Towards a CBSE Formal Approach for Developing Trustworthy Systems

Mubarak Mohammad

http://users.encyclopedia.concordia.ca/~ms_moham/
ms_moham@cse.concordia.ca
An Era of Ubiquitous Computing

• Networked computers with sensors and actuators are embedded into all kinds of daily devices and places.

To which extent the current software development paradigms are capable of producing trustworthy systems that control the lives of people and manage their private data?
Trustworthiness

• We depend on technology in our daily life although it fails from time to time causing catastrophic consequences.
• There is no “absolute” trust.
• Properties of trust must be defined and examined.
Trustworthiness

- Trust is a social issue that is hard to define formally.

How these properties can be satisfied collectively in one development process?, and whether the current state of the art of software development paradigms can address their requirements?
CBSE

• CBSE promises many advantages to software development including reuse, managing complexity, and reducing the development’s time, effort, and cost.

• The essential constituents of CBSE are:
  1. components engineering,
  2. component model, and
  3. development process model.
<table>
<thead>
<tr>
<th>Definition</th>
<th>State of the arts with reference to trustworthiness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component Engineering</strong></td>
<td>addresses the issues of component’s specification, development, qualification, documentation, cataloguing, adaptation, and selection for reuse.</td>
</tr>
<tr>
<td><strong>Component Model</strong></td>
<td>defines what constitutes a component and how it can be composed. It addresses the issues of assembling components to develop component-based systems</td>
</tr>
<tr>
<td><strong>Development Process Model</strong></td>
<td>addresses the activities involved throughout the entire component and system lifecycle</td>
</tr>
</tbody>
</table>
Contributions

1. A formal component–based approach that collectively addresses the requirements of trustworthiness, reusability, and CBD;

2. A uniform architecture description language capable of expressing structural, functional, and nonfunctional requirements;

3. A formal development process model that describes component engineering and CBD;

4. A framework with a comprehensive set of tools that support the formal development process.
Uniform Component Model

System Specification

Safety

Security

Reliability

Availability

Analysis Result

Trustworthiness

Real-time

Component-Based
Agenda

• Basic definitions of the essential elements of a trustworthy component model;
• TADL – Architecture description language;
• Meta-Architecture;
• Verification of trustworthiness properties.
• Q & A Discussion
Meta Architecture

- An architecture type that comprises structural, functional, and non-functional elements.
TADL

• An architecture description language.
• Every architectural element is described separately to increase reuse of existing specifications.
What do we want from a Component?
Significance of Modeling Services

• The explicit specification of services at architectural level enables:
  – Regulating services by defining timeliness requirements for service processing
  – Restricting services by defining constraints to ensure safe and correct behaviour.
  – Filtering services according to security specification.
  – Basis for component classification and discovery.
  – Basis for automatic test scenarios
Types of Services

• Input Request
• Output Response
• Output Request
• Internal Processing
Service-Centric Component Model

• Components provide and request services via public interfaces.
• A service model a functionality provided or required by a component.
• Service calls are parameterized.
• A data parameter is a variable value passed to component within a request for service or passed from the component within a provided service.
Significance of modeling data parameters

• Allows modeling different types of simple and complex data communicated at the interfaces of a component. (rich comm. Specs.)

• Provide mechanism for securing information communicated through the interfaces of a component.

• Enables rich specification of safety contracts by regulating reactions of the component based on values of data parameters.

• Basis for building test data automatically.
Service Type Definition

• A service type definition includes:
  – Name
  – Data parameters
  – Attributes (ex. Priority)
  – Constraints (ex. invariants)
  – Direction (Input-Request, Output-Response, Output-Request, Internal)

• A data parameter type definition includes:
  – Name
  – Data type
  – Default value
  – Constraint
ParameterType <name> { 
    DataType <Type> ;
    Default <value> ;
    Constraint <FOPL> ;
}

Attribute <name> { 
    DataType <Type> ;
    Default <value> ;
    Constraint <FOPL> ;
}

ServiceType <name> { 
    (Attribute <Type> <name> ) * ;
    (Constraint <FOPL> ) * ;
    (Parameter <Type> <name> ) * ;
    Direction <direction> ;
}
Example

Attribute **Priority**{
    DataType integer ;
    Default 1;
}

Attribute **WCET**{
    DataType integer ;
    Default 15;
}

ParameterType **Temperature**{
    DataType float ;
    Constraint value >=-25 \ value <=100;
}

ServiceType **Control_Temperature** {
    Parameter Temperature temp;
    Priority p;
    WCET w;
    Direction Input-Request;
    Constraint p=1 \rightarrow w <120;
}
Predictability Oriented Component Model

- In order to trust, the behaviour must be predictable.

The association between requests and provides services is defined using *reactivity*.
Reactivity

- Reactivity is a relation between requests for services and their corresponding responses.
- Reactions can be ...
  - restricted using Data Constraints.
  - regulated using Time Constraints.
  - filtered using Security Mechanism.
  - Controlled by other non functional requirements can be stated using attributes and constraints.
Data Constraints

• A special type of constraint that is used to decide whether or not a specific response for a requested services should be sent.
• The definition comprises:
  – Two services
  – Constraint
  – attributes
• The constraint uses values of:
  – Data parameters
  – Attributes of:
    • The requested service
    • The interface through which the service is provided
    • The component that provides the service
DataConstraint <name> { 

  (Attribute <Type> <name>)*;

  ServiceType <request-name>;
  ServiceType <response-name>;

  RequestService(request-name);
  ResponseService(response-name);

  Constraint <FOPL>;

}
Time Constraint

• A special type of constraint that specifies the maximum safe amount of time allowed to elapse between the arrival of the request and issuing the proper response.

• The definition includes:
  – Two services
  – Attributes
  – Maximum safe time
TimeConstraint <name> { 

  (Attribute <Type> <name>)*; 

  ServiceType <request-name>; 
  ServiceType <response-name>; 

  RequestService(<request-name>); 
  ResponseService(<response-name>); 

  (Constraint <FOPL>)*; 

  float Max_Safe_Time; 

}
Reactivity Definition

• Two service types
• Data constraint
• Time constraint
• Update statements (actions committed after response to set attribute values or trigger other service calls)
• Attributes
Reactivity <name> {

(Attribute <Type> <name>)*;

ServiceType <request-name>;
ServiceType <response-name>;

RequestService(<request-name>);
ResponseService(<response-name>);

DataConstraint <name>;
TimeConstraint <name>;

(Update <statements>)*;

}

Mubarak Mohammad
Prague 2008
Example

Response should occur within 45 units of time
Example

DataConstraint dc_Raise {
    ServiceType Control_Temperature;
    ServiceType Raise;
    RequestService(Control_Temperature);
    ResponseService(Raise);
    Constraint Control_Temperature.temp <= 20;
}

DataConstraint dc_Lower {
    ServiceType Control_Temperature;
    ServiceType Lower;
    RequestService(Control_Temperature);
    ResponseService(Lower);
    Constraint Control_Temperature.temp > 20;
}
**TimeConstraint** \( tc_{\text{Raise}} \) {

ServiceType \textit{Control\_Temperature};
ServiceType \textit{Raise};
RequestService(\textit{Control\_Temperature});
ResponseService(\textit{Raise});
float \textit{Max\_Safe\_Time} = 45;
}

**Reactivity** \textit{Control\_Raise} {

ServiceType \textit{Control\_Temperature};
ServiceType \textit{Raise};
RequestService(\textit{Control\_Temperature});
ResponseService(\textit{Raise});
DataConstraint \textit{dc\_Raise};
TimeConstraint \textit{tc\_Raise};
}

Safety Property

• Invariant over the behaviour of a component.

• Safety property is a special type of constraint defined over a set of services provided by the component.
Example

```
SafetyProperty Safety {

    ServiceType Control_Temperature;
    ServiceType Raise;
    ServiceType Lower;

    Property Control_Temperature.temp<=20 → Raise
          Control_Temperature.temp>20 → Lower

}
```
Safety Contract

• Comprises:
  – Reactivity definitions
  – Attributes
  – Safety property
  – Constraints

• Defined separately to allow reuse and reconfiguration without changing component specification.
ContractType <name> { 

  (Reactivity <name>)+;
  
  (SafetyProperty <name>)*;
  
  (Attribute <name>)*;
  
  (Constraint <name>)*;

}
Structural Definitions

• Defining structure is an important part of CBD.
• It includes interface types, connector types, connector role types, architecture types, and component types.
Interface Type

- An enumerated type whose elements are services.
- A service can be provided by only one interface type.
- Direction depends on defined services.
- Two interface types are compatible if they define the same service types with opposite directions.
- The definition comprises:
  - Name
  - Services
  - Attributes
  - Constraints

```c
InterfaceType <name> {
    (ServiceType<name>)*;
    (Constraint<FOPL>)*;
    (Attribute <name>)*;
}
```
Connector Type

- Abstracting the connector role type from the connector type helps in abstracting the communication method from the access point.
**ConnectorRoleType** `<name>` {  
  (Attribute `<name>`)*;  
  (Constraint<FOPL>)*;  
  InterfaceType `<name>`;  
}

**ConnectorType** `<name>` {  
  (Attribute `<name>`)*;  
  (Constraint<FOPL>)*;  
  (ConnectorRoleType `<name>`)++;  
}
Architecture Type

• Defines the structure of a composite component.
• The constituent components and their inner connectors are specified.
• The definition comprises:
  – Connector types
  – Attachment specification
  – Attributes
  – Constraints
• It is defined outside a component type, independently.
ArchitectureType <name> { 

( Attribute <name>)*;
( Constraint<FOPL>)*;
( ConnectorType <name>)*;

Attachment ( ConnectorType.ConnectorRoleType.InterfaceType,
ConnectorType.ConnectorRoleType.InterfaceType)*;

}
Component Type

• The definition comprises
  – Interface types
  – Architecture type
  – Contract
  – Attributes
  – Constraints

• If no architecture type is specified then it is a primitive component type.

• The list of non attached interface types form the interface types of the composite component, whereas the attached ones form the internal interface types.
ComponentType <name> {

    (Attribute <name>)*;

    (Constraint<FOPL>)*;
    User u;

    (InterfaceType <name>)*;

    (ServiceType <name>)*;

    (ArchitectureType <name>)*;

    ContractType <name>;
}

Security Mechanism

• Role-base Access Control (RBAC).
Security Mechanism

• Every component has a default attribute of type User. This attribute is set at instantiation time.

• Functions:
  – User-group-assignment
  – User-role-assignment
  – Role-group-assignment
  – Privilege-service-assignment
  – Privilege-data-assignment
User <name> {  
  (Attribute <name>)*;  
  (Constraint<FOPL>)*;  
}

Group <name> {  
  (Attribute <name>)*;  
  (Constraint<FOPL>)*;  
}

Role <name> {  
  (Attribute <name>)*;  
  (Constraint<FOPL>)*;  
}

Privilege <name> {  
  (Attribute <name>)*;  
  (Constraint<FOPL>)*;  
}

RBAC <name> {  
  (User <name>)*;  
  (Group <name>)*;  
  (Role <name>)*;  
  (Privilege <name>)*;  

  (User-Group-Assn ( User , Group ))*;  
  (Role-Group-Assn ( Role , Group ))*;  
  (User-Role-Assn ( User , Role ))*;  

  (ServiceType <name>)*;  
  (ParameterType <name>)*;  

  (Privilege-for-Services (ServiceType, Privilege, Role))*;  
  (Privilege-for-Parameters (ParameterType, Privilege, Role))*;  
}

Mubarak Mohammad

Prague 2008
Security access Functions

- User-Service-Access( User, ServiceType) : returns Privilege
- User-Has-Privilege( User, ServiceType) : returns Boolean
- User-Data-Access( User, DataParameter) : returns Privilege
- User-Has-Role( User, Role): returns Boolean
- Get-User-Roles (User) : returns list of Roles
- Get-User-Privileges (User) : returns list of Privileges
- Get-Role-Privileges (Role) : returns list of Privileges
- Role-Service-Access( Role, ServiceType) : returns Privilege
- Role-Data-Access( Role, DataParameter) : returns Privilege
- .... Any other function
Security Properties

• Service Security:
  – For every service request received, the request should come from a user who has permission to request the service.
  – For every provided service, the user to whom the service is provided should have permission to receive the service.

• Data Security:
  – For every service request, for every data parameter, the user sending the request should have permission to access the data parameter.
  – For every provided service, for every data parameter associated with it, the user who will receive the service should have permission to access the data parameter.

• If the user doesn’t have permission the service request will be ignored, the service won’t be provided, or the data parameter will be filtered and set to null value.
ServiceSecurity <name> {  
    ServiceType <name>;  
    (Role <name>);  
    (User <name>);  
    Security Constraint;  
}

DataSecurity <name> {  
    DataParameter <name>;  
    (Role <name>);  
    (User <name>);  
    Security Constraint;  
}

ServiceSecurity <Role-Control-Temperature> {  
    ServiceType Control-Temperature;  
    User Operator;  
    Security Constraint User-Has-Privilege(Operator, Control-Temperature)==true;  
}
Reactivity <name> { 

(Attribute <Type> <name>)*; 

ServiceType <request-name>; 
ServiceType <response-name>; 

RequestService(request-name); 
ResponseService(response-name); 

(DataConstraint <name>); 
(TimeConstraint <name>); 
(Update <statements>)*; 

(ServiceSecurity <name>); 
(DataSecurity <name>); 

}
System Definition

- Hardware components are defined and used as deployment units.
- They can have attributes to specify hardware capabilities and constraints to control deployment.
- Configuration specification instantiate the system elements and specify deployment.
Configuration<name> { 
  (SystemElement )+;
  Deploy( HardwareComponentType, ComponentType; 
}
Verification of Trustworthiness

TADL Specification → Model Transformation → Behavior Protocol → Model Checking

Model Transformation → Real-time Model → Real-time Analysis

Model Transformation → Markov Chain Model → Reliability Availability Assessment
Behavior Specification

• Behavior specification of component types are generated automatically as Extended timed-automata in a one to one relation.

• It is based on analyzing the component contract specification.
UPPAAL Modeling Language

- **Time Automata** \((L, l_0, K, A, E, I)\)

  - \(L\) is a set of locations denoting states;
  - \(l_0\) is the initial location;
  - \(K\) is a set of clocks;
  - \(A\) is a set of actions, events causing transitions;
  - \(E\) is a set of edges, transition specifications; and
  - \(I\) is a function assigning clock constraints to locations as invariants.

Extending UPPAAL

• User identity parameter
• In the global declaration add security functions:
  – List of roles, privileges, groups, and representative users.
  – Assignment of services-privileges-roles
  – Assignment of data parameters-privileges-roles
  – Assignment of users-roles
  – Event access security matrix
  – Data access security matrix
Transformation Rules

Component Type

Component definition

Services

Data Parameters

Interface Types, Architecture Types, Connector Types

Contract Definition

Data Constraints

Data Security

Service Security

Reactivity

Time Constraints

UPPAAL Template

Create a location for every request for service

Locations (L)

Create an action for every request for service or request from service

Actions (A)

Create an edge for every request for service or request from service

Set values of parameters in the Update expression

Edges (E)

Expressions:
1- Select
2-Guard
3- Sync
4- Update

Used in Guard conditions, preconditions

Used to constrain Updates to data parameters

Used in Guard conditions, preconditions

Create an edge for every response from the service

Create a clock for every time constraint

Invariants (I)

Clocks (K)

Create an invariant for every time constraint

SAVCBS @ Dubrovnik, Croatia, 2007
Safety Property

• Stated positively stating that some thing good is invariantly true.

Let $\varphi$ be a formula,

$\Box \varphi$ : means that $\varphi$ should be always true.

$\Diamond \varphi$ : means that $\varphi$ will be eventually satisfied.
Security Properties

• Event Security:

\[ A \forall i \in [1, \text{NoOfUsers}] \quad C.user == i \land C.Service \implies \text{User-Has-Privilege}(C.user, C.Service) \]

• Data Security:

\[ A \forall i \in [1, \text{NoOfUsers}] \quad C.user == i \land \text{DataParameter} != \text{Null} \implies \text{User-Data-Access}(C.user, \text{DataParameter}) == \text{"read"} \]
Model Checking
This is just the beginning...

• Define trustworthiness requirements as first class architectural element.
• How to adopt/adapt the approach to other component models.
• Define a formal development process model.
• What type of tools are necessary to make the approach realizable?

http://users.encyclopedia.concordia.ca/~ms_moham/
Discussion