Modeling, Specifying, Discovering, and Integrating Trust into Distributed Real-time and Embedded (DRE) Systems

Lahiru S. Gallege, Dimuthu U. Gamage, James H. Hill, Rajeev R. Raje
{lspileth, dcundupi, hillj, rraje}@cs.iupui.edu
Presentation Outline

- Background and Existing Approaches
- Proposed Approach
  - Initial Trust Model Proposal
- Case Study
  - Distributed Tracking System
- Conclusions
- Future Work
Overview of Trust

- Trust is critical in the development of DRE systems
- Trust is also highly subjective!
  
  "Trust is choosing of an ambiguous path by the truster which could be good or bad, where the occurrence of good or bad is a result of the action performed by another person." – Marsh et al. (1994)

- "Trust can be simplified to two important factors reputation, i.e. recommendation and competence," – Wang et al. (2009)

- "An entity can be trusted if it always behaves in the expected manner for the intended purpose." – Trusted Computing Group (Pearson et al. 2002)

- And many more…

- "Trust of a software service is its conformance to a published specification”
  
  Includes both functional and QoS aspects.
Trust Metrics in the Literature

Perform, Agents, Agent, ..., (12)
Subjective (11)
Method (9)
Context (9)
Distributed (10)
Trust Management (12)
Trustee (18)
Truster (12)
Services (10)
Security (14)
Peer, Peer-to-Peer (6)
Trust Defined (18)

- Perform, Agents, Agent, Monitor (12)
- Trust Management (12), Services (10)
- Trustee (18), Truster (12)
- Security (14)
- Distributed (10), Systems (12)
- Subjective (11), Method (9)
- Trust is Defined (18)
- Peer, Peer-to-Peer (6)
- Context (9)
Limitations of Existing Approaches

- Despite its criticality, trust is not considered as an integral part of software development life-cycle, but as an afterthought.

- Existing trust metrics do not use inherent properties or the evidences from intermediate software lifecycle phases.

- Existing trust models do not provide a generalized trust model that is parameterizable by trust attributes (e.g., ratings, feedback of a deployed system and test coverage, and coding conventions).
Our View of Trust

*Artifacts are conceptual or physical outcomes of a particular phase of software lifecycle

Prevalent Approaches

Proposed Approach

Requirements Artifacts
Design Artifacts
Development Artifacts
Testing Artifacts
Integration Artifacts
Deployment Artifacts

RUST
Proposed Approach

- Trust is represented using Subjective Logic proposed by Jøsang at al.
  - Trust as a tuple of (Belief, Disbelief, Uncertainty)
- Trust of an artifact is measured from both internal and external views
  - **Internal View:** Inherent properties of the artifacts
    - E.g., who made it (source),
    - what it is made of (factors),
    - and how it is made (process)
  - **External View:** External user experience of the artifact
- The policy defines the significance of views and associated evidences
## Trust of Artifacts

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Phase</th>
<th>Internal View Properties</th>
<th>External View Properties</th>
<th>Trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Requirement Specification</td>
<td>Requirements</td>
<td>Requirement engineers</td>
<td>Use cases, State charts, Data flow diagrams</td>
<td>Formality, Complying with standards</td>
</tr>
<tr>
<td>(2) Design Specification</td>
<td>Design</td>
<td>Software engineers</td>
<td>Class diagrams, Component diagrams, Requirements</td>
<td>Formality, Complying with design principles, standards</td>
</tr>
<tr>
<td>(3) Classes/Components</td>
<td>Development</td>
<td>Software Developers</td>
<td>Functions, Source code, Code Coverage, Design</td>
<td>Coding Convention, Correctness proofs</td>
</tr>
<tr>
<td>(4) Test Oracle, Test Suites</td>
<td>Testing</td>
<td>Testers</td>
<td>Test scripts, Test cases</td>
<td>The unit test</td>
</tr>
<tr>
<td>(5) Integrated system</td>
<td>Integration</td>
<td>Integration Engineers</td>
<td>Classes/Components, Integration Framework</td>
<td>Integration methods</td>
</tr>
<tr>
<td>(6) Deployed service</td>
<td>Deployment</td>
<td>Deployment Engineers</td>
<td>Integrated system, Deployment Framework</td>
<td>Deployment Methods</td>
</tr>
</tbody>
</table>

- Trust is defined as the tuple: $T_i = (B_i; D_i; U_i)$.
- Trust of internal view of an artifact – $T_{iv(pa)}$ is therefore defined as the collection of $(B;D;U)$ tuples of internal properties ($iv_{pa} (B;D;U)$).

- Similarly $T_{xv(pa)}$ is defined as $(xv_{pa} (B;D;U))$.

- Overall trust of the artifact $T_a$ is defined as:
  $T_a = f (T_{iv(pa)}, T_{xv(pa)})$.
Development Artifacts:
- Modeling trust while developing (writing code), at the many levels, including:
  - Statement Level
  - Functional Level
  - Component Level
  - Services Level

Deployment Artifacts:
- Specifying trust of deployed artifacts using multi level specifications.
- Modeling trust of a composition of software services.
Trust Contract

➤ Trust of the service represented in the “Trust Attributes” using Multi-level software specification.

```xml
<Deployment Artifact Multi-level Contract>
  <ComponentAttributes> ... <ComponentAttributes> (A)
  <FunctionalAttributes> ... <FunctionalAttributes> (B)
  <Non-functionalAttributes> ... <Non-functionalAttributes> (C)
  <SynchronizationAttributes> ... <SynchronizationAttributes> (D)
  <Trust Attributes>
    <InternalView>
      <FunctionalTrustAttributes>
        ... <Developers’ views of trust of (reference B)>
      <FunctionalTrustAttributes>
      <Non-functionalTrustAttributes>
        ... <Developers’ views of trust of (reference C)>
      </Non-functionalTrustAttributes>
    </InternalView>
    <ExternalView>
      <FunctionalTrustAttributes>
        ... <Users’ views of trust of (reference B)>
      <FunctionalTrustAttributes>
      <Non-functionalTrustAttributes>
        ... <Users’ views of trust of (reference C)>
      </Non-functionalTrustAttributes>
    </ExternalView>
  </Trust Attributes>
</Deployment Artifact Multi-level Contract>
```
## Trust Rules for Composition Patterns

<table>
<thead>
<tr>
<th>Interaction Pattern</th>
<th>Quality Attribute(q)</th>
<th>Quality of Composition(Qs)</th>
<th>Trust of Composition T(Qs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequence</strong></td>
<td>Response Time</td>
<td>Qs = Qs₁ + Qs₂</td>
<td>T(Qs) = T(Qs₁) ^ T(Qs₂) *</td>
</tr>
<tr>
<td></td>
<td>Availability</td>
<td>Qs = Qs₁ * Qs₂</td>
<td>T(Qs) = T(Qs₁) ^ T(Qs₂)</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
<td>Qs = Qs₁ * Qs₂</td>
<td>T(Qs) = T(Qs₁) ^ T(Qs₂)</td>
</tr>
<tr>
<td><strong>Parallel Split/Join</strong></td>
<td>Response Time</td>
<td>Qs = max(Qs₁, Qs₂)</td>
<td>T(Qs) = T(Qsₛ) ; where sₓ = s₁ or s₂,</td>
</tr>
<tr>
<td></td>
<td>Availability</td>
<td>Qs = Qs₁ * Qs₂</td>
<td>Qₛₓ = max(Qs₁, Qs₂)</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
<td>Qs = Qs₁ * Qs₂</td>
<td>T(Qs) = T(Qs₁) ^ T(Qs₂)</td>
</tr>
<tr>
<td><strong>Exclusive choice</strong></td>
<td>Response Time</td>
<td>Qs = Wₛ₁. Qs₁ + Wₛ₂.Qs₂</td>
<td>T(Qs) = Wₛ₁. T(Qs₁) + Wₛ₂.T(Qs₂)</td>
</tr>
<tr>
<td></td>
<td>Availability</td>
<td>Qs = Wₛ₁. Qs₁ + Wₛ₂.Qs₂</td>
<td>T(Qs) = Wₛ₁. T(Qs₁) + Wₛ₂.T(Qs₂)</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
<td>Qs = Wₛ₁. Qs₁ + Wₛ₂.Qs₂</td>
<td>T(Qs) = Wₛ₁. T(Qs₁) + Wₛ₂.T(Qs₂)</td>
</tr>
<tr>
<td><strong>Discriminator</strong></td>
<td>Response Time</td>
<td>Qs = min(Qs₁, Qs₂)</td>
<td>T(Qs) = T(Qsₛ) ; where sₓ = s₁ or s₂,</td>
</tr>
<tr>
<td></td>
<td>Availability</td>
<td>Qs = Qs₁ * Qs₂</td>
<td>Qₛₓ = min(Qs₁, Qs₂)</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
<td>Qs = Qs₁ * Qs₂</td>
<td>T(Qs) = T(Qs₁) ^ T(Qs₂)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Qs = Qs₁ * Qs₂</td>
<td>T(Qs) = T(Qs₁) ^ T(Qs₂)</td>
</tr>
<tr>
<td><strong>Loop</strong></td>
<td>Response Time</td>
<td>Qs = Qs₁.n</td>
<td>T(Qs) = T(Qs₁) ^ ... T(QS₁) (n times)</td>
</tr>
<tr>
<td></td>
<td>Availability</td>
<td>Qs = Qs₁.n</td>
<td>T(Qs) = T(QQ₁) ^ ... T(QS₁) (n times)</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
<td>Qs = Qs₁^n</td>
<td>T(Qs) = T(QQ₁) ^ ... T(QS₁) (n times)</td>
</tr>
</tbody>
</table>

* \( b^{p.q} = (b_p.b_q) \); (d^{p.q} = d_p + d_q - d_p.d_q) ; (u^{p.q} = b_p.u_q + u_p.b_q + u_p.u_q) *
Case Study – Distributed Tracking System

- The system's main goal is to accurately predict an object location.
- How to specify trust attributes of a deployed system?
- How to compose the system while meeting requirements, including trust?
Case Study : First Step

- Identification of the Interaction Patterns in the Tracking System
Simplification of Interaction Patterns

1. Parallel Split/Join

2. Sequence

3. Loop
Trust of a ‘Quality Attributes’

- Application of Trust composition rules for the response time to adhere to real-time constraints

- E.g., Response time($Q_s$) of the composition of $S_1$, $S_2$, $S_3$ (parallel and join)
  - $Q_s = \max(Q_{s_1}, Q_{s_2}, Q_{s_3})$

- Trust of the response time attribute ($T(Q_s)$)
  - $T(Q_s) = T(S_x)$ where $S_x$ has the maximum response time
Conclusions

- Representation of Trust using principles of subjective logic.
  - Extending the (B, D, U) model in representing trust.

- Association of the Trust model throughout the software life cycle
  - Consider ‘Internal View’ of a Service, along with the “External View”, while evaluating the trustworthiness of a Service.
  - Identify trust metrics and policies

- Trust attributes as a new level of contract in the multi-level specifications.

- Trust composition rules based upon common interaction patterns of software services could provide the overall trust of a composed system.
Future Directions

- Investigation of trust composition rules at the function and service levels
  - Trust-by-Construction
  - Extending Weakest pre-condition rules to derive trust of software services

- Modeling of Trust using Tools such as GME

- Designing a Trusted Discovery Service

- Empirical validation of the trust model by developing prototypical trust-aware DRE systems.
Thank You!
Any Questions/Comments?