Towards a CBSE Formal Approach for Developing Trustworthy Systems

Mubarak Mohammad

http://users.icens.concordia.ca/~ms_moham/
ms_moham@cse.concordia.ca
Agenda

• Outline activities of the development process.
• Blue print of a framework
• Critics
• Discussions
DEVELOPMENT PROCESS
Component Engineering

• The activities involved in the component development process are:
  – Component Development
  – Component Assessment
  – Component Reuse

• Domain engineering activities are done before component development.
Domain Engineering

- Define requirements for all domain applications including trustworthiness requirements.
- Identify the applications and define their boundaries, relations, and trustworthiness requirements.
- Define domain applications and their relations and trustworthiness.
- Define application's components and their relations and trustworthiness requirements.
- Detail definition of domain components.
- Store domain engineering artifacts in repository.

Component Development

- Elicitate component requirements: functional, structural, and trustworthiness.
- Reuse existing documents and domain components information.
- Define requirements for reuse.
- Formal definition of components using TADL. Reuse domain formal definitions if exists.
- Validate formal definitions using TADL correction rules.
- Verify safety, security, timeliness, reactivity, and constraints using Model Checking.

Component Reuse

- Use requirements to search for components.
- Searching
- Selection
- Adaptation

Component Assessment

- Use design and implementation information to assess components implementation.
- Traceability Analysis
- Automated black-box Testing
- Certification

Repository

- Reuse components requirements, design, and implementation.
- Store components requirements, design, implementation, and assessment information.
Component-based System Development

• The activities in component engineering and component-based system development run in parallel.
System Development

--- Elicit system requirements: functional, structural, and trustworthiness.
--- Elicit environmental requirements.
--- Elicit deployment requirements.
--- Reuse existing domain knowledge from repository.

--- Selecting components using component reuse.
--- Formal definition of the system using TADL.
--- Reuse application architecture from domain if exists.

Formal model of software unit by composing components specification from the repository
Formal model of trustworthiness
Formal model of environment
Formal model of deployment

Validate formal definitions using TADL correction rules.
Verify safety, security, timeliness, reactivity, and constraints of the composite system using Model Checking.
Integration of components' implementations retrieved from the repository and writing glue code.

Testing

Deployment
Run-time monitoring and maintenance

redesign
reintegrate
DEVELOPMENT FRAMEWORK
The diagram illustrates the flow of tools across the design-time, implementation, and run-time phases. At the design-time phase, tools such as ADL, Behavioral Model, Real-Time Model, Architectural Analysis Tool, Model Checking Tool, Real-Time Analysis Tool, Transformation Analysis Tool, Compiler Tool, and Visual Modeling Tool are interconnected.

During the implementation phase, the Code Generation Tool and Repository are central, with links to the Integrated Compiler and Certification Authority. The repository provides code for reuse across different phases.

In the run-time phase, Run-Time Analysis Tool and Run-Time Environment Tool are connected, indicating the feedback and analysis cycle that supports the overall system. The diagram highlights the iterative process of development, testing, and deployment, showcasing the interaction and dependencies between various tools and components.
Design-time Tools
Visual Modeling

• Provide user friendly interface to design components and systems.
• Specify functional, structural, and non-functional requirements.
• Projects textual (TADL) and visual representations of the design.
• Project the model into 3 views:
  – CBD,
  – Real-time,
  – Trustworthiness
• The specification are transformed into an XML file according to TADL.
Compiler

- This tool checks the syntactic correctness of the visual modeling design with respect to its abstract definitions.
- If the design is syntactically correct, the compiler generates a formal descriptions of the visual model into different formats:
  - TADL
  - Behavioral model using UPPAAL extended timed automata
  - Real-time model using time automata extended with tasks
Transformation Analysis

- The correctness and completeness of the transformation process must be subject to reasoning.
- Design and implementation flaws are still possible during the development of the automatic transformation tools.
Simulation and Model Checking

• We have defined formal transformation rules that transform component types to UPPAAL extended timed automata.
• It is possible to integrate different model checkers into the framework to perform formal verification.
Real-time Analysis

• We have defined formal transformation rules that transform component types to *timed automata extended with tasks*.  
• We use *Times tool* to perform schedulability analysis
Architectural Analysis

• This tool analyzes the correctness of the architectural style and system configuration specification relative to architectural constraints defined in the system design.
Implementation Tools
Component Repository

• This is a storage place to store and reuse developed trustworthy components.

• The repository provides storage facilities for:
  – component specification (structure and contract),
  – development source code,
  – compiled, execution ready assembly of the component, and
  – usage profiles and certificates.
Code Generation

• This tool produces source code.
• It supports different programming languages such as C++, C#, and java.
• It analyzes the system design specification. Then, for every component or connector, if the component exists in the component repository then it should reuse it; otherwise, it should produce source code or skeleton for new components.
Traceability

• These tools analyze newly developed components and verify their conformance to design specifications.
• The traceability tool verifies that the code satisfies CBD, real-time, and trustworthiness specifications.
• Traceability analysis is not a trivial task. It requires scrutinizing the generated and developed components code.
Traceability of CBD specifications

• Maintain the relation between each CBD structural design element and its implementation construct.

• Information can be added to component’s meta data to link the implementation to its source design time specification.

• Then, Reflection techniques are used to read attributes and analyze component’s meta data.
Traceability of real-time specification

• *Worst case execution time* (WCET) of services can be specified as an attribute to a service at design time and as a custom attribute at implementation time.

• Then, during traceability analysis, the functions that implement real-time services can be executed to check if their measured execution time is bound by their WCET.
Traceability of trustworthiness

• The actual traceability of security and safety behavior can be analyzed using run-time analysis techniques.
Certification

- After traceability analysis, the tool interacts with a certification authority to obtain a certificate that indicates the trustworthiness of the component and the level of development conformity to design and quality attributes stated in its specifications.

- The certificate is based on design-time and traceability analysis.
Run-time Tools
Run-time environment

• The tool is a *middleware* between the component repository and the run-time environment that communicates with the operating system (e.g., J2EE or .NET run-time environment).

• It communicates with the component repository to load component assemblies.

• The tool allows a controlled reconfiguration to the running system (e.g., adding a new component or replacing an existing one).
Run-Time Analysis

• This tool performs run time analysis during system execution.
• The tool ensures that system behavior conforms to the stated functional and nonfunctional properties.
• This is done by observing input, output, and system states during program execution.
• Execution sequences can be monitored, logged, and visualized to ease analyzing system behavior.
• These sequences are used to build usage profiles for components.
• These profiles can be used to monitor the availability and analyze the reliability of components and system.
• The execution profiles can be subjected to formal verification.
• Verification is done by ensuring that system execution doesn’t reach a state that violates trustworthiness.
Critics

• “..nice to add to my shopping list once it is implemented..”
• “..It is impossible to implement it..”
Discussion