:

  Every device that is known to the control server should be sending signals to the location server periodically and the maximum period of not receiving a signal should not be less than *m* time units (or otherwise it won't be possible to calculate the position of the device)

- Accuracy of location information:

  The accuracy of the device location information that is provided by the location server must always (on average) exceed a certain accuracy threshold

Monitoring rules of DLP pattern

Device Location Pattern

Properties

Location Server: has TPM-based identity

Components

Location Server

locationRequest(devID:ID,loc: Location, acc: Float)

signal(devID: ID)

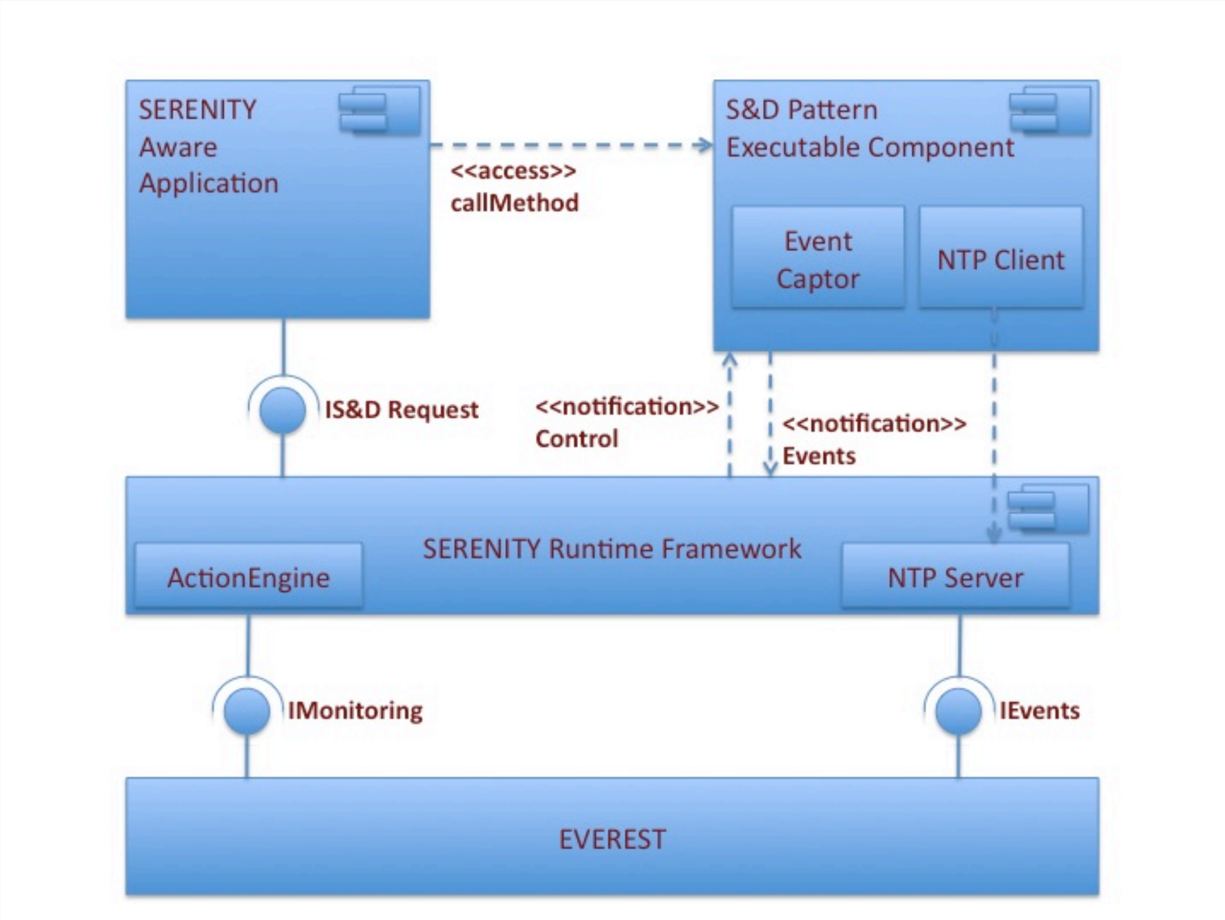
Monitoring Rules

<availability of location server>, notify SRF

<liveness of mobile device daemons>, notify application

<accuracy of location information>, notify SRF

SERENITY Infrastructure



SERENITY Runtime Framework

- Activates patterns and their executable implementations
- Sends monitoring rules to EVEREST
- Receives events from captors of pattern implementations and forwards them to EVEREST
- Polls EVEREST for results and executes actions according to them

EVEREST

- Is available as a service to the SERENITY runtime framework (SRF)
- Receives specifications of the rules to be monitored and runtime events from the SRF
- Performs the checking
- Can be polled for monitoring results

Part II:
The SERENITY monitoring infrastructure

Runtime monitoring

3 alternatives

- The application performs the checks itself
- The checks are performed by an external entity
- The checks are performed by both the application and an external entity

Runtime monitoring

3 alternatives

- The application performs the checks itself

  Requires **extra programming** effort, **expensive** to change when the system is in operation and needs to deploy a new S&D solution, some checks need to be applied on the deployed S&D solution which the application has no control of

- The checks are performed by an external entity

- The checks are performed by both the application and an external entity

Runtime monitoring

3 alternatives

- The application performs the checks itself

  Requires **extra programming** effort, **expensive** to change when the system is in operation and needs to deploy a new S&D solution, some checks need to be applied on the deployed S&D solution which the application has no control of

- The checks are performed by an external entity

  Requires **monitoring specifications**, **more flexible** when operational **environments change** and S&D **solutions change**, **can be applied** to **external collaborators,** **less efficient** than application based testing

- The checks are performed by both the application and an external entity

Runtime monitoring

3 alternatives

- The application performs the checks itself

  Requires **extra programming** effort, **expensive** to change when the system is in operation and needs to deploy a new S&D solution, some checks need to be applied on the deployed S&D solution which the application has no control of

- The checks are performed by an external entity

  Requires **monitoring specifications**, **more flexible** when operational **environments change** and S&D **solutions change**, **can be applied** to **external collaborators,** **less efficient** than application based testing

- The checks are performed by both the application and an external entity

  Increased fault tolerance (two independent implementations of the same check), more **expensive** and **less flexible** option, **necessary** in certain circumstances

Runtime monitoring: The SERENITY approach

3 alternatives

- The application performs the checks itself

  Requires **extra programming** effort, **expensive** to change when the system is in operation and needs to deploy a new S&D solution, some checks need to be applied on the deployed S&D solution which the application has no control of

- The checks are performed by an external entity

  Requires **monitoring specifications**, **more flexible** when operational **environments change** and S&D **solutions change**, **can be applied** to **external collaborators,** **less efficient** than application based testing

- The checks are performed by both the application and an external entity

  Increased fault tolerance (two independent implementations of the same check), more **expensive** and **less flexible** option, **necessary** in certain circumstances

Monitoring life cycle

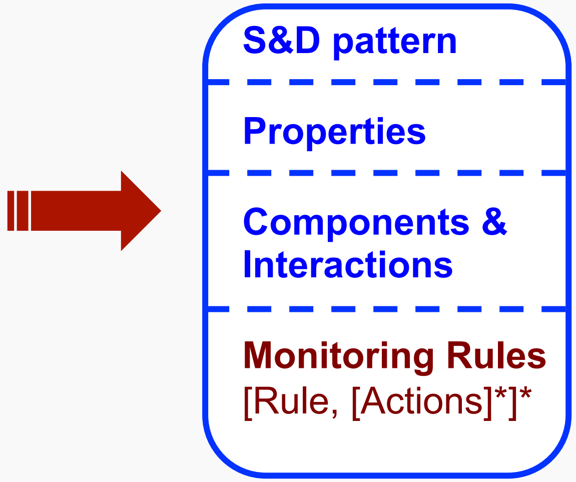
Monitoring life cycle

Development of S&D solutions

Monitoring life cycle

Development of S&D solutions

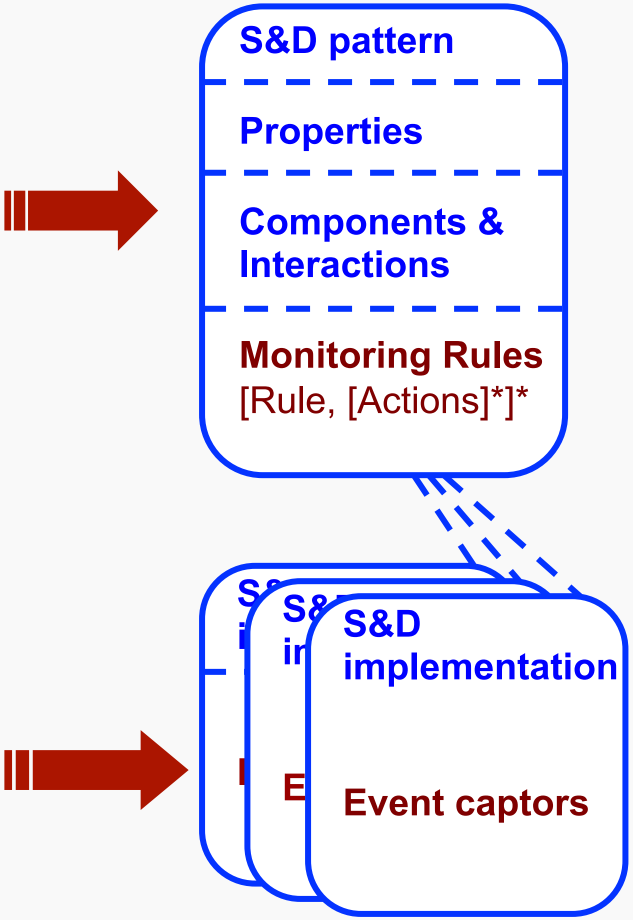
- Specify the conditions that need to be monitored at runtime and the actions that need to be taken when the conditions are violated within S&D patterns



Monitoring life cycle

Development of S&D solutions

- Specify the conditions that need to be monitored at runtime and the actions that need to be taken when the conditions are violated within S&D patterns



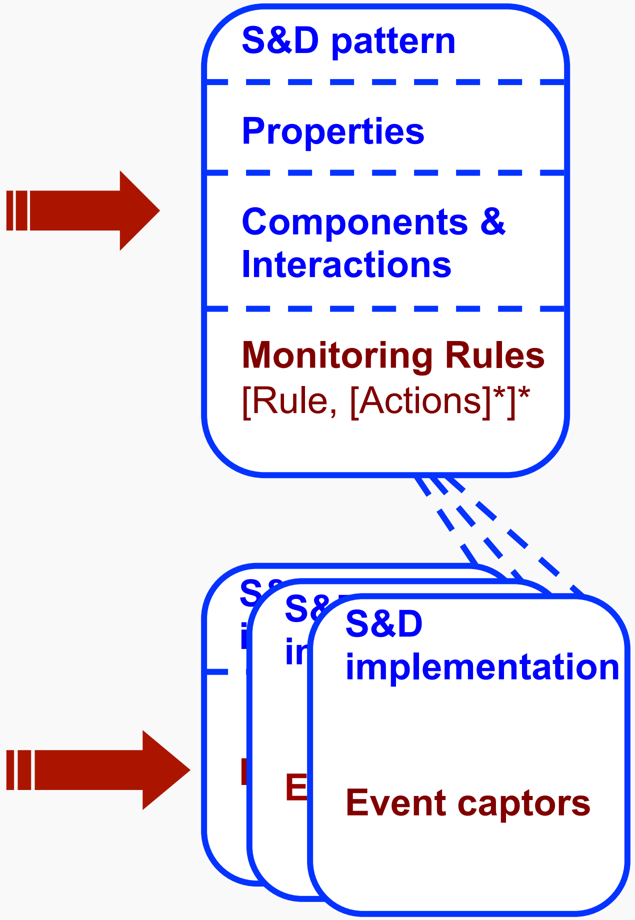
- Provide implementations of patterns (aka S&D solutions) incorporating captors that can provide the events required to monitor the conditions of the pattern

Monitoring life cycle

Development of S&D solutions

- Specify the conditions that need to be monitored at runtime and the actions that need to be taken when the conditions are violated within S&D patterns

- Provide implementations of patterns (aka S&D solutions) incorporating captors that can provide the events required to monitor the conditions of the pattern



At runtime

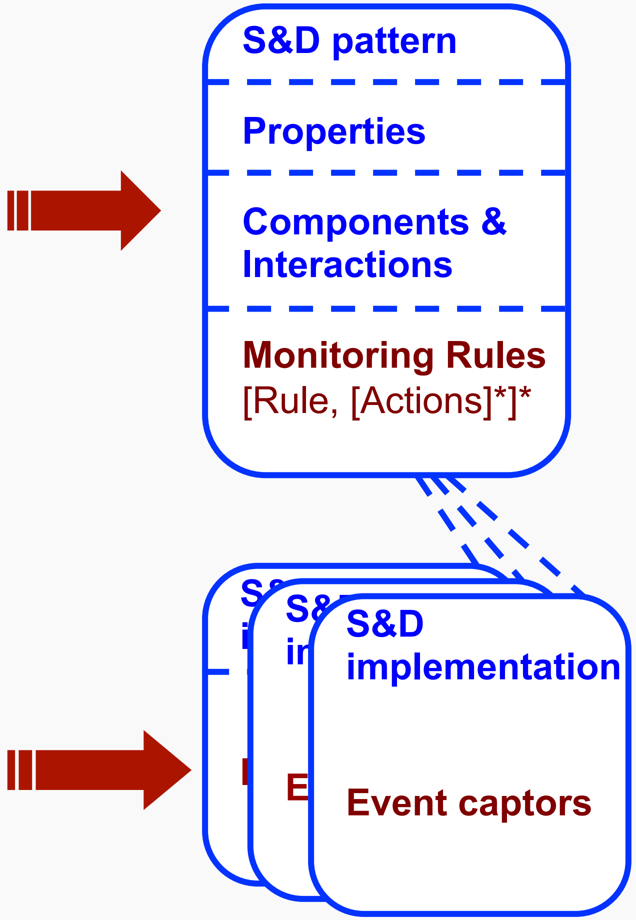
When an S&D pattern is selected:

- Start the process of checking its monitoring rules
- Activate the relevant S&D implementation and its captors

Monitoring life cycle

Development of S&D solutions

- Specify the conditions that need to be monitored at runtime and the actions that need to be taken when the conditions are violated within S&D patterns
- Provide implementations of patterns (aka S&D solutions) incorporating captors that can provide the events required to monitor the conditions of the pattern



At runtime

When an S&D pattern is selected:

- Start the process of checking its monitoring rules
- Activate the relevant S&D implementation and its captors

When a monitoring rule is violated:

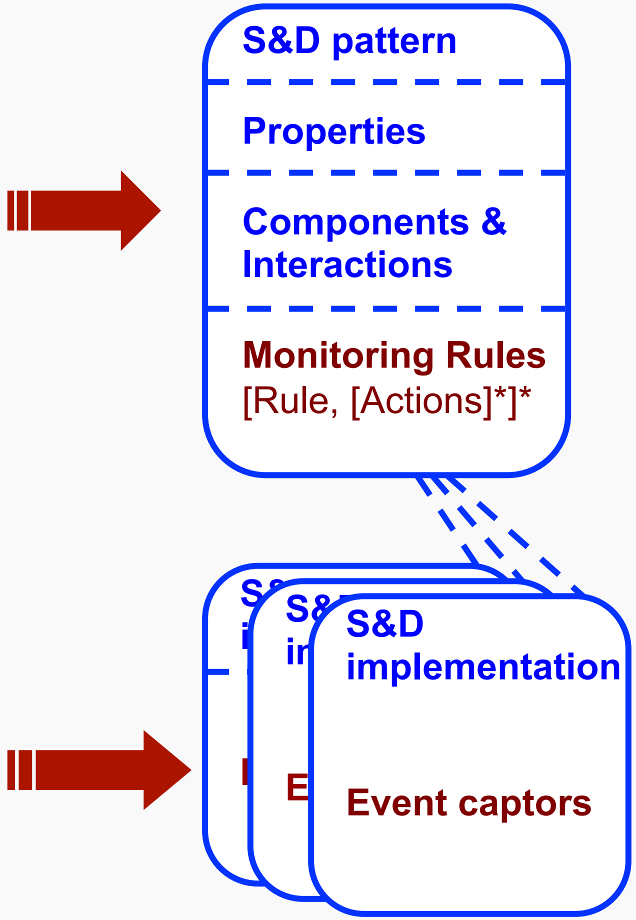
- Execute the action(s) specified for it (if any)

Monitoring life cycle

Development of S&D solutions

- Specify the conditions that need to be monitored at runtime and the actions that need to be taken when the conditions are violated within S&D patterns

- Provide implementations of patterns (aka S&D solutions) incorporating captors that can provide the events required to monitor the conditions of the pattern



At runtime

When an S&D pattern is selected:

- Start the process of checking its monitoring rules
- Activate the relevant S&D implementation and its captors

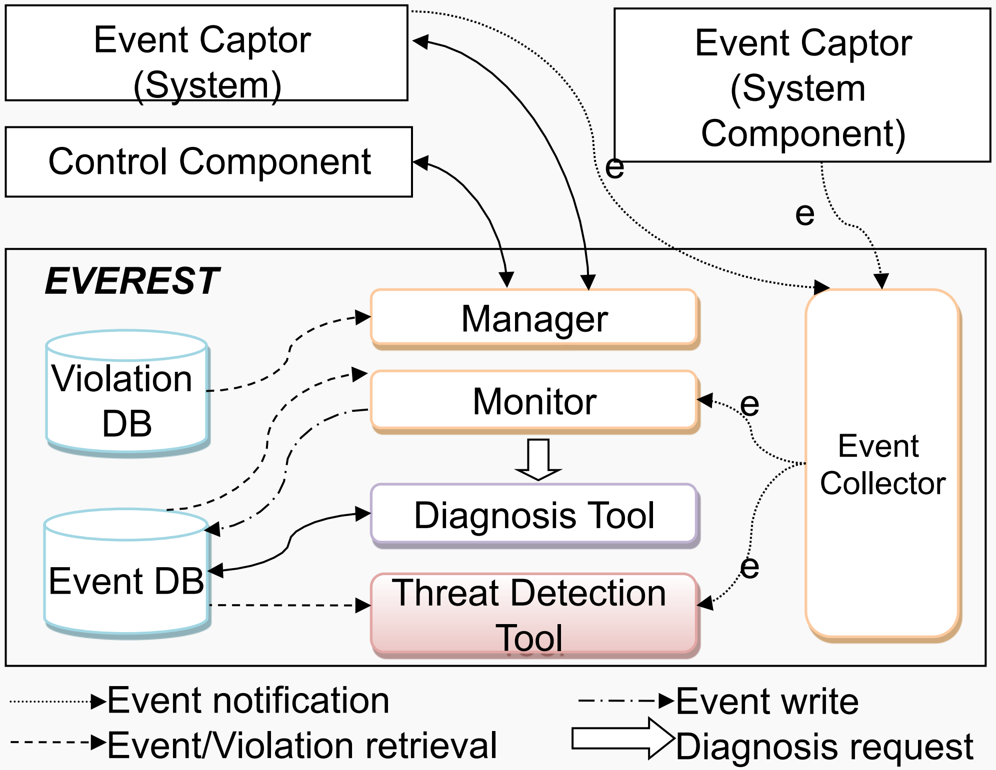
When a monitoring rule is violated:

- Execute the action(s) specified for it (if any)

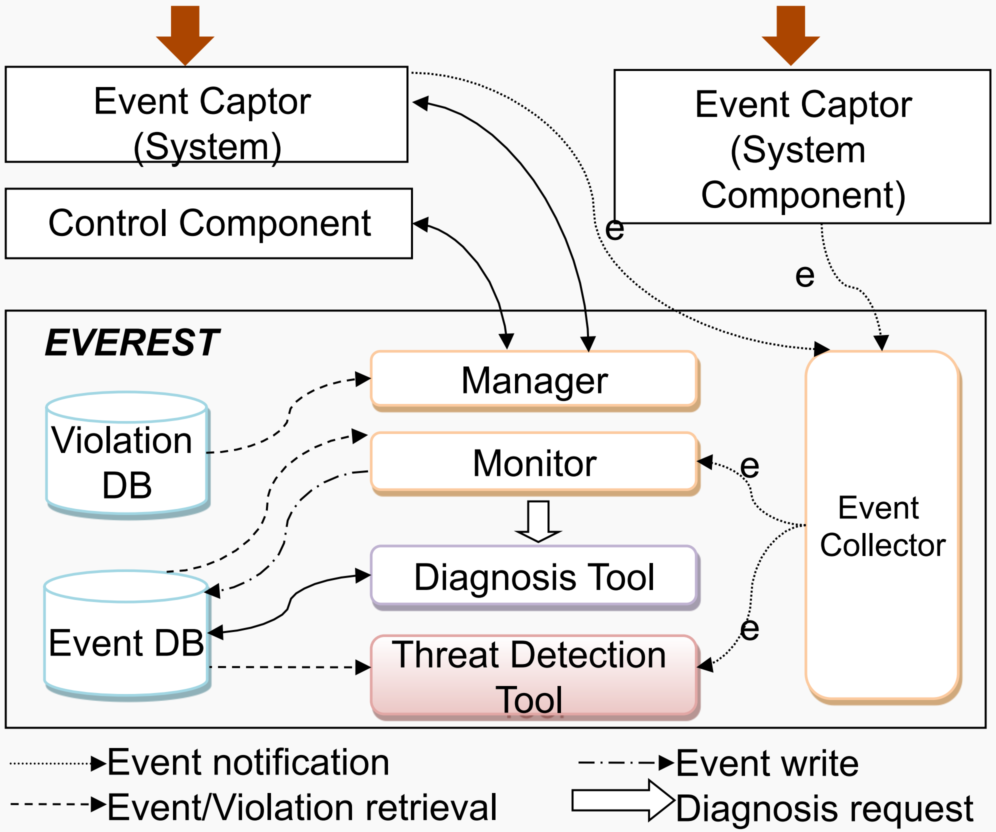
When an S&D pattern is deactivated:

- Stop the process of checking its monitoring rules
- Deactivate the relevant S&D implementation and its captors

EVEnt REaSoning Toolkit (EVEREST)

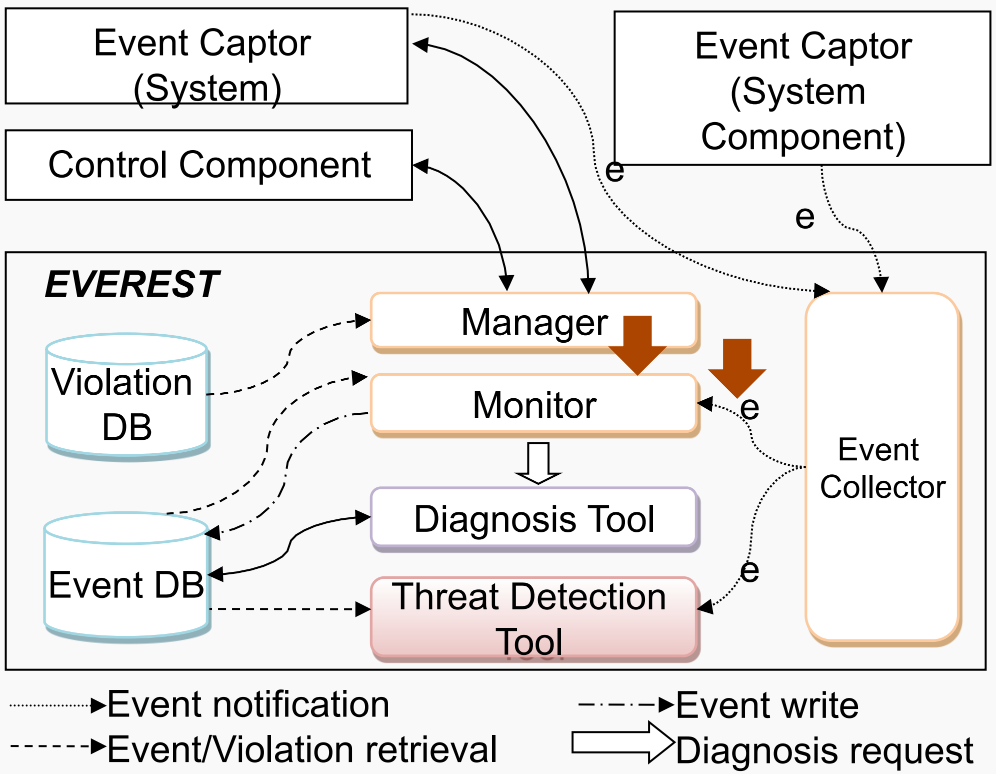


EVEnt REaSoning Toolkit (EVEREST)



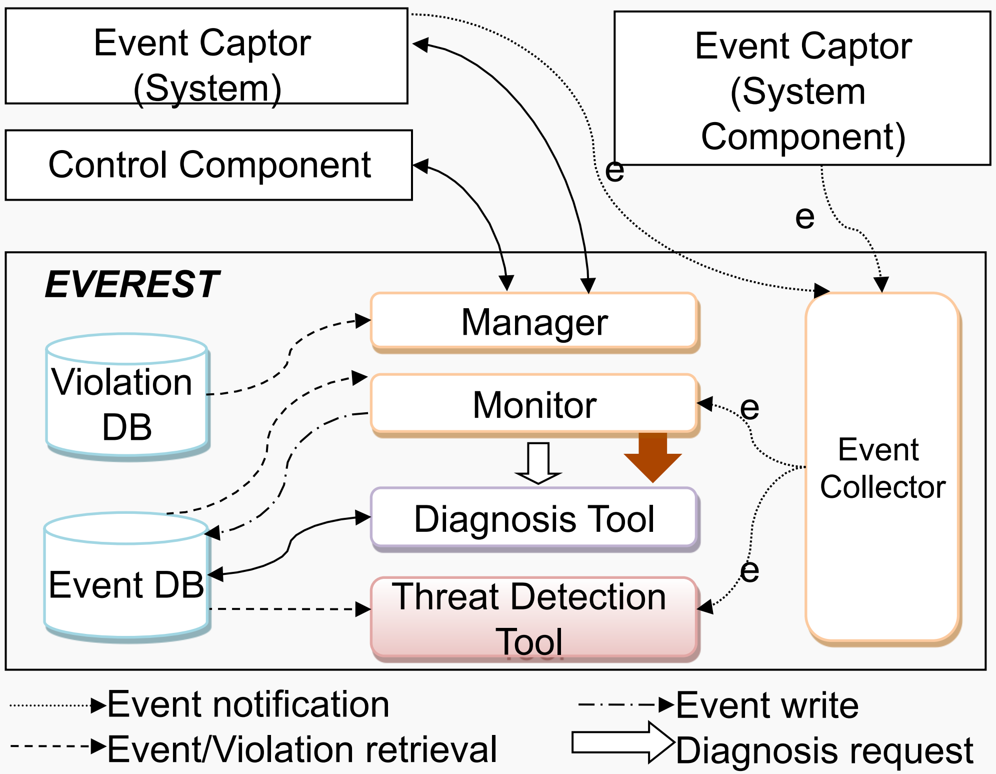
- Captures **events** through **event captors** associated with systems and their components

EVEnt REaSoning Toolkit (EVEREST)



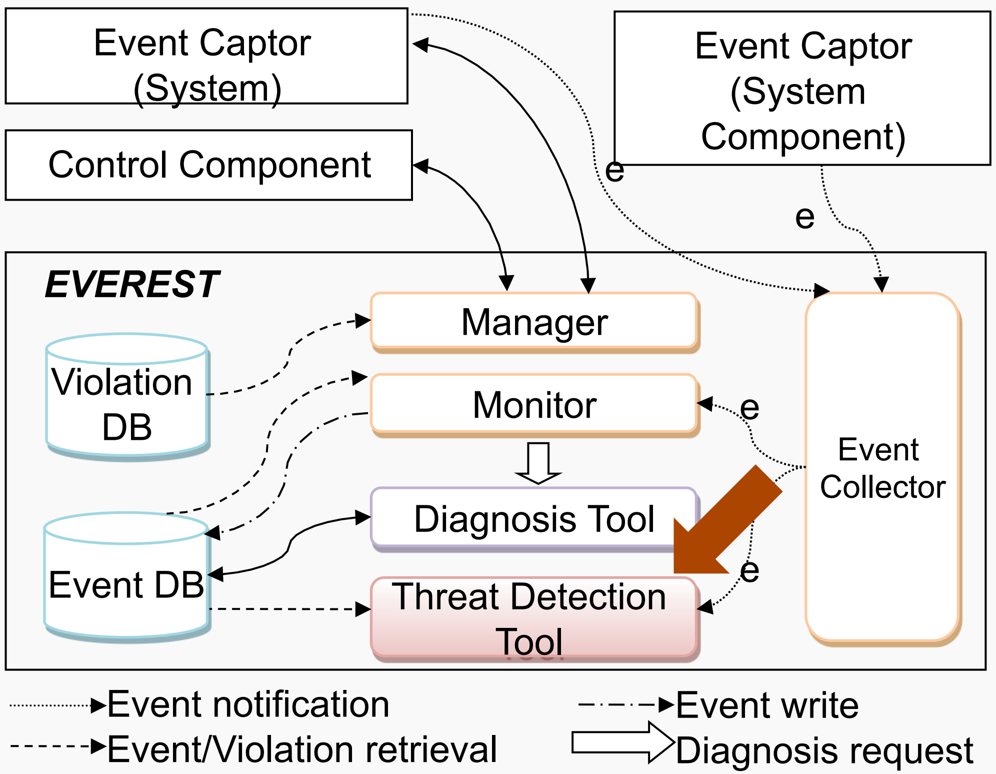
- Captures events through event captors associated with systems and their components
- Checks whether captured events (and events deduced from them) satisfy specific **S&D properties** expressed as **monitoring rules** (core **monitor**)

EVEnt REaSoning Toolkit (EVEREST)



- Captures events through event captors associated with systems and their components
- Checks whether captured events (and events deduced from them) satisfy specific S&D properties expressed as monitoring rules (core monitor)
- Assesses **event genuineness** by attempting to derive **explanations** of captured events (**diagnosis tool**)

EVEnt REaSoning Toolkit (EVEREST)



- Captures events through event captors associated with systems and their components
- Checks whether captured events (and events deduced from them) satisfy specific S&D properties expressed as monitoring rules (core monitor)
- Assesses event genuineness by attempting to derive explanations of captured events (diagnosis tool)
- Predicts **potential violations** of monitoring rules based on historical data (**threat detection tool – TDT**)

Part III:

Specification of monitorable S&D properties

Specification of monitoring rules (1)

- Monitoring rules: express the properties/requirements that need to be monitored

- General form

$$B_{t1} \Rightarrow H_{t2} \text{ (if } B_{t1} \text{ is true then } H_{t2} \text{ must be true)}$$

- B_{t1}:
  - rule's body (a conjunction of conditions, e.g. occurrences of events, conditions regarding the state of the system)
  - It is typically expressed as a conjunction of Happens, HoldsAt, relational or time predicates
- H_{t2}:
  - rule's head (a number of consequences)
  - It is typically expressed as a conjunction of Happens, HoldsAt, relational or time predicates

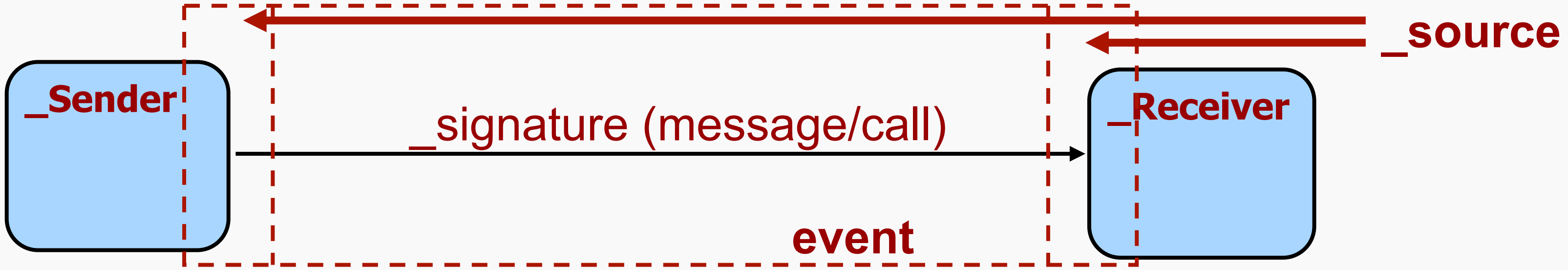
Specification of monitoring rules (2)

- Rules and assumptions are specified in Event Calculus – a first order temporal logic language – in terms of
  - **Events**: things that happen within a system of instantaneous duration (e.g. receipt of component messages, execution of internal or system operations)
  - **Fluents**: conditions about the state of a system

    relation(obj_1, …, obj_N)

- Predefined predicates:
  - Happens(e, t, $\Re$(t1,t2)) – occurrence of an event *e* of instantaneous duration at some time *t* within the time range $\Re$(t1,t2)
  - Initiates(e,f,t) – fluent *f* starts to hold after the event *e* at time *t*.
  - Terminates(e,f,t) – fluent *f* ceases to hold after the event *e* occurs at time *t*
  - HoldsAt(f,t) – fluent *f* holds at time t.
  - Relational predicates: x REL y (e.g. EqualTo, NotEqualTo, …)
  - Time predicates: t1 TREL t2 (e.g. TEqualTo, TLessThan …)

Specification of monitoring rules (3)



Events: General form

e(_id, _senderRole, _senderID, _receiverRole, _receiverID, _status, _signature _sourceRole, _sourceID))

- _signature: the type of a message sent by the component/system
- _status: indicates whether the message is incoming or outgoing
- _senderRole: the role of the component that sends the message
- _senderID: the id of the component that sends the message
- _receiverRole: the role of the component that receives the message
- _receiverID: the id of the component that receives the message
- _sourceRole: the role of the component at which the message is captured
- _sourceID: the id of the component at which the message is captured

Events typically correspond to operations defined in the interfaces of the components of the S&D pattern

Specification of monitoring rules (4)

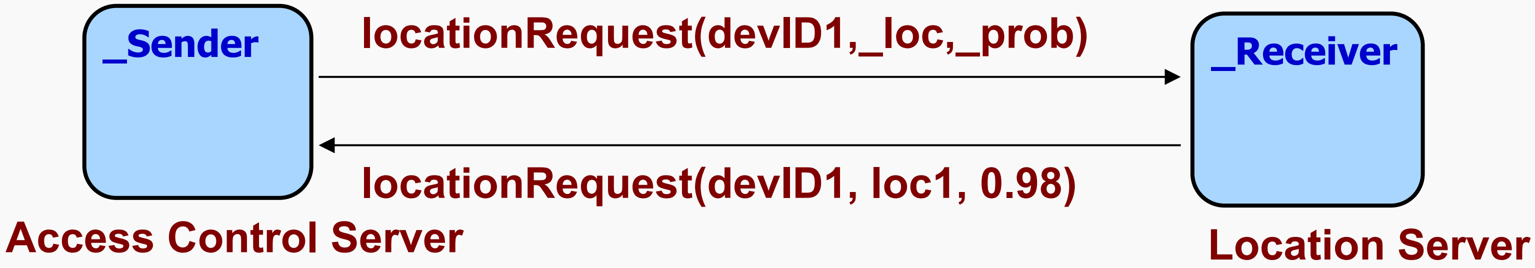
- Other features
  - Calls to built-in functions implementing complex computations (e.g. statistical functions)

Happens(e(…,REQ, o(),…), t_1, R(t_1, t_1)) ∧
Happens(e(…, RES, o(),…), t_2, R(t_1, t_2)) ∧
HoldsAt(o_response_times(RT[]), t_2) ⇒ m:append(RT[], $t_2 - t_1$), t_2)
HoldsAt(o_response_times(RT[]), t_1) ⇒ m:avg(RT[]) < k

Examples of monitoring rules:

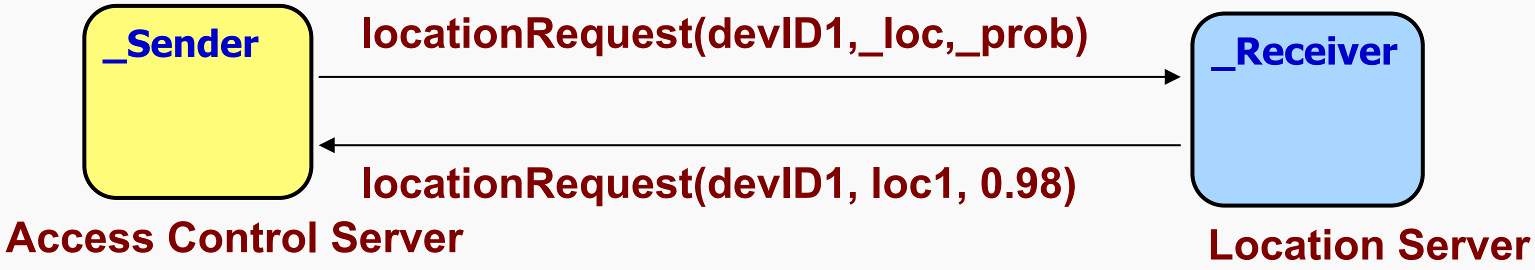
Part III

Rule for location server availability



Condition: when the access control server sends a location request to the location server it should receive a response from it within 3 seconds

Examples of monitoring rules:
Rule for location server availability



Condition: when the access control server sends a location request to the location server it should receive a response from it within 3 seconds

Rule 1

Happens(e(_eID1, _controlServerRole, _controlServerID, _locationServerRole, _locationServerID, REQ, locationRequest(_dev,_loc,_prob), _controlServerRole, _controlServerID), t1, R(t1, t1))

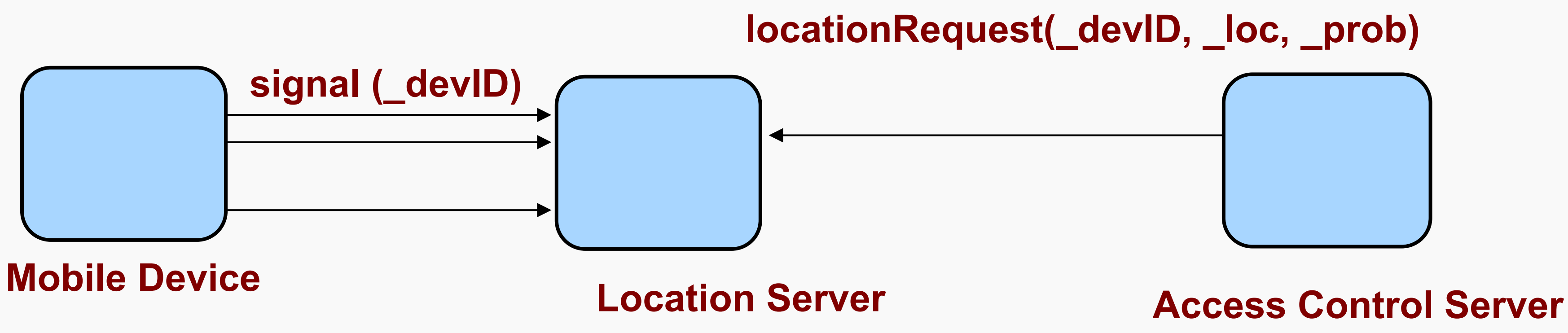
$\Rightarrow$

Happens(e(_eID2, _locationServerRole, _locationServerID, _controlServerRole, _controlServerID, RES, locationRequest(_dev, _loc, _prob), _controlServerRole, _controlServerID), t2, R(t1+1, t1+3000))

Examples of monitoring rules:

Part III

Rules for liveness of device daemons

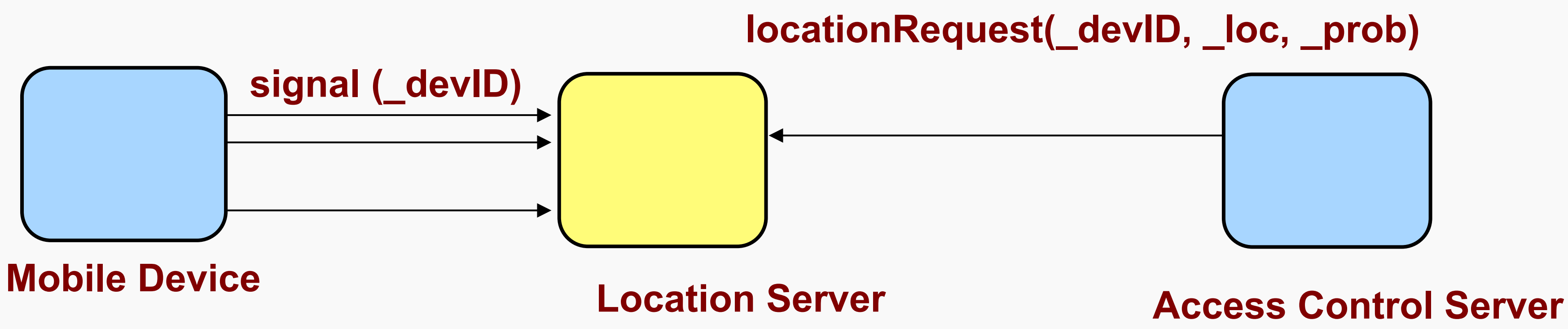


Condition: Every mobile device that is known to the control server should be sending signals to the location server periodically and the maximum period of not receiving a signal should not be less than *m* time units

Can be specified by 2 rules:

- A rule for checking when the first signal from a mobile device should be received
- A rule for checking the continuous receipt of signals

Examples of monitoring rules:
Rules for liveness of device daemons



Rule 2:

Happens(e(_eID1, _cServerRole, _cServerID, _lServerRole, _lServerID, REQ, locationRequest(_devID,_loc,_prob), _lServerRole, _lServerID), t1, R(t1,t1)) ∧

¬∃t2. **Happens**(e(_eID2, _cServerRole, _cServerID, _lServerRole, _lServerID, REQ, locationRequest(_devID,_loc1,_prob1), _lServerRole, _lServerID), t2, R(0,t1-1)) ⇒

∃t3. **Happens**(e(_eID3, _deviceRole, _devID, _lServerRole, _lServerID, RES, signal(_devID), _lServerRole, _lServerID), t3, R(t1-m,t1))

Rule 3:

Happens(e(_eID1, _deviceRole, _devID, _lServerRole, _lServerID, REQ, signal(_devID), _lServerRole, _lServerID), t1, R(t1,t1)) ⇒

Happens(e(_eID2, _deviceRole, _devID, _lServerRole, _lServerID, REQ, signal(_devID), _lServerRole, _lServerID), t1, R(t1,t1+m)) ∧ _eID1 ≠ _eID2

Examples of monitoring rules:

Rule for accuracy of location information

Condition: The accuracy of the device location information that is provided by the location server must always exceed a certain accuracy threshold

Examples of monitoring rules:
Rule for accuracy of location information

Condition: The accuracy of the device location information that is provided by the location server must always exceed a certain accuracy threshold

Rule 4

Happens(e(_eID1, _locationServerRole, _locationServerID, _controlServerRole, _controlServerID, RES, locationRequest(_dev,_loc,_prob), _controlServerRole, _controlServerID), t1, R(t1, t1))

$\Rightarrow$ _prob $\geq$ AT

Assumptions

- Used to deduce information about the state of the system and/or the occurrence of events
- Two types:
  - Monitoring assumptions: express how the state of a "system" that is being monitored is affected by events
  - Diagnostic assumptions: express expected patterns of correlated events (e.g. sequences of operation calls)
- Have the same general form with rules:

$$B_{t1} \Rightarrow H_{t2}$$

- B_{t1}: assumption's body (a conjunction of Happens, HoldsAt, relational or time predicates
- H_{t2}: assumption's head
  - In monitoring assumptions: a conjunction of fluent initiation and/or termination predicates (Initiates, Terminates predicates)
  - In diagnostic assumptions: a conjunction of Happens predicates

Assumptions: example

Condition: A device requesting access to a resource must have been authenticated

Assumptions: example

Condition: A device requesting access to a resource must have been authenticated

Rule 5

Happens(e((_eID1,,_sndRole,_sndID,
_recRole, _recID, REQ, **requestAccess(_devID, _resID)**, _recRole,_recID), t_1,
$R(t_1, t_1)) \Rightarrow$
HoldsAt(AUTHENTICATED(_devID), t_1, $R(t_1, t_1)$)

Assumption A1 (monitoring assumption)

Happens(e(_eID2,_recRole,_recID,
_senRole, _senID, RES, **connect(_devID, _res)**
_recRole,_recID), t_1, $R(t_1, t_1)) \wedge$ _res = True $\Rightarrow$
Initiates(e(_eID2, …), AUTHENTICATED(_devID), t_1, $R(t_1, t_1)$)

Monitoring Process

- It is based on a generic event calculus reasoning engine (see [1,6,7,8])
- Rule checking using
  - Runtime events
  - Fluents established by assumptions (deductive reasoning)
- Checks cover both past and bounded future EC formulas
  - Past formulas:

    $Happens(\ e_1,\ t_1,\ R(t_1,\ t_1)) \Rightarrow Happens(\ e_2,\ t_2,\ R(0,\ t_1))$
  - Bounded Future formulas:

    $Happens(\ e_1,\ t_1,\ R(t_1,\ t_1)) \Rightarrow Happens(\ e_2,\ t_2,\ R(t_1,\ t_1+K))$
- Ability to analyse
  - events captured from distributed sources with different clocks
  - events arriving at the monitor not in the same order as the order of their capture

Part IV:
Advanced Capabilities (Diagnosis and Prediction)

Monitoring process: diagnostic capabilities

- Given a violation of an S&D monitoring rule

  $$R: E_1, E_2, E_3, \ldots, E_n \Rightarrow E_{n+1}$$

  Calculate beliefs in the genuineness of the events $E_1, E_2, \ldots, \neg E_{n+1}$ which are involved in the violation since events might be the result of an attack or fault
- Overall Approach (see [5] and [7])
  - The genuineness of an event depends on the ability to find a valid explanation for it
    - An event explanation is a logical combination of other events and states of the system which would have the event as a consequence
    - An event explanation is considered to be valid if it has as consequences other events which have also been observed and are genuine
  - Possible event explanations are generated by abductive reasoning using the monitoring specifications of the active patterns of the system that is being monitored
  - Event genuineness is assessed by beliefs computed according the Dempster-Shafer theory of evidence

Diagnosis: Assessing Event Genuineness

Belief in event genuineness:

Assumption:

An event is genuine if there is at least one valid explanation for it, i.e., an explanation whose further consequences (if any) are genuine

Process:

- Generate explanations using abductive reasoning and a system behaviour model (expressed as assumptions in EC-Assertion)
- Check explanation validity by checking if the expected consequences of an explanation are genuine events themselves
- Limit analysis to a period "around" the event (diagnosis window)

Belief functions:

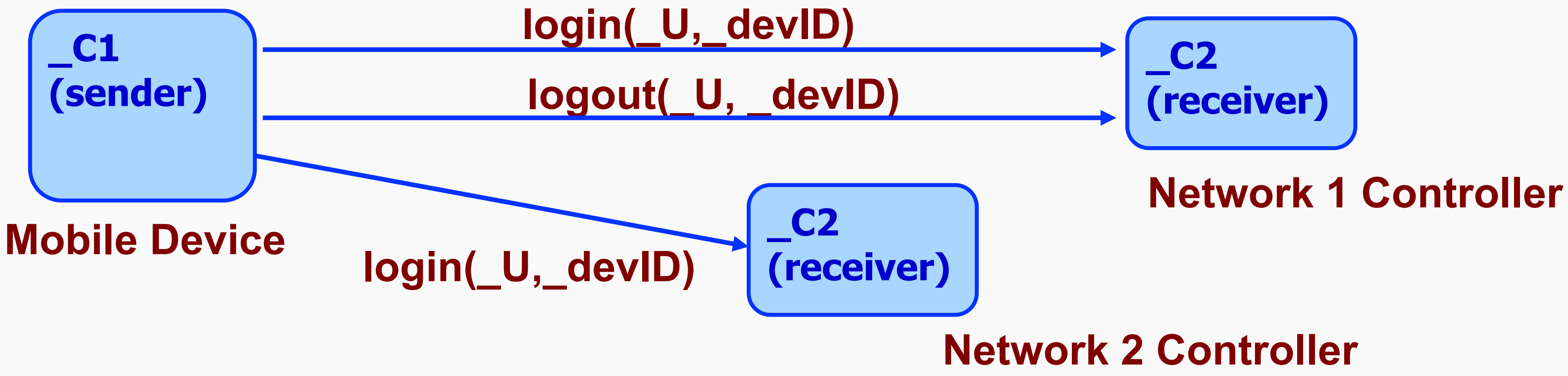
$m(E_i) = m^o(E_i) \times \{\Sigma_{J \subseteq EXP(Ei) and\ J \neq \emptyset}(-1)^{|J|+1}\{\Pi_{x \in J}\ mv\,(x,E_i)\}$ *If* $EXP(E_i) \neq \emptyset$

$= m^o(E_i) \times \beta_1$ *Otherwise*

$mv(x,E_i) = \Sigma_{S \subseteq Cons(x/Ei)\ and\ S \neq \emptyset}(-1)^{|S|+1}\{\Pi_{e \in S}\ m(e,E_i)\}$ *If* $Cons(x/E_i) \neq \emptyset$

$= \beta_2$ *Otherwise*

Diagnosis: Example



- Condition: no user should be allowed to login onto different parts of the WiFi network simultaneously (to reduce scope for masquerading attacks):

Rule-5:

∀ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time

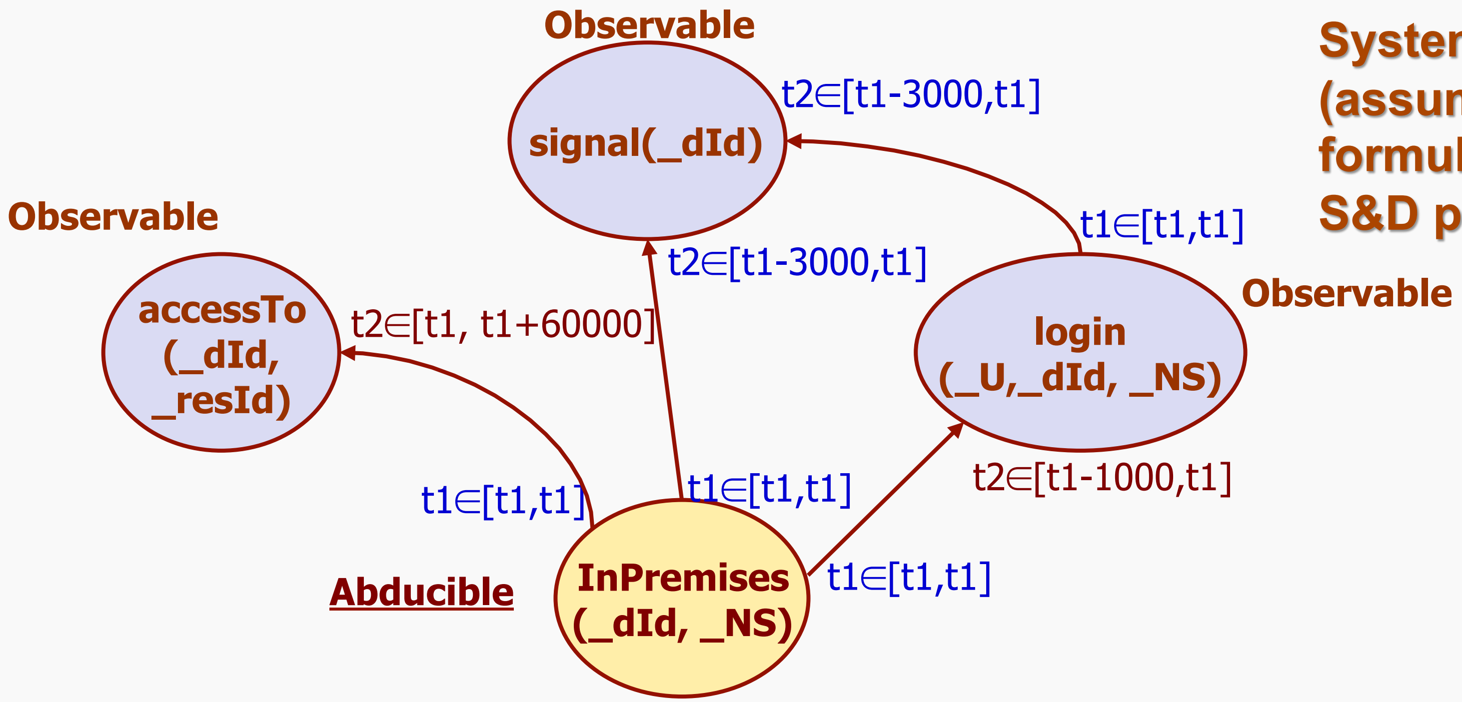
Happens(e(_E1, _C1Role, _C1, _C2Role, _C2,REQ, login(_U,_C1), _C2Role, _C2), t1,ℜ(t1,t1)) ∧

Happens(e(_E2, _C1Role , _C1, _C3Role, _C3,REQ, login(_U,_C1),_C3Role,_C3), t2,ℜ(t1,t2)) ∧ _C2 ≠ _C3

⇒

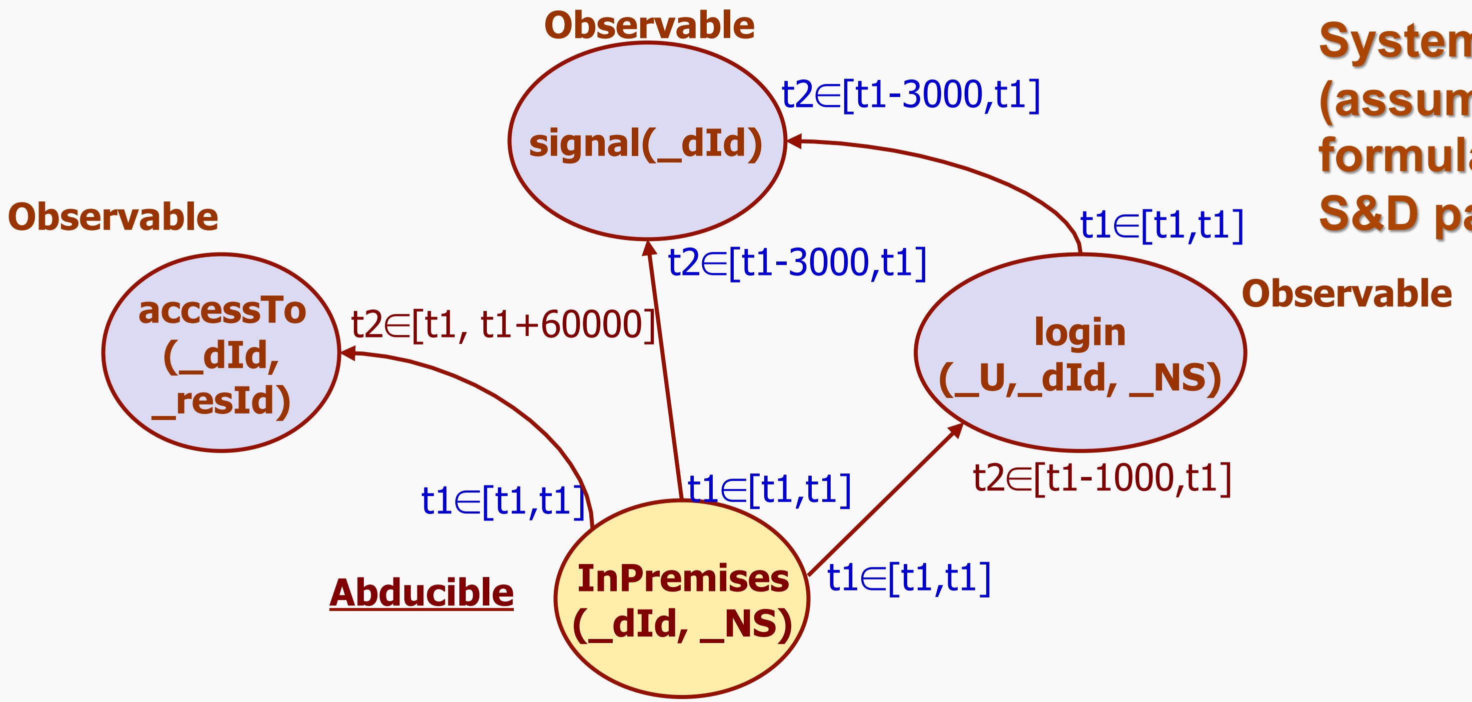
∃ t3: Time **Happens**(e(_E3,_C1,_C2,REQ, logout(_U,_C1), _C2),t3,ℜ(t1+1,t2-1))

Diagnosis: Example



System model (assumption formulas in the S&D pattern)

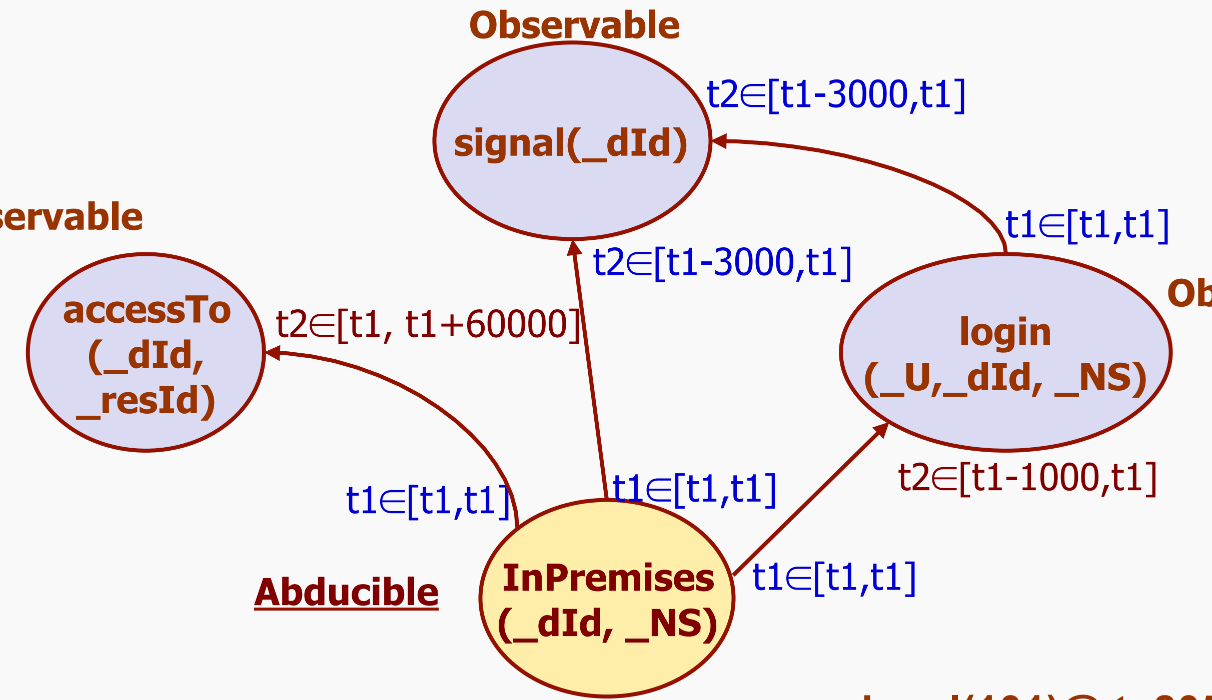
Diagnosis: Example



System model (assumption formulas in the S&D pattern)

- login(_, 101, n1) @ t=10050 $\Rightarrow_A$ InPremises (101,n1) @ t∈[9050,10050)
- InPremises(101,n1) @ t∈[9050,10050) ⇒ **signal(101) @ t∈[6050,10050)**
- InPremises(101,n1) @ t∈[9050,10050) ⇒ **accessTo(101, _) @ t∈[9050,69050)**

Diagnosis: Example



System model (assumption formulas in the S&D pattern)

- login(_, 101, n1) @ t=10050 $\Rightarrow_A$ InPremises (101,n1) @ t∈[9050,10050)
- InPremises(101,n1) @ t∈[9050,10050) ⇒ **signal(101) @ t∈[6050,10050)**
- InPremises(101,n1) @ t∈[9050,10050) ⇒ **accessTo(101, _) @ t∈[9050,69050)**

- **signal(101)@ t=8050**
  m(login(...)) = β_1 = 0.2
- **signal(101)@ t=8050**
  accessTo(101, _) @ t = 9801
  m(login(…)) = $\beta_1 + \beta_1 - \beta_1 \times \beta_1$ = 0.36
- **Explanation with no consequences**
  m(login(…)) = β_2 = 0.1

Monitoring process: threat detection capabilities

Detection of potential violations of S&D monitoring rules

$$R: E_1, E_2, E_3, \ldots, E_n \Rightarrow E_{n+1}$$

Calculate belief that R will be violated given the observation of a subset of $E_1, E_2, \ldots, E_{n+1}$

- Events might
  - Not be observed in the order they are expected by R
  - Be the result of an attack or fault (and therefore a belief in their genuineness needs to be estimated; see diagnosis)
- Approach (see [1])
  - Use DS beliefs to measure the likelihood of events genuineness and the likelihood of conditional event occurrence
  - Negate the rule to get the exact pattern of events that violates it
  - Construct a belief network indicating how beliefs in the violation of the rule can be updated as partial evidence about events in the pattern emerges

Threat detection: Belief graphs

- Negate the rule

  <u>Rule-5 attack signature</u>:

  ∀ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time

  Happens(e(_E1,_C1Role,_C1,_C2Role,_C2,REQ, login(_U,_C1,_C2),_C2Role, _C2),t1,$\Re$(t1,t1)) ∧

  Happens(e(_E2,_C1Role,_C1,_C3Role,_C3,REQ, login(_U,_C1,_C3),_C3Role, _C3),t2,$\Re$(t1,t2)) ∧ _C2 ≠_C3

  ⇒ ∀t3:Time ¬**Happens**(e(_E3,_C1Role,_C1,_C2Role, _C2,REQ, logout(_U,_C1,_C2), _C2Role, _C2),t3,$\Re$(t1+1,t2-1))

- Belief graph
  - Nodes represent events in rule attack signatures
  - "Start node": starting point for evidence collection
  - Edges: temporal constraints over events + belief functions

Threat detection: Belief graphs

- Negate the rule

  Rule-5 attack signature:

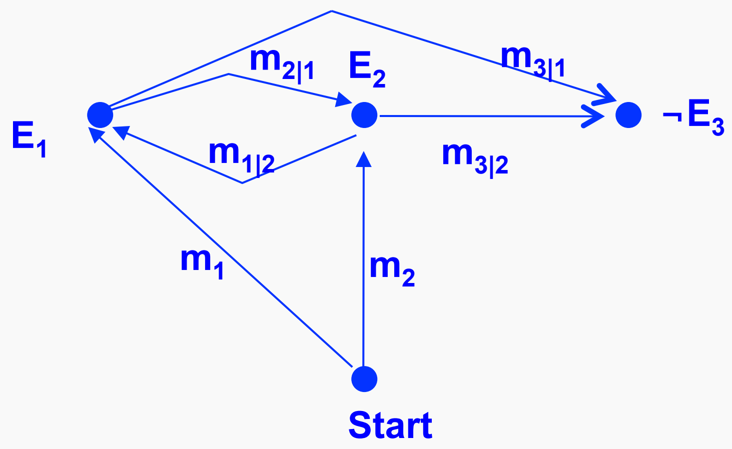
  ∀ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time

  Happens(e(_E1,_C1Role,_C1,_C2Role,_C2,REQ, login(_U,_C1,_C2),_C2Role, _C2),t1,ℜ(t1,t1)) ∧

  Happens(e(_E2,_C1Role,_C1,_C3Role,_C3,REQ, login(_U,_C1,_C3),_C3Role, _C3),t2,ℜ(t1,t2)) ∧ _C2 ≠ _C3

  ⇒ ∀t3:Time ¬**Happens**(e(_E3,_C1Role,_C1,_C2Role, _C2,REQ, logout(_U,_C1,_C2), _C2Role, _C2),t3,ℜ(t1+1,t2-1))

- Belief graph
  - Nodes represent events in rule attack signatures
  - "Start node": starting point for evidence collection
  - Edges: temporal constraints over events + belief functions



Threat Detection: Belief functions

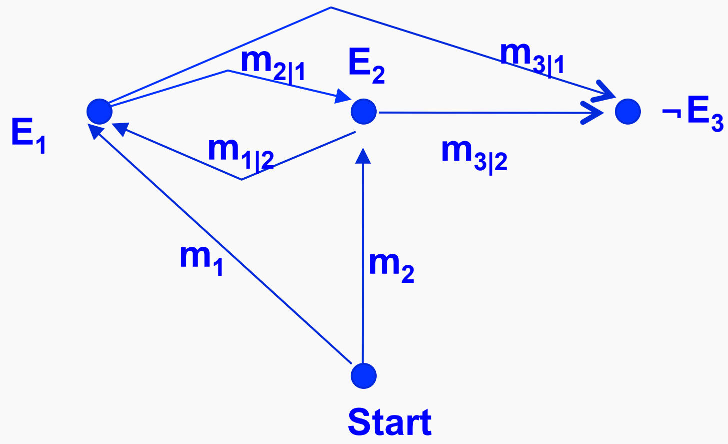
Conditional belief in event occurrences:

$$m_{i|j}(e_i) = \frac{\Sigma_{e \in Elog(Ej)}\, m(e) \times \{\Sigma_{J \subseteq Elog(Ei|e) and\ J \neq \emptyset} (-1)^{|J|+1} \{\Pi_{x \in J}\, m(x)\}}{\Sigma_{e \in Elog(Ej)}\, m(e)}$$

$$m_{i|j}(\neg e_i) = \frac{\Sigma_{ej \in Elog(Ej)}\, m(e) \times \{\Sigma_{ei \in Elog(Ei|ej)}\, m(\neg e_i)\}}{\Sigma_{ej \in Elog(Ej)}\, m(e)}$$

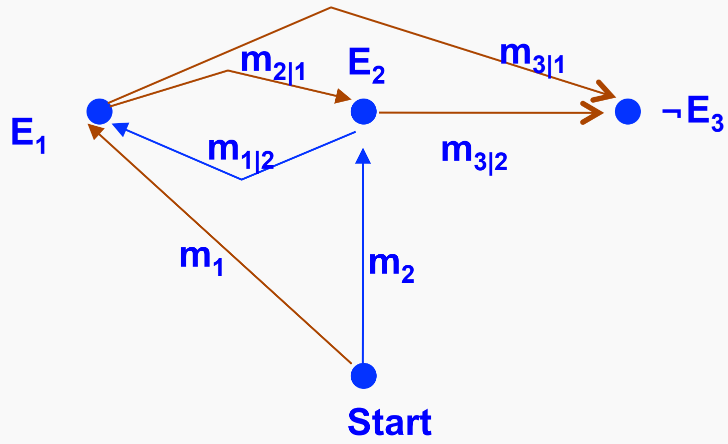
- Elog(E_j): Sample of N (sample size) randomly selected E_j events within the given sampling period
- Elog(E_i|e): set of the events of type E_i in the event log that have occurred within the time period determined by *e* and up to the time point when $m_{i/j}$ is calculated
- m(e)/m(x): basic belief in genuineness of e/x

Threat Detection: Example



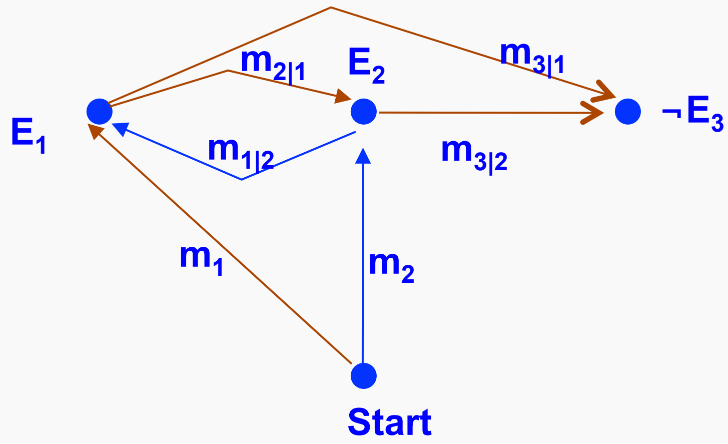
Threat Detection: Example

- login(u1, 101, n1) @ t=10050 occurs



Threat Detection: Example

- login(u1, 101, n1) @ t=10050 occurs



$m_1(E1) = k_1 = 0.8$

$m_1(\neg E1) = k_1' = 0.1$

$m_{2|1}(E2|E1) = k_{21} = 0.6$

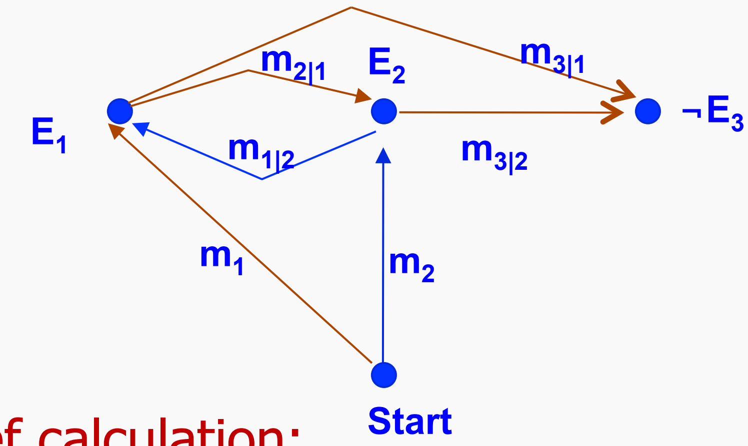
$m_{2|1}(\neg E2|E1) = k_{21}' = 0.4$

$m_{3|1}(E3|E1) = k_{31} = 0.2$

$m_{3|1}(\neg E3|E1) = k_{31}' = 0.6$

Threat Detection: Example

- login(u1, 101, n1) @ t=10050 occurs



$m_1(E1) = k_1 = 0.8$

$m_1(\neg E1) = k_1' = 0.1$

$m_{2|1}\ (E2|E1) = k_{21} = 0.6$

$m_{2|1}\ (\neg\ E2|E1) = k_{21}' = 0.4$

$m_{3|1}\ (E3|E1) = k_{31} = 0.2$

$m_{3|1}\ (\neg\ E3|E1) = k_{31}' = 0.6$

Threat belief calculation:

$$(m_1 \oplus m_{2|1} \oplus m_{3|1}(E_1 \wedge E_2 \wedge \neg E_3)) =$$

$$\frac{k_{31}^{\copyright}k_{21}k_1 + k_{31}^{\copyright}k_1(1-k_{21}-k_{21}^{\copyright}) + k_{31}^{\copyright}k_{21}(1-k_1-k_1^{\copyright})}{1-(k_{31}^{\copyright}k_{21}^{\copyright}(1-k_1^{\copyright})+k_{31}^{\copyright}k_{21}^{\copyright}(1-k_1^{\copyright})} =$$

$$\frac{0.6*0.6*0.8+0.6*0.8*0+0.6*0.6*1}{1*(0.2*0.4*0.9+0.6*0.4*0.9)} = 0.45$$

Threat Detection: Evaluation

Evaluated properties

- Threat reaction time: TRT = $T_{mon} - T_{TDT}$
- Precision: $PR = TTS_{BR} / (TTS_{BR} + FTS_{BR})$
  - TTS_{BR} : number of threat signals with a belief in a given range (BR) that ended up to eventual violations of the relevant rule detected by the EVEREST monitor (true signals)
  - FTS_{BR}: number of the threat signals with belief in a given range (BR) that did not correspond to an eventual violation of the relevant rule
- Analysis of effect of
  - Diagnosis window (DW)
  - Sample size (SS)

Set up

  - Simulation of workflow of LBACS system
  - 8 sets of 2,000 events (different variances in inter-event arrival times)

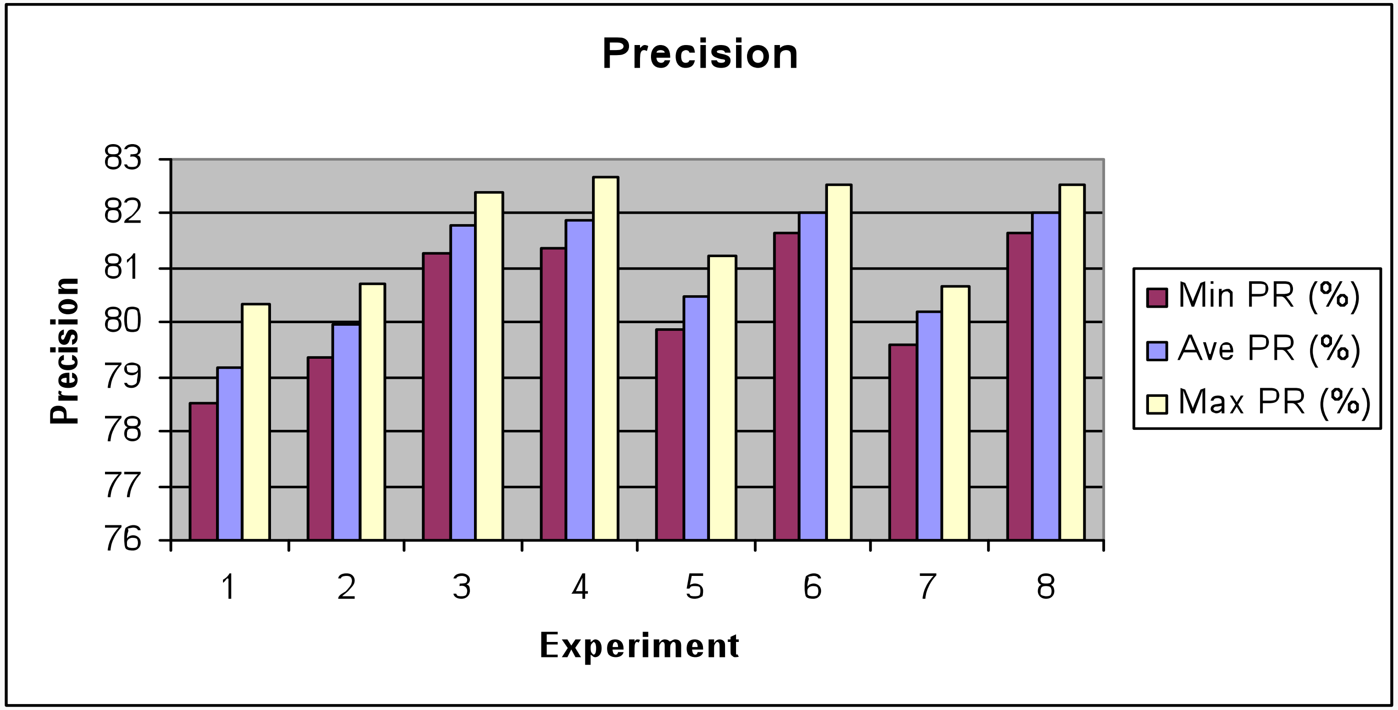
Threat Detection: TRT

EXP	VV	DW	SS	pos %	neg %	ave TRT	max TRT	min TRT
1	0.3	15000	10	77.54	21.51	9.3	852.5	-4.2
2	0.3	20000	15	73.21	26.53	10.4	753.9	-4.5
3	0.5	15000	10	80.18	19.02	12.5	1137	-1.9
4	0.5	20000	15	72.08	27.39	13.2	1111	-3
5	0.6	15000	10	79.45	20.03	12.3	1077	-2.3
6	0.6	20000	15	74.87	24.74	14	1077	-29
7	0.9	15000	10	80.24	18.85	13.6	1077	-3
8	0.9	20000	15	74.87	24.74	14.1	1077	-29

TRT (secs)

- Average threat reaction time: 9.3 to 14.1 seconds
- Sufficient time for taking some types of pre-emptive action (e.g. deactivation of system components)

Threat detection: precision



- Varied from 78% to 83%
- Diagnosis window (DW) and sample size (SS) increments caused marginal increase in it (≤ 1.8 %) – see Exp1/Exp2, Exp3/Exp4, Exp5/Exp6, Exp7/Exp8 (caused maximum increase)

Part V:
Reaction

Reaction to monitoring results

- In some cases, following the detection of a problem whilst monitoring an S&D solution it might be possible to take some action that
  - Rectifies the problem, and/or
  - Prevents further harm
- Examples: In LBACS:
  - If the location server becomes unavailable, it might be necessary to deactivate the operation of the system unless the problem is repaired (action 1)
  - If more than X location sensors become unavailable the system may switch to WiFi only access control solution and access to certain resources may be deactivated (action 2)
- Some actions are possible to automate …

Our approach in SERENITY

- Reactions are realised by actions taken at runtime by the SERENITY Runtime Framework following the receipt of monitoring results from EVEREST

- Specification of actions:

  Rule specification = EC formula + [($action_1$, cnd_1), …, ($action_N$, cnd_N)]

- Semantics:
  - Each of the actions ($action_i$) is executed only if the condition associated with it is also satisfied (cnd_i)
  - The actions are executed in the exact order that they appear in the rule specification

- The SRF supports only predefined types of actions
- Complex conditions may be associated with actions

Predefined action types

- Action types
  - `DeactivatePattern()`
  - `RestartPattern()`
  - `NotifySRF(String external_SRF_ID, String Message)`
  - `NotifyApplication(String message)`
  - `StopMonitoringRules(String ruleID1, String ruleID2,… String ruleIDn)`
  - `StartMonitoringRules(String ruleID1, String ruleID2,… String ruleIDn)`
  - `Log()`

Monitoring results

- **Basic monitoring**

  Rule: $E_1, E_2, E_3, \ldots, E_n \Rightarrow E_{n+1}$

  - detect whether $E_1, E_2, E_3, \ldots, E_n, \neg E_{n+1}$ has happened
  - **RESULTS:** Instances of the events $E_1, E_2, E_3, \ldots, E_n, \neg E_{n+1}$ that have caused the violation are returned by EVEREST

- **Monitoring with enabled diagnosis**

  Rule: $E_1, E_2, E_3, \ldots, E_n \Rightarrow E_{n+1}$

  - detect whether $E_1, E_2, E_3, \ldots, E_n, \neg E_{n+1}$ are genuine
  - **RESULTS:** As in core monitoring + a belief range $[Bel(E_i), 1\text{-}Bel(\neg E_i)]$ indicating the belief in the genuineness of each of the events E_i

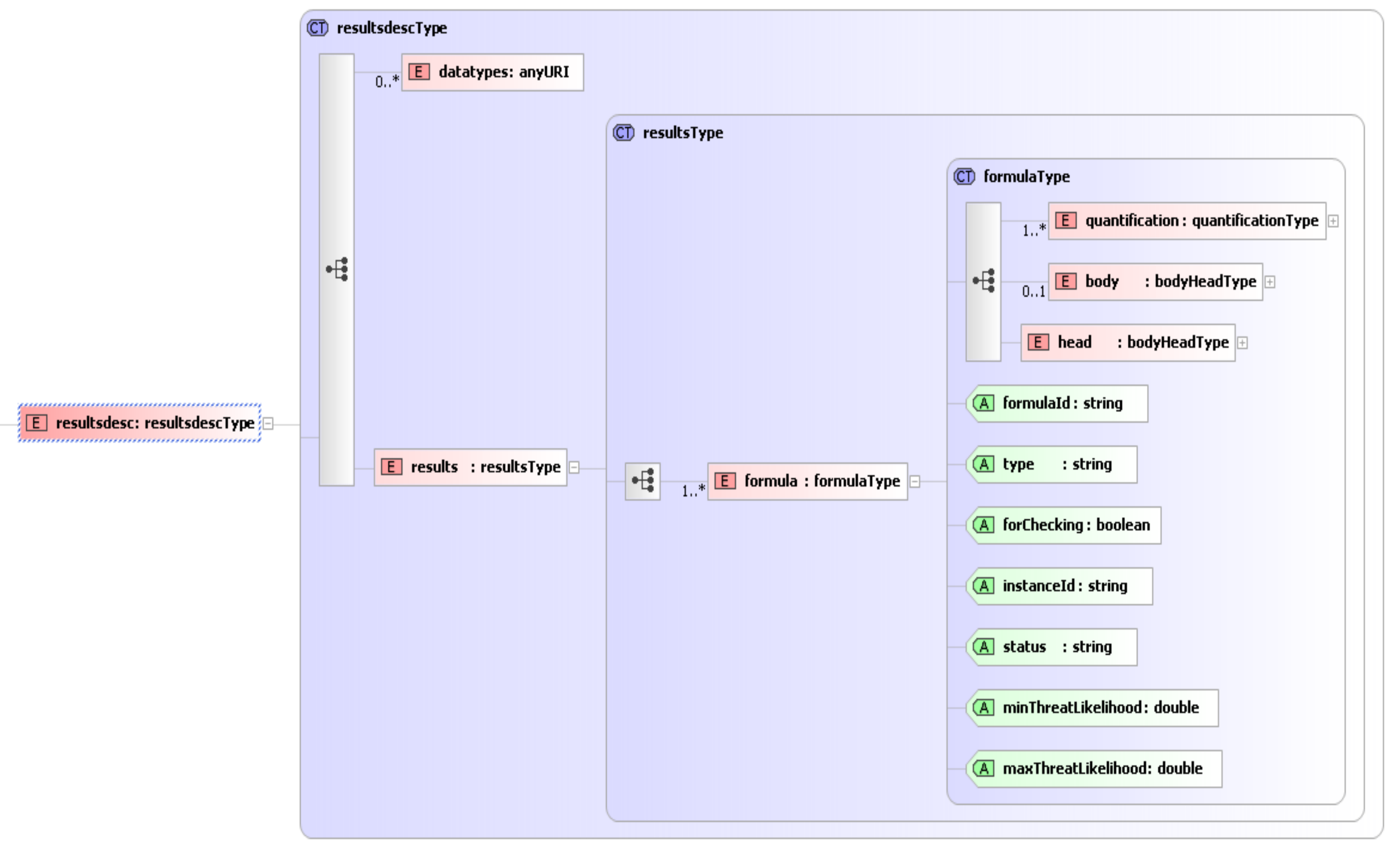
- **Treat detection**

  Rule: $E_1, E_2, E_3, \ldots, E_n \Rightarrow E_{n+1}$

  - Given a subset of seen events $OE \subset \{E_1, E_2, E_3, \ldots, E_n\}$ calculate the probability that $\{E_1, E_2, E_3, \ldots, E_n\} - OE \cup \{\neg E_{n+1}\}$ will occur
  - **RESULTS:** instances of the seen set of events OE, belief ranges for their genuineness + a belief range for a potential violation of the rule

Monitoring results

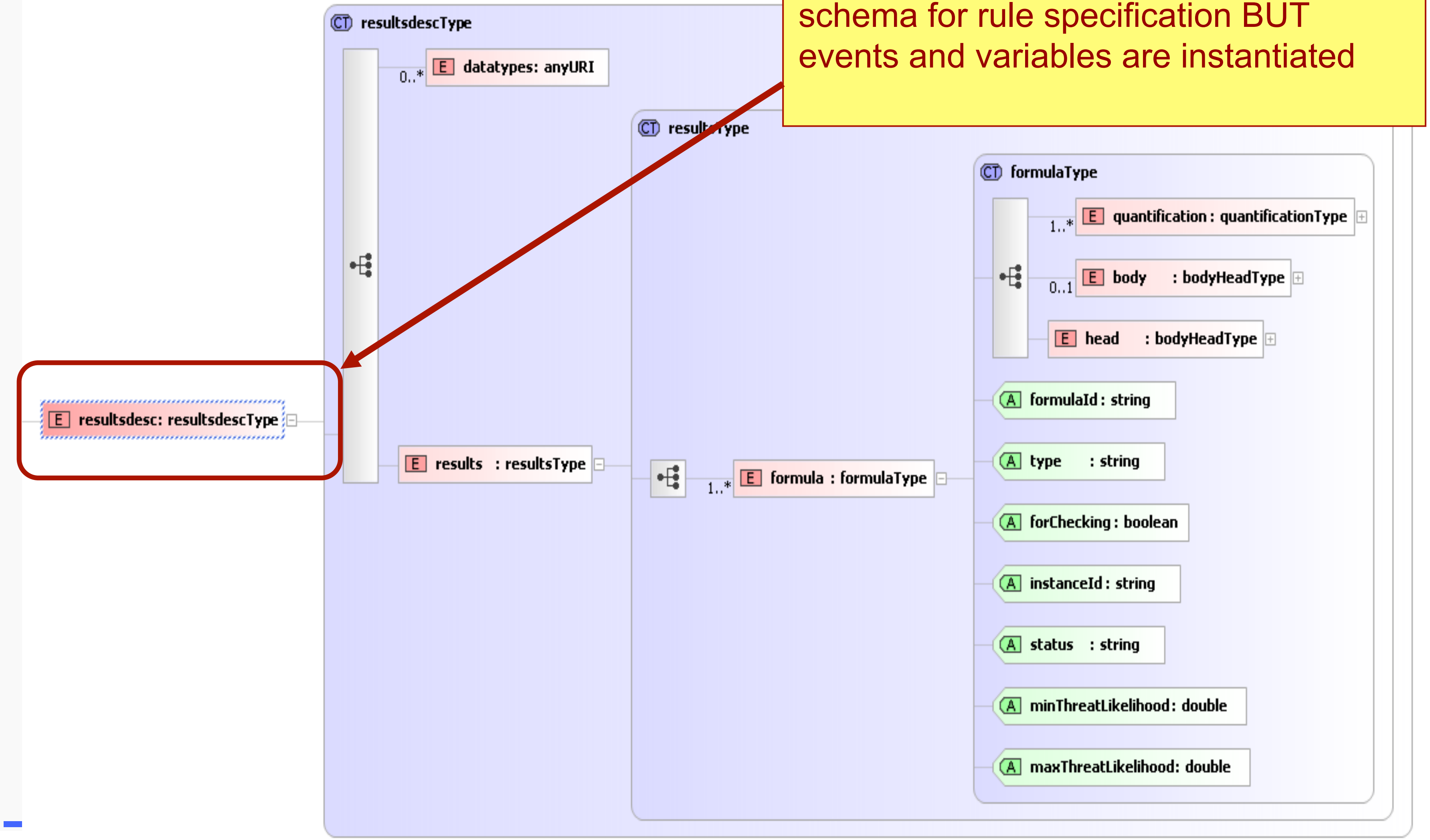
- Reported to SRF in XML



Monitoring results

- Reported to SRF in XML

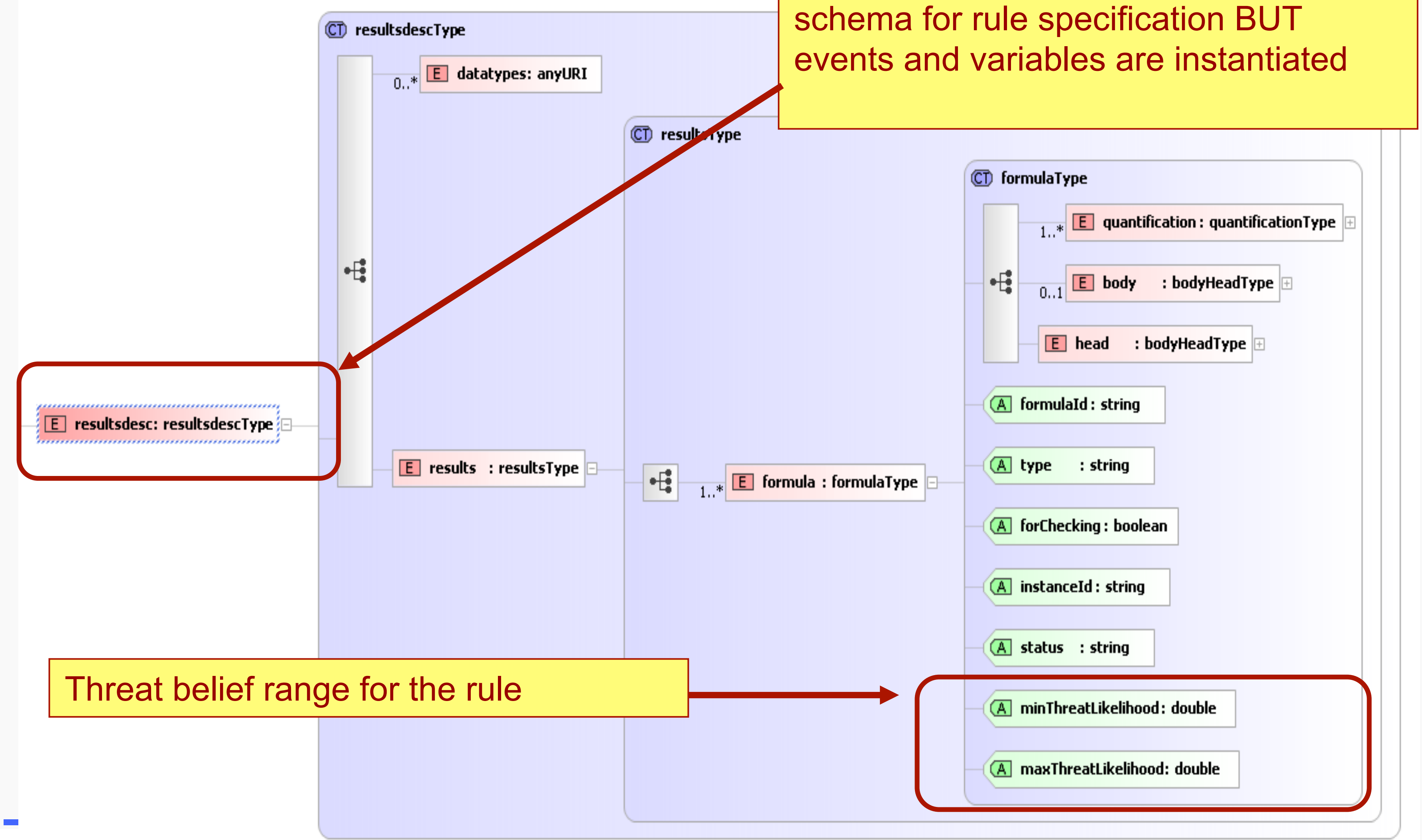
The basic schema is the same as the schema for rule specification BUT events and variables are instantiated



Monitoring results

- Reported to SRF in XML

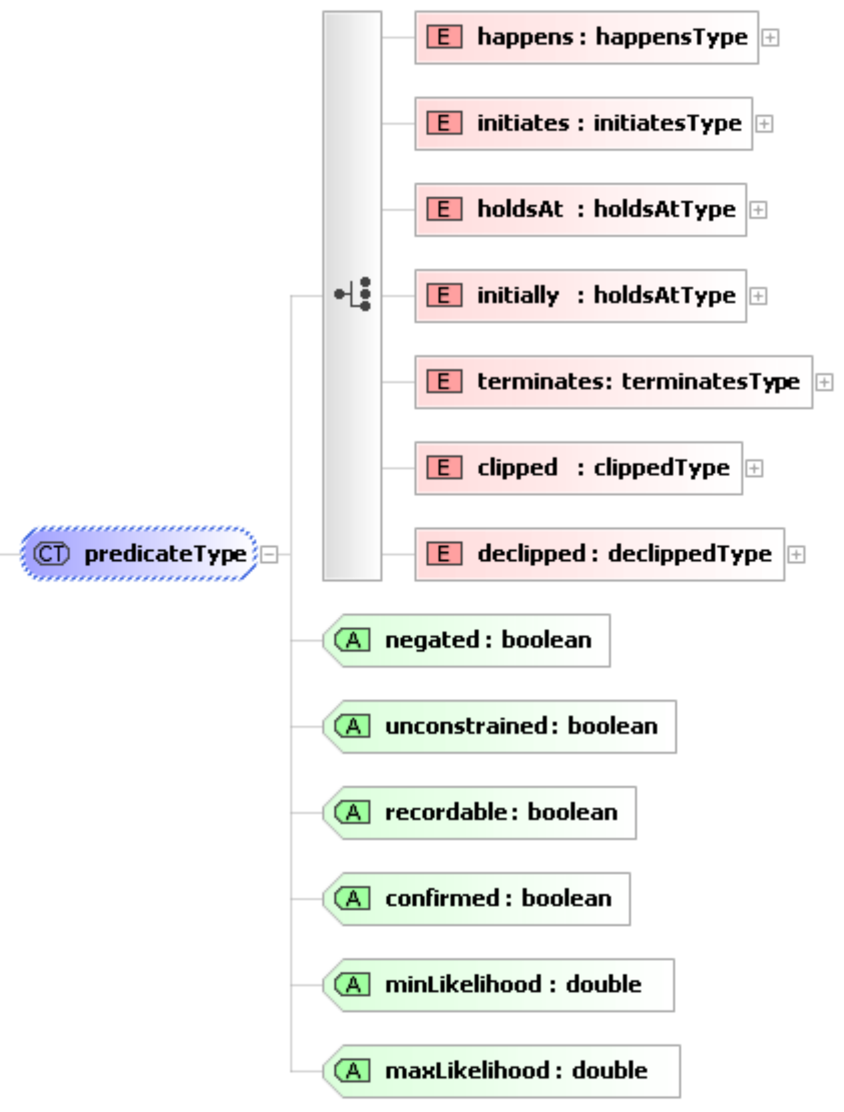
The basic schema is the same as the schema for rule specification BUT events and variables are instantiated



Threat belief range for the rule

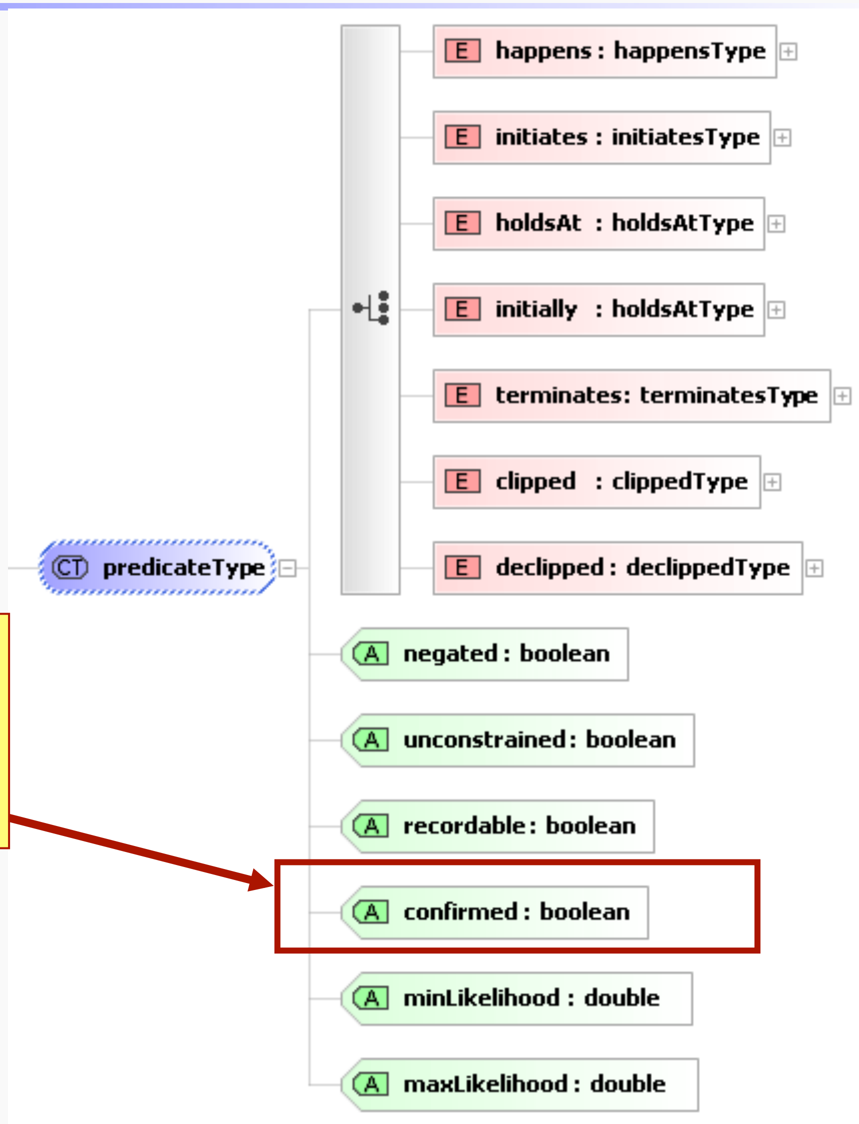
Monitoring results

- **At the level of individual conditions**



Monitoring results

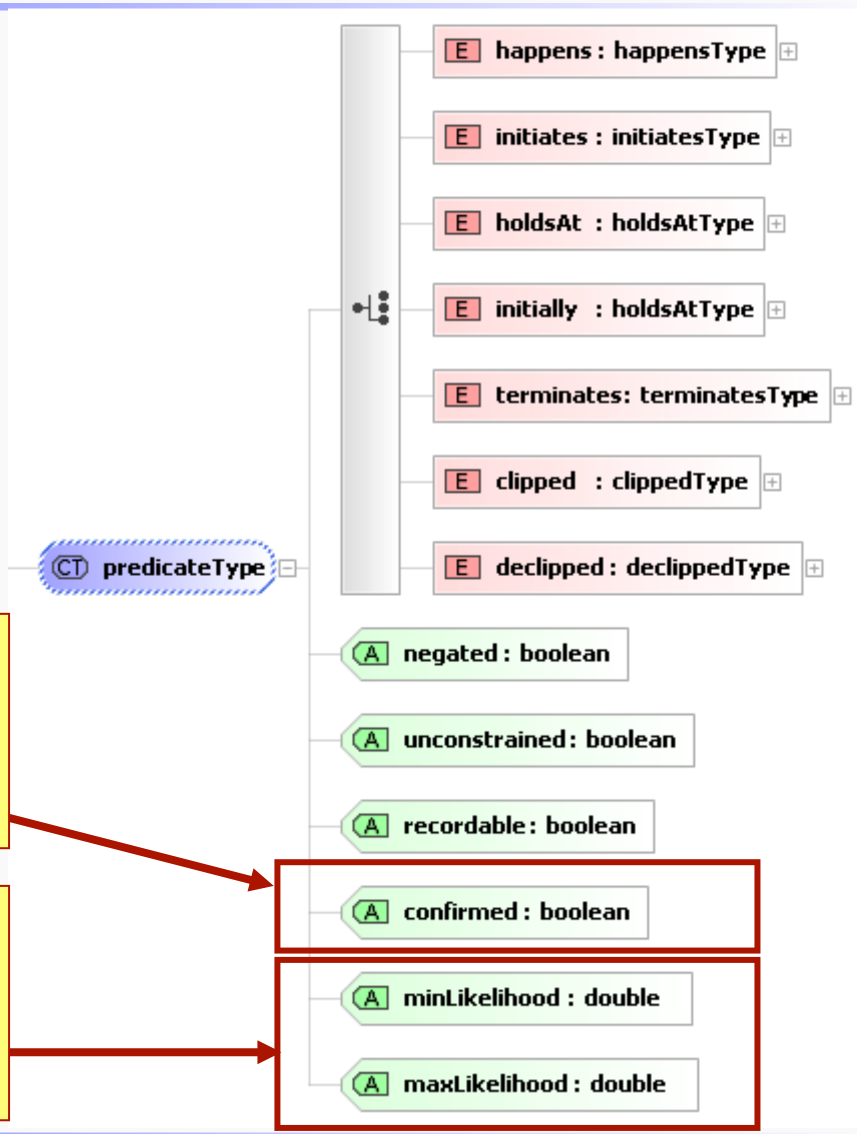
- **At the level of individual conditions**



Attribute indicating whether the event unified with the predicate is genuine; used only in diagnosis results

Monitoring results

- **At the level of individual conditions**

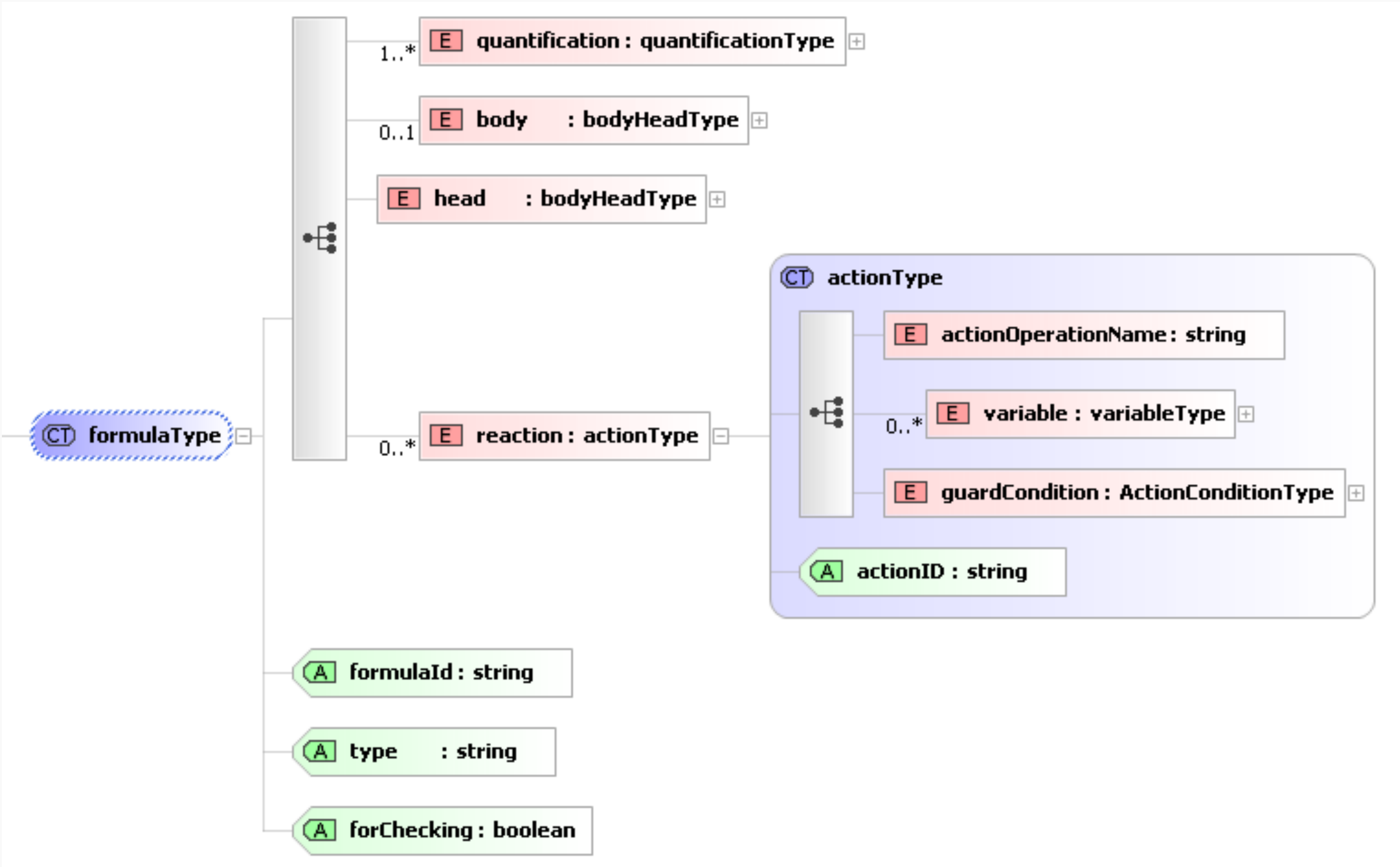


Attribute indicating whether the event unified with the predicate is genuine; used only in diagnosis results

Attributes representing the predicate belief range; used both for diagnosis and threat detection results

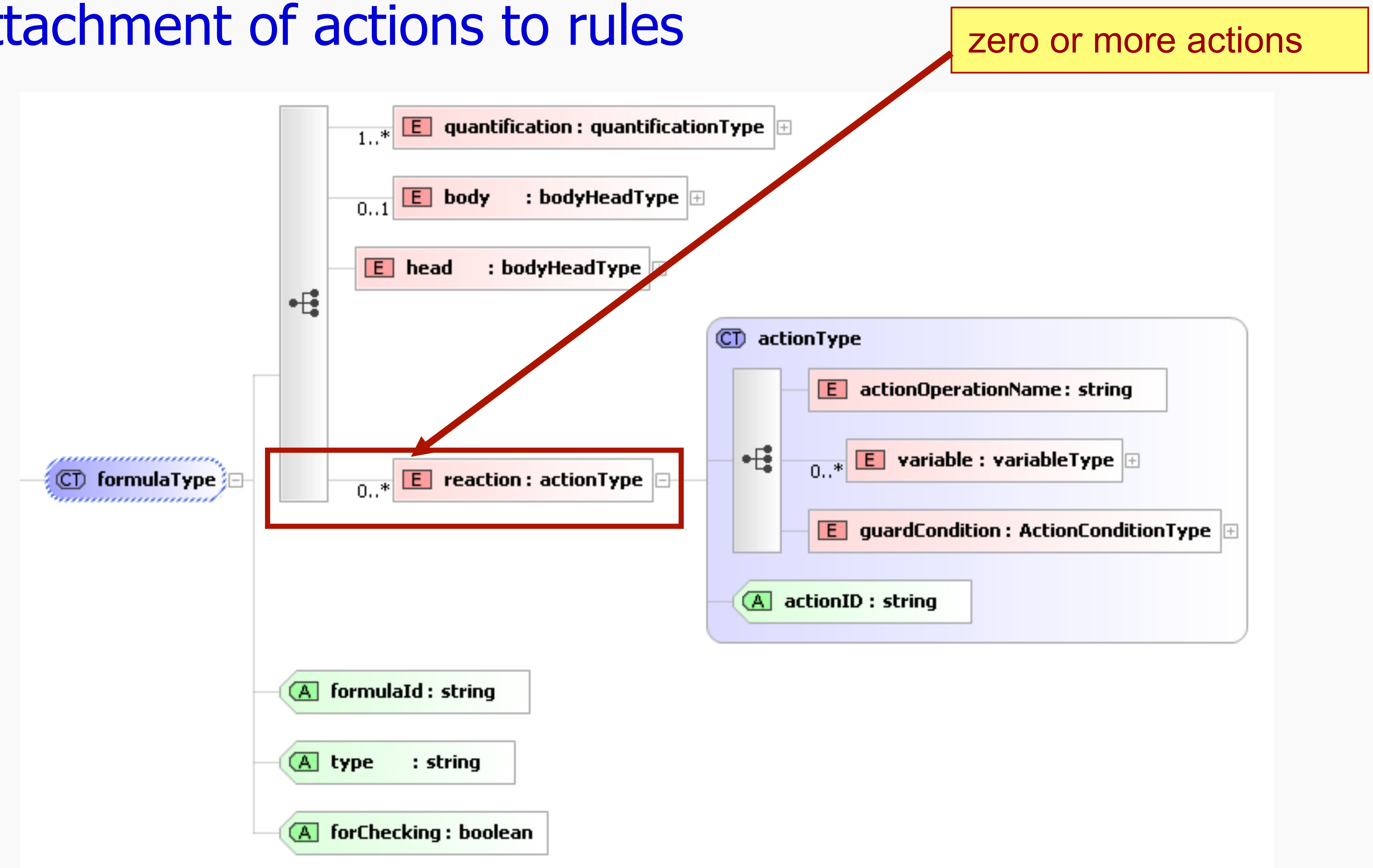
Action specification schema

- Attachment of actions to rules



Action specification schema

- Attachment of actions to rules

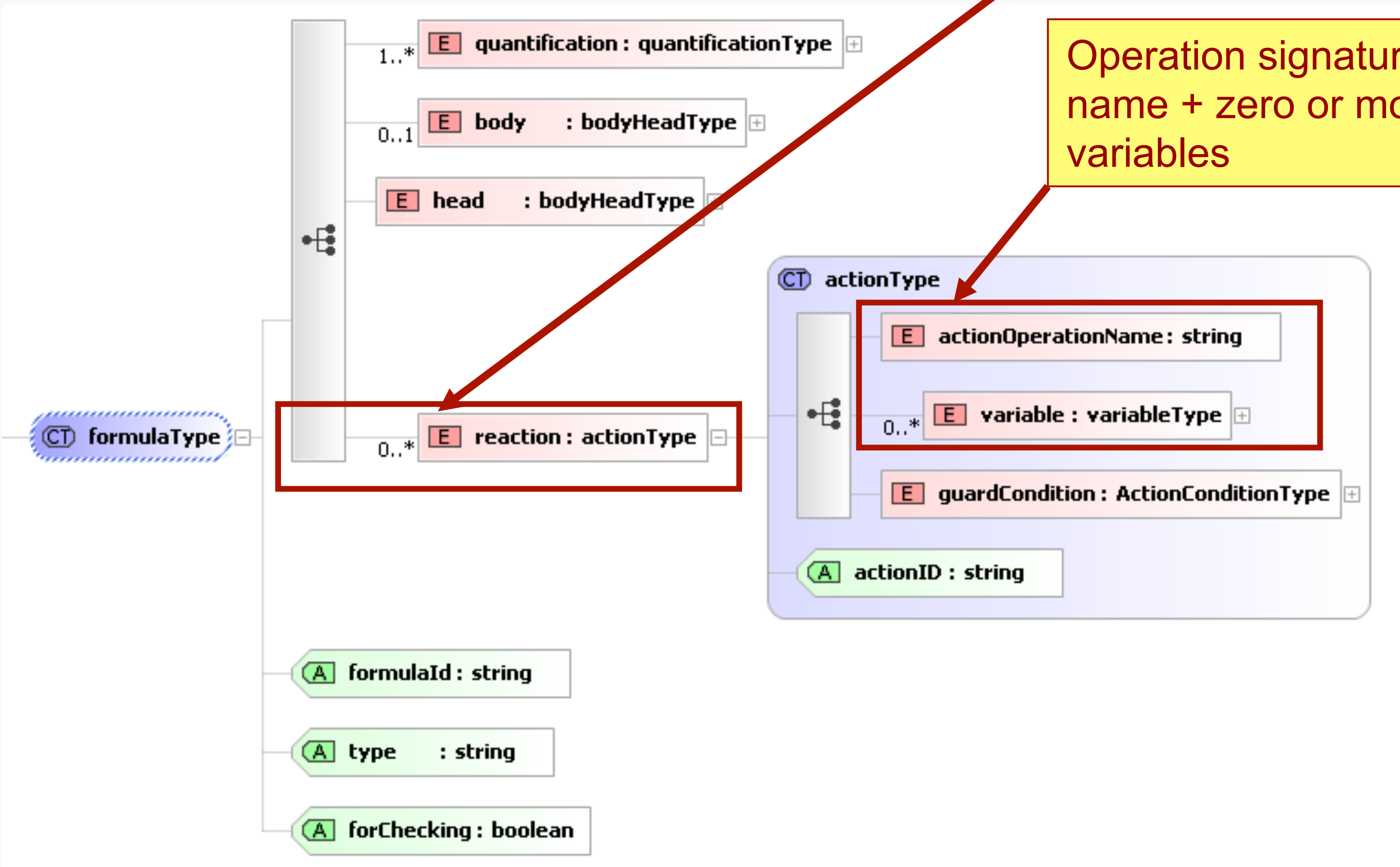


Action specification schema

- Attachment of actions to rules

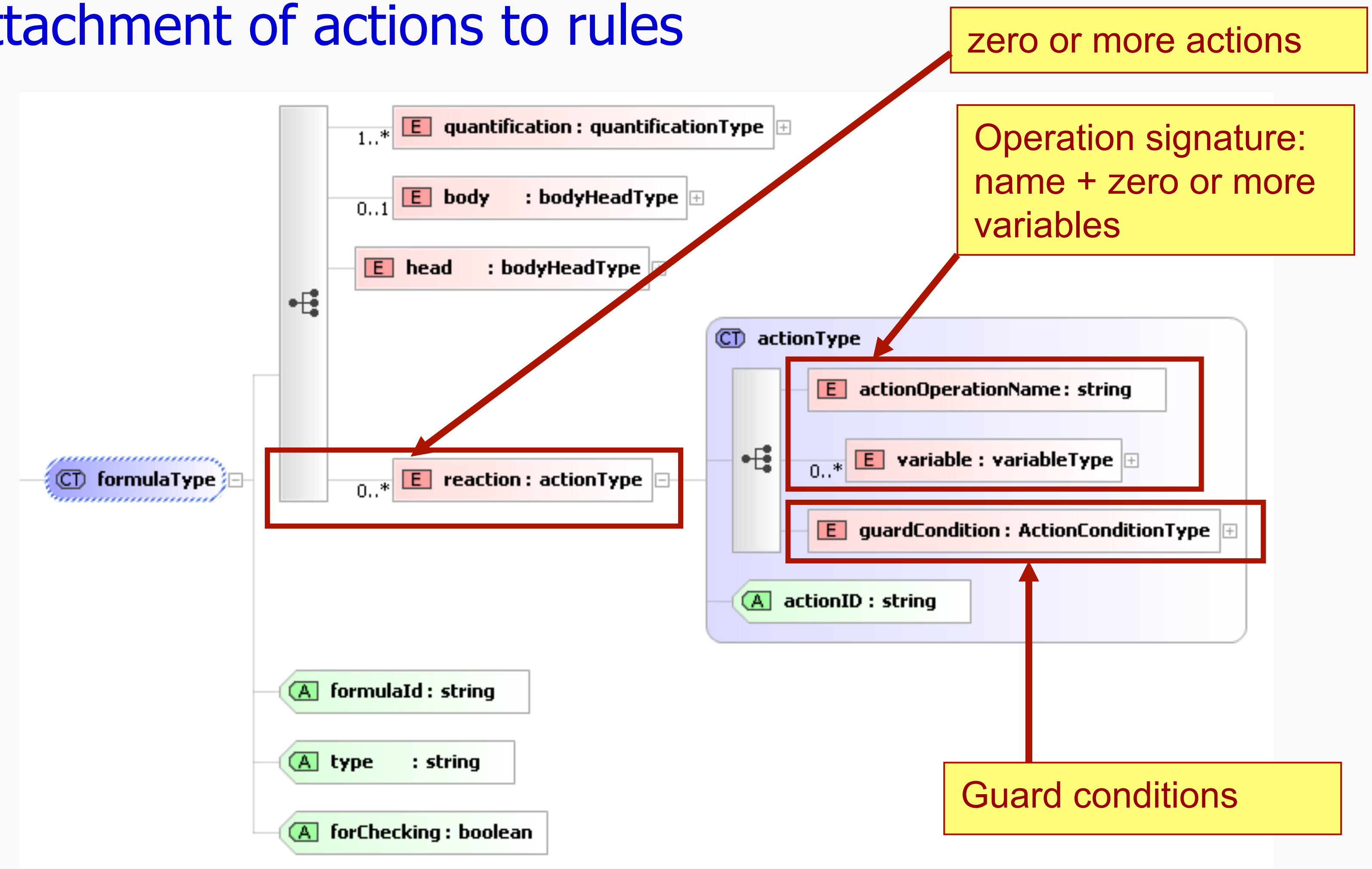
zero or more actions

Operation signature: name + zero or more variables



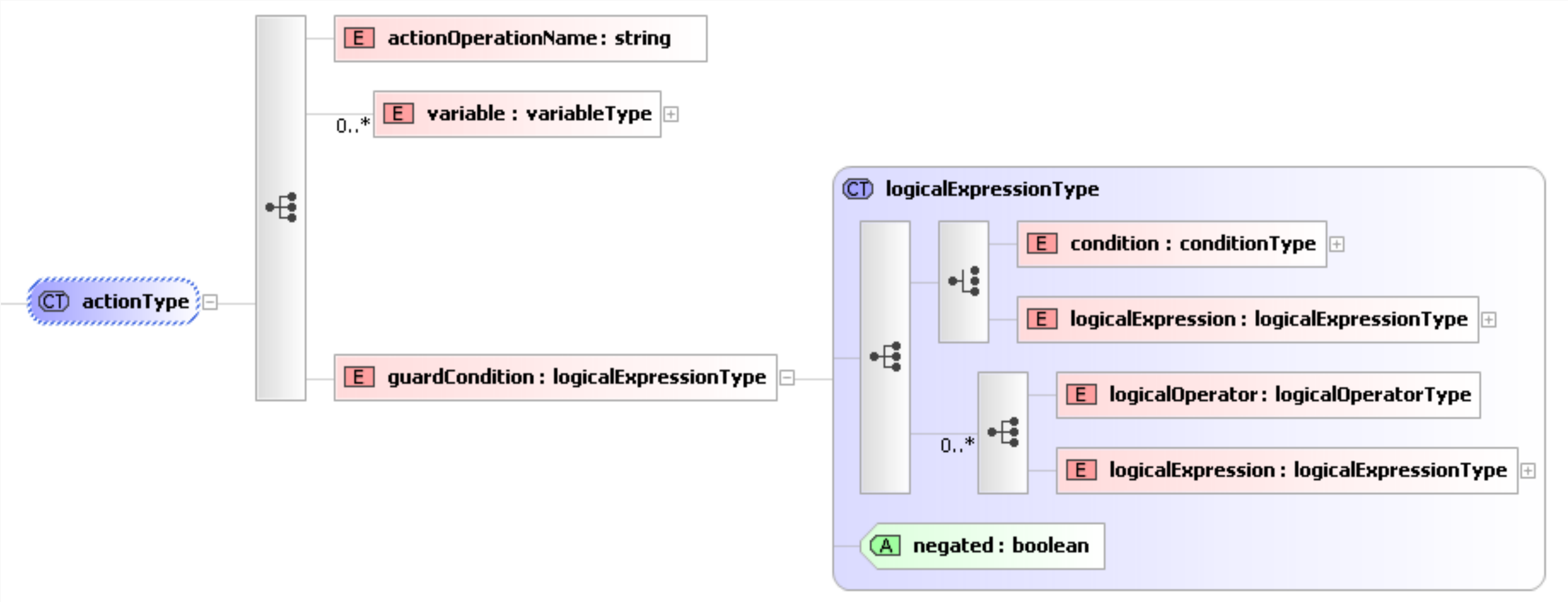
Action specification schema

- Attachment of actions to rules



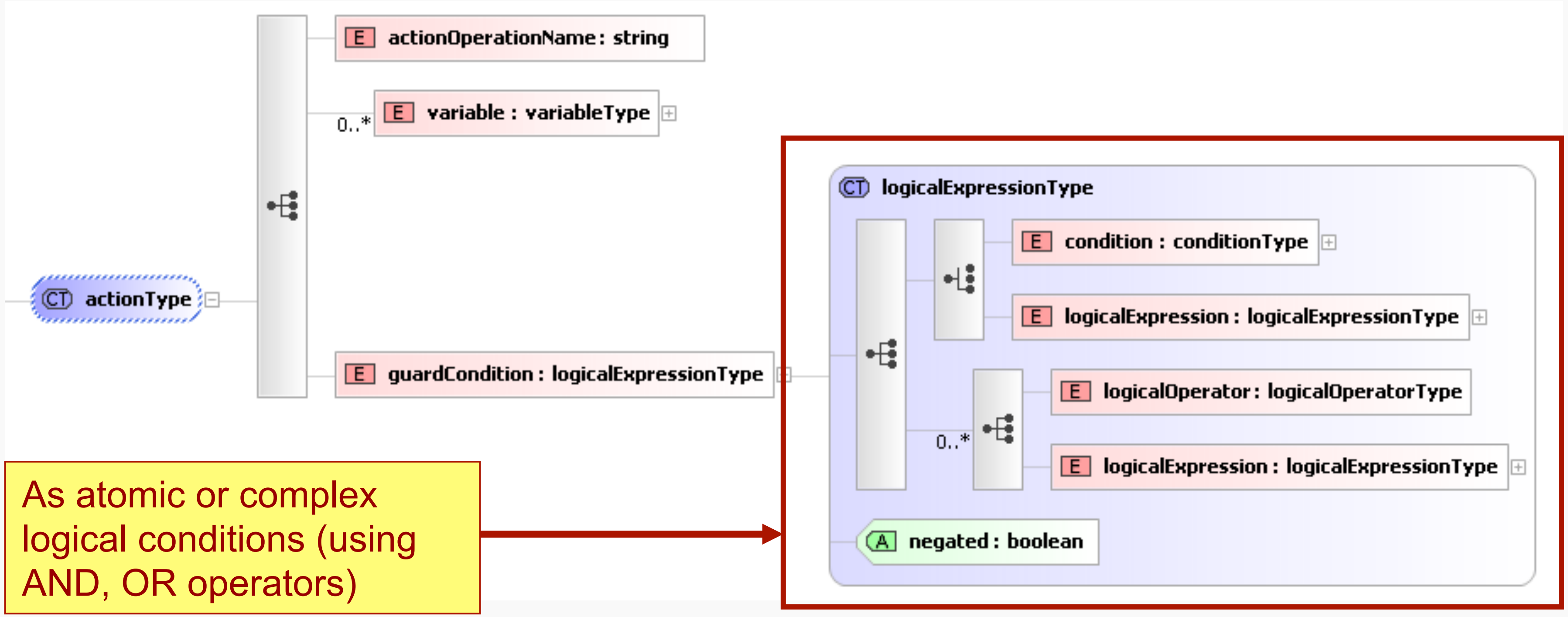
Action specification schema

- Specification of guard conditions for actions



Action specification schema

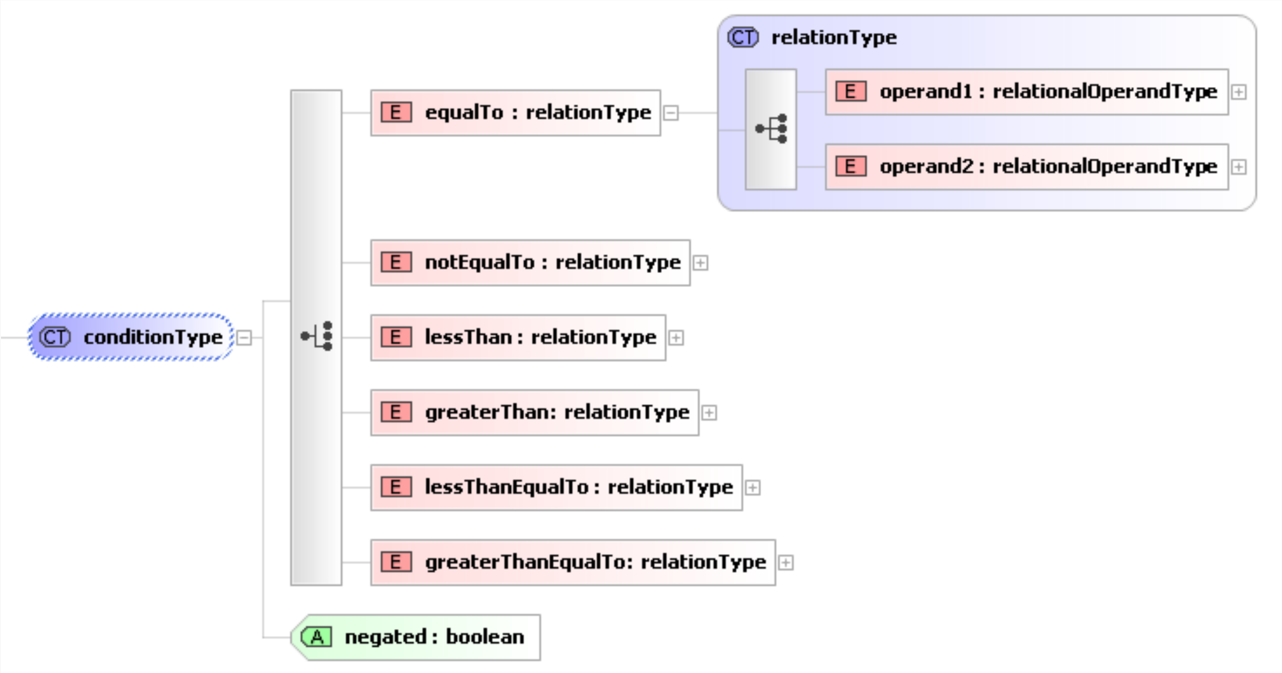
- Specification of guard conditions for actions



As atomic or complex logical conditions (using AND, OR operators)

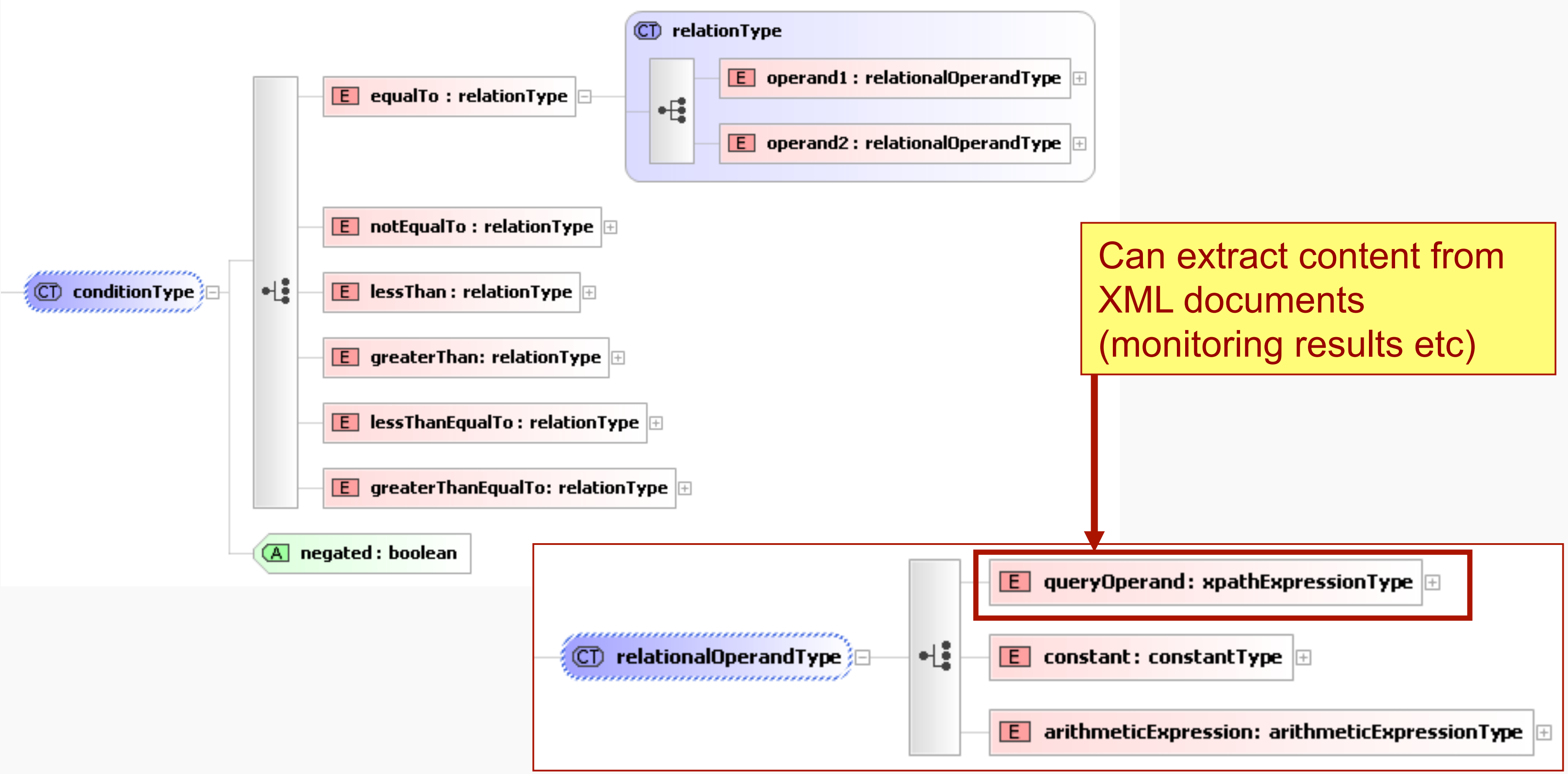
Action specification schema

- Specification of guard conditions for actions



Action specification schema

- Specification of guard conditions for actions



Actions: example 1

Rule-5: $\forall$ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time
Happens(e(_E1, _C1R, _C1, _C2R, _C2,REQ, login(_U,_C1), _C2R, _C2), t1,$\Re$(t1,t1)) $\wedge$
Happens(e(_E2, _C1R , _C1, _C3R, _C3,REQ, login(_U,_C1),_C3R, _C3), t2,$\Re$(t1,t2)) $\wedge$ _C2 $\neq$_C3
$\Rightarrow$ $\exists$ t3: Time **Happens**(e(_E3,_C1,_C2,REQ, logout(_U,_C1), _C2),t3,$\Re$(t1+1,t2−1))

Actions: example 1

Rule-5: $\forall$ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time
Happens(e(_E1, _C1R, _C1, _C2R, _C2,REQ, login(_U,_C1), _C2R, _C2), t1,$\Re$(t1,t1)) $\wedge$
Happens(e(_E2, _C1R , _C1, _C3R, _C3,REQ, login(_U,_C1),_C3R, _C3), t2,$\Re$(t1,t2)) $\wedge$ _C2 $\neq$ _C3
$\Rightarrow$ $\exists$ t3: Time **Happens**(e(_E3,_C1,_C2,REQ, logout(_U,_C1), _C2),t3,$\Re$(t1+1,t2−1))

```
<action>
   <actionOperationName>NotifyApplication</actionOperationName>
   <variable persistent="0" forMatching="false“>
            <varName>userId</varName><varType>string</varType>
            <value>/resultsdesc/results/formula/body/predicate[0]/happens/ic_term/variable[0]
            /varName[text()=“_U"]/value</value>
   </variable>
  <guardCondition negated="false">
  <condition negated="false">
     <equalTo>
            <operand1><queryOperand>
            <document><name>R5_Result</name><type>MonitoringResults</type></document>
            <xpath>/resultsdesc/results/formula[@status] </xpath>
            </queryOperand></operand1>
            <operand2><constant><type>STRING</type>
                        <value>Inconsistency_WRT_Recorded_Behaviour</value></constant>
            </operand2>
     </equalTo>
  </condition>
 </guardCondition>
</action>
```

Action taken if *Rule-5* is violated

Actions: example 2

Rule-5: ∀ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time
Happens(e(_E1, _C1R, _C1, _C2R, _C2,REQ, login(_U,_C1), _C2R, _C2), t1,$\Re$(t1,t1)) ∧
Happens(e(_E2, _C1R , _C1, _C3R, _C3,REQ, login(_U,_C1),_C3R, _C3), t2,$\Re$(t1,t2)) ∧ _C2 ≠ _C3
⇒ ∃ t3: Time **Happens**(e(_E3,_C1,_C2,REQ, logout(_U,_C1), _C2),t3,$\Re$(t1+1,t2−1))

Actions: example 2

Rule-5: ∀ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time
Happens(e(_E1, _C1R, _C1, _C2R, _C2,REQ, login(_U,_C1), _C2R, _C2), t1,ℜ(t1,t1)) ∧
Happens(e(_E2, _C1R , _C1, _C3R, _C3,REQ, login(_U,_C1),_C3R, _C3), t2,ℜ(t1,t2)) ∧ _C2 ≠_C3
⇒ ∃ t3: Time **Happens**(e(_E3,_C1,_C2,REQ, logout(_U,_C1), _C2),t3,ℜ(t1+1,t2−1))

```
<action>
   <actionOperationName>NotifySRF</actionOperationName>
   <variable persistent="0" forMatching="false“>
         <varName>instanceId</varName><varType>string</varType>
         <value>/resultsdesc/results/formula [@instanceId]</value>
   </variable>
  <guardCondition negated="false">
  <condition negated="false">
     <greaterThan>
         <operand1><queryOperand>
         <document><name>R5_Result</name><type>MonitoringResults</type></document>
         <xpath>/resultsdesc/results/formula[@minThreatLikelihood]</xpath>
         </queryOperand></operand1>
         <operand2><constant><type>DOUBLE</type> <value>0.6</value></constant>
         </operand2>
     </greaterThan>
  </condition>
 </guardCondition>
</action>
```

Action taken if the overall threat likelihood of *Rule-5* exceeds 0.6

Actions: example 3

Rule-5: $\forall$ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time
Happens(e(_E1, _C1R, _C1, _C2R, _C2,REQ, login(_U,_C1), _C2R, _C2), t1,$\Re$(t1,t1)) $\wedge$
Happens(e(_E2, _C1R , _C1, _C3R, _C3,REQ, login(_U,_C1),_C3R, _C3), t2,$\Re$(t1,t2)) $\wedge$ _C2 $\neq$_C3
$\Rightarrow$ $\exists$ t3: Time **Happens**(e(_E3,_C1,_C2,REQ, logout(_U,_C1), _C2),t3,$\Re$(t1+1,t2−1))

Actions: example 3

Rule-5: $\forall$ _U: User; _C1: Client; _C2, _C3: NetworkController; t1, t2:Time
Happens(e(_E1, _C1R, _C1, _C2R, _C2,REQ, login(_U,_C1), _C2R, _C2), t1,$\Re$(t1,t1)) $\wedge$
Happens(e(_E2, _C1R , _C1, _C3R, _C3,REQ, login(_U,_C1),_C3R, _C3), t2,$\Re$(t1,t2)) $\wedge$ _C2 $\neq$ _C3
$\Rightarrow$ $\exists$ t3: Time **Happens**(e(_E3,_C1,_C2,REQ, logout(_U,_C1), _C2),t3,$\Re$(t1+1,t2−1))

```
<action>
   <actionOperationName>NotifyApplication</actionOperationName>
   <variable persistent="0" forMatching="false“>
               <varName>networkControllerId</varName><varType>string</varType>
               <value>/resultsdesc/results/formula/body/predicate[1]/happens/ic_term/
                        variable[2]/varName[text()=“_C1"]/value</value>
   </variable>
  <guardCondition negated="false">
  <condition negated="false">
     <greaterThan>
               <operand1><queryOperand>
               <document><name>R5_Result</name><type>MonitoringResults</type></document>
               <xpath>/resultsdesc/results/formula/body/predicate[2][@minLikelihood]</xpath>
               </queryOperand></operand1>
               <operand2><constant><type>DOUBLE</type> <value>0.6</value></constant>
               </operand2>
     </greaterThan>
  </condition>
 </guardCondition>
</action>
```

Action taken if the belief in the genuineness of second login is less than 0.4

CITY UNIVERSITY LONDON

Conclusions

- SERENITY provides an infrastructure for selecting and deploying S&D solutions at runtime based on S&D patterns
- It also provides a monitoring framework for runtime checks of conditions related to the correct operation of S&D patterns
- These conditions are specified as monitoring rules in Event Calculus
- Monitoring rules are specified as part of S&D patterns and need to be accompanied by the actions that should be taken when they are violated
- The monitoring infrastructure provides
  - basic monitoring and diagnosis capabilities
  - threat detection capabilities (i.e., detection of potential violations of monitoring rules)

Ongoing work

- Extension of predictive capabilities of EVEREST to support forecasting of violations of aggregate properties (e.g., MTTF, MTTR)
- Extension of EVEREST to support protocols for reliable messaging (WS-ReliableMessaging) and message authentication (WS-Security)
- Support for evolution of S&D solutions both at the pattern and the implementation level

Main resources

- SERENITY Book

  Spanoudakis G., Mana A., Kokolakis S.: Security and dependability for Ambient Intelligence, Advances in Information Security Book Series, Springer, ISBN-978-0-387-88775-3, 2009

- SERENITY Forum

  www.serenity-forum.org

  Includes technical reports, papers, examples of S&D patterns, tutorials e.t.c

Thank you

References (1)

1. Lorenzoli D., Spanoudakis G.: Detection of Security and Dependability Threats: A Belief Based Reasoning Approach , 3rd International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2009), June 2009
2. Spanoudakis G, Kloukinas C. Mahbub K.: The SERENITY Runtime Monitoring Framework, In Security and Dependability for Ambient Intelligence, In *Security and Dependability for Ambient Intelligence*, Information Security Series, Springer, pp. 213-238, 2009
3. Tsigritis T. Spanoudakis G, Kloukinas C. Lorenzoli D.: Diagnosis and Threat detection capabilities of the SERENITY Runtime Framework, In *Security and Dependability for Ambient Intelligence*, Information Security Series, Springer, pp 239-272, 2009
4. Kloukinas C., Spanoudakis G., Mahbub K.: Estimating Event Lifetimes for Distributed Runtime Verification, 20th International Conference on Software Engineering and Knowledge Engineering, 2008
5. Tsigritis T., Spanoudakis G.: Diagnosing Runtime Violations of Security & Dependability Properties, 20th International Conference on Software Engineering and Knowledge Engineering, July 2008
6. Amalio N., Spanoudakis G.: From Monitoring Templates to Security Monitoring and Threat Detection, 2nd International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2008), August 2008
7. Kloukinas C., Spanoudakis G., : A Pattern-Driven Framework for Monitoring Security and Dependability , 4th International Conference on Trust, Privacy and Security in Digital Business (TrustBus`07), Lecture Notes in Computer Science 4657/2007, DOI: 10.1007/978-3-540-74409-2_23, September 2007

References (2)

7. Tsigkritis T. Spanoudakis G, Kloukinas C. Lorenzoli D.: Diagnosis and Threat detection capabilities of the SERENITY Runtime Framework, In *Security and Dependability for Ambient Intelligence*, Information Security Series, Springer, pp 239-272, 2009
8. Sanchez-Cid F. , Mana A., Spanoudakis G., Serrano D., and Munnoz A.: Representation of Security and Dependability Solutions, In *Security and Dependability for Ambient Intelligence*, Information Security Series, Springer, pp. 69-96, 2009
9. Androutsopoulos K., Ballas K., Kloukinas C., Mahbub K., and Spanoudakis G, "V1 of Dynamic Validation Prototype", Deliverable A4.D3.1, SERENITY Project. Available from http://www.serenity-forum.org/IMG/pdf/A4.D3.1_dynamic_validation_prototype_v1.2_final.pdf, 2006
10. Mahbub K., Spanoudakis G., Kloukinas C. (2007): "V2 of dynamic validation prototype". Deliverable A4.D3.3, SERENITY Project, Available from: http://www.serenity-forum.org/IMG/pdf/A4.D3.3_-_V2_of_Dynamic_validation_Prototype.pdf, 2007
11. Spanoudakis G., Tsigkritis T. : "1st Version of Diagnosis Prototype". Deliverable A4.D5.1, SERENITY Project, Available from: http://www.serenity-forum.org/IMG/pdf/A4.D5.1_first_version_of_diagnosis_prototype_v1.1_final.pdf, 2008
12. Amalio N., DiGiacomo V., Kloukinas C., Spanoudakis G.: "Mechanisms for detecting potential S&D threats". Deliverable A4.D4.1, SERENITY Project, Available from: http://www.serenity-forum.org, 2008
13. Li K, et. Al: "Scenario S&D solutions v1", Deliverable A7.D4.2, SERENITY Project, Available from: http://www.serenity-forum.org, 2008