A Model-Driven Approach to Situations: Situation Modeling and Rule-Based Situation Detection

Patrícia Dockhorn Costa
Izon Thomas Mielke
Isaac Pereira
João Paulo A. Almeida

jpalmeida@ieee.org
http://nemo.inf.ufes.br

Computer Science Department
Federal University of Espírito Santo
Motivation

• Situation awareness increasingly important
• Pushes the notion of context-awareness further
• Several areas of application:
  – in autonomic computing
  – in human-computer interaction
  – in context-aware business process modeling
• Situation awareness requires the ability to perceive facts and to identify in these the patterns that characterize situations of interest.
• Detect situation of interest and react accordingly
Motivation

• Despite the importance of situation awareness, lack of support for situation specification (and detection)

• Approaches end up composing low-level events instead of focusing on the higher-level patterns that emerge in relevant situations
  – E.g., consider the situation in which a patient has fever

• Situation types are often reduced to logic propositions
  – Failing to address properties of situations (such as duration) and temporal relations between situations
Approach

- Define a domain-specific language for situations (SML)
- Assume an existing context model that determines the type of entities which exist and their relations
  - UML class diagram
- Use visual patterns in SML for situation specification
- Reify situations and include temporal operators to relate situations
- Compose complex situations from simpler ones
- Synthetize a rule-based implementation for situation detection based on the situation types modeled in SML
  - Model-2-text transformation, generating Drools code
Banking Scenario: Context Model

![Diagram of Banking Scenario: Context Model](image)
Logged in situation
On-going Suspicious Withdrawal
Suspicious Parallel Login

Diagram:

- SuspiciousParallelLogin
- LoggedIn
- account
  - overlaps
  - equals
- account
  - account
Suspicious Parallel Login

Figure 2 depicts the definition of the LoggedIn situation. Figure 3 defines the OngoingSuspiciousWithdrawal situation. In this section we discuss the concept of composite situations.

A composite situation is defined as a situation that encompasses two or more situations which must overlap in time. SML allows for the composition of situations (for the same account), in which the first occurrence must have finished by the time the second occurs. All relations hold simultaneously.

Figure 4 shows an example of a composite situation being defined. These bordering diamonds represent the entities participating in this situation type, which captures the situation type, graphically represented as a rounded rectangle. The other directed arrow labeled overlap points to the entities that are neighbors (for the same account). Note that the situation can be represented with orange elements composing a particular situation type, graphically represented as a rounded rectangle. The stent with names indicates additional constraints. The second directed arrow labeled equals points to the entities that are neighbors (for the same account), in which we define the situation type.

Figure 5 depicts an example timeline for the SuspiciousParallelLogin situation. Figure 6 depicts a more complex example of the啊情况.
Suspicious Far Away Login

Diagram:
- **SuspiciousFarawayLogin**
  - **LoggedIn**
    - device = Device
    - account
    - before[0s, 2h] = equals
  - **Location**
    - not near[500km]

- **Device**
  - location

- **LoggedIn**
  - device
  - account
OngoingSuspiciousWithdrawal indicates that any instance of SuspiciousFarawayLogin ceases to exist. This temporal relation is specified by the direct formal relation

\[ \text{OngoingSuspiciousWithdrawal} \quad (\text{non-empty}) \text{ set } S \]

which is parameterized with lower and upper time limits. In this example, the situation begins to exist simultaneously with the first instance of SuspiciousFarawayLogin and stretches for 30 days after the second occurrence of situation LoggedIn. When the second occurrence of situation SuspiciousFarawayLogin is no longer said to exist, the situation begins to cease at most 2 hours earlier than the second occurrence of situation SuspiciousFarawayLogin.

This temporal relation is expressed as:

1. The situation type SuspiciousFarawayLogin is represented by the model with the oval shapes.
2. The properties of entities participating in the nested situation, which are participating in the nested situation type, should not be near each other, at the time the relation is expressed. In this case, the withdrawal must have occurred in the past 30 days.
3. The consequence of this temporal relation is that the time window stretches for more than 30 days. The 30 days window can be extended to account for the time at which the second occurrence of situation SuspiciousFarawayLogin is no longer said to exist.
4. Figure 8 depicts the situation where the situation type SuspiciousFarawayLogin is parameterized with lower and upper time limits (between 0 and 2 h).

V. Frame-based Model

We discuss here a formal semantics for SML. Our purpose is to define the language of SML. We assume a Kripke-style frame-based model to define the semantics of SML. The model represents what exists at a particular point in time. The semantics for SML is characterized by the formal relation

\[ \text{OngoingSuspiciousWithdrawal} \quad (\text{non-empty}) \text{ set } S \]

between worlds; accessibility reflects changes from one world to another. In this case, the withdrawal must have occurred within the past 30 days.
Account Under Observation

∃ OngoingSuspiciousWithdrawal

Account

Time

finalTime

within the past

30 days
Account Under Observation
Example Timeline

- **AccountUnderObservation**
- **OngoingSuspiciousWithdrawal**
- **30 days window**
- **30 days window**
Any Account Under Observation

AnyAccountUnderObservation

\exists \text{ OngoingSuspiciousWithdrawal}

\text{finalTime} \rightarrow \text{within the past} \rightarrow 30 \text{ days}
### Supported temporal relations between situations

<table>
<thead>
<tr>
<th></th>
<th>Active - Active</th>
<th>Active - Inactive</th>
<th>Inactive - Inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Before B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B After A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A Meets B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B Met by A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A Overlaps B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B Overlapped by A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A Finishes B</td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>B Finished by A</td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>A Includes B</td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>B During A</td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>A Starts B</td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>B Started by A</td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>A Coincides B</td>
<td></td>
<td></td>
<td>A</td>
</tr>
</tbody>
</table>
Formal Semantics

The entities, their context (at instance-level) and their dynamics are represented by a Kripke-style frame-based model with the following elements:

− A (non-empty) set $W$ of worlds, with each world $w$ representing what exists at a particular point in time.
− A (non-empty) set $U$ of all possible entities and context elements (instances of the classes in the context model).
− A (non-empty) set $S$ of all possible situations (instances of situation types in the SML model).
− A binary relation $R$, representing the accessibility between worlds
− A function $\text{time}(w)$
Simple Situation Type

Loggedln(w, s, d, a, ac) iff
(Device(w, d) ∧ Account(w, a) ∧ Access(w, ac) ∧
isAccessing(w, d, ac) ∧ isAccessed(w, a, ac) )

(we should admit as an axiom the universal closure of the formula above)
Simple Situation Type

A simple situation type is interpreted as an *open sentential formula* formed by a conjunction of terms, each of which corresponds to an element in a situation type.

Free variables to represent:
- the world $w$ in which the situation exists
- the situation itself
- each entity and relational context element in the situation.

Loggedln($w$, $s$, $d$, $a$, $ac$) iff

(Device($w$, $d$) $\land$ Account($w$, $a$) $\land$ Access($w$, $ac$) $\land$
isaAccessing($w$, $d$, $ac$) $\land$ isAccessed($w$, $a$, $ac$))

(we should admit as an axiom the universal closure of the formula above)
Important axioms

There is a unique situation for each “binding” of participants in a world:

\[ \forall w \in W, \forall s \in S, \forall s' \in S, \forall d \in U, \forall a \in U, \forall ac \in U, \]

\[ ((\text{LoggedIn}(w, s, d, a, ac) \land \text{LoggedIn}(w, s', d, a, ac)) \rightarrow s=s') \]

If the binding remains stable in subsequent worlds, the situation is also the same:

\[ \forall w \in W, \forall w' \in W, \forall s \in S, \forall s' \in S, \forall d \in U, \forall a \in U, \forall ac \in U \]

\[ ((w \ R \ w') \land \text{LoggedIn}(w, s, d, a, ac) \land \]

\[ \text{LoggedIn}(w', s', d, a, ac)) \rightarrow s=s'). \]
Formal Semantics

Diagram showing the relationship between four sets of elements (w1, w2, w3, w4) connected by relations s1 and s2.
Complex Situations

- Same rules discussed for simple situation types
- Composing situation are just like other participants
- Each composition situation adds a free variable

\[
\text{SuspiciousParallelLogin}(w, s, s', s') \iff (\text{LoggedIn}(w, s, d, a, ac) \land \text{LoggedIn}(w, s', d', a', ac') \land (s \neq s') \land \text{overlaps}(s, s') \land (a = a')).
\]
Situation Detection in Drools
Logged In Situation Rule

rule "LoggedInRule"

when
  account : Account()
  device : Device()
  $access1 : Access(isAccessed == account,
                  isAccessing == device)

then
  SituationHelper.situationDetected(drools,
                                     LoggedInSituation.class);

End

• Helper introduces a “logical fact” (CurrentSituation) that is maintained while the left-hand side is true
• Additional (generic) rule is used to keep past situations
Deactivation Rule to Maintain Records of Past Situations

- Deactivation rule deactivates the situation when the conditions for the CurrentSituation logic fact are no longer applicable.
- The RHS then updates the related SituationType instance state to non-active (a past situation).

```
rule "SituationDeactivation"
when
  $sit: SituationType(active == true)
  not (exists CurrentSituation(situation == $sit))
then
  SituationHelper.deactivateSituation(drools, (Object) $sit);
end
```
Ongoing Suspicious Withdrawal Rule

rule "OngoingSuspiciousWithdrawalRule"

when

   $account1 : Account()
   $atm1 : ATM()
   $ongoingwithdrawal1 : OngoingWithdrawal(
       value > new Monetary("$1000"),
       hasWithdrawal == $account1,
       doWithdrawal == $atm1)

then

   SituationHelper.situationDetected(drools,
       OngoingSuspiciousWithdrawalSituation.class);

end
Suspicious Parallel Login Rule

rule "SuspiciousParallelLoginRule"
when
  $loggedin1 : LoggedInSituation( $loggedin1_account:account)
  exists(CurrentSituation ( situation == $loggedin1))
  $loggedin2 : LoggedInSituation(
    this overlaps $loggedin1,
    account == $loggedin1_account)
  exists(CurrentSituation ( situation == $loggedin2))
then
  SituationHelper.situationDetected(drools,
    SuspiciousParallelLoginSituation.class);
end
Mappings between SML constructs and Drools Constructs

<table>
<thead>
<tr>
<th>SML Constructs</th>
<th>Drools Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Situation Type</td>
<td>Drools rule (and a Java Class representing the Situation Fact Template)</td>
</tr>
<tr>
<td>Entity types, Relational Context types, and nested</td>
<td>Rule patterns</td>
</tr>
<tr>
<td>Situations types</td>
<td></td>
</tr>
<tr>
<td>Intrinsic Context types</td>
<td>Pattern constraints</td>
</tr>
<tr>
<td>Relations (formal, temporal and material)</td>
<td>Pattern restrictions (with operators)</td>
</tr>
</tbody>
</table>
Related Work

• Several approaches to situation definition
  – Learning-based vs specification-based
• Specification-based approaches often specify situations in terms of logical expressions or formal ontologies.
  – Devlin, Heckmann, Kokar et al, Yau & Liu, Dockhorn-Costa et al.
  – Either platform-specific or make use of general-purpose languages, such as OWL and OCL
  – Do not offer primitive situation constructs. Many of these languages still lack expressiveness with respect to situation composition and temporal reasoning.
• Many approaches offer reactive query interfaces instead of detecting situations attentively
  – Hang Wang, Henricksen & Indulska, Strang et al.
Concluding Remarks

• A graphical language for situation specification (SML)
• A model-driven approach to synthesize situation detection on top of Drools Situations

• Formal semantics described in the paper
• Ecore metamodels described in tech report
  – https://github.com/izontm/SML
• Editor for SML implemented in Obeo Designer and transformations in Acceleo

• Future work:
  – Analyze performance of situation detection
  – Investigate approach for the semantic integration of heterogeneous context sources
SML Metamodel
Node
Participant