A Component-based Approach for Constructing High-confidence Distributed Embedded Systems

Barrett Bryant¹, Rajeev Raje², Mikhail Auguston³, Jeff Gray¹, Shih-Hsi Liu¹, Mihran Tuceryan² and Andrew Olson²

1. University of Alabama at Birmingham
2. Indiana University-Purdue University Indianapolis
3. Naval Postgraduate School
Realizing Distributed Embedded Systems Using Service-Oriented Architectures

- DES as a composition of heterogeneous, independently developed components
- Each component offers services along with associated assurances about them.
- Confidence characteristics incorporated during design, construction, deployment, and composition of these services
- Cost of verification and validation reduced
Research Goals

- Develop service-oriented models for DES which incorporate high-confidence characteristics such as correctness and QoS
- Develop, discover and select components using service-oriented models, so that components and their ensemble exhibit high confidence
- Automate the composition of components to minimize vulnerability arising from handcrafting
- Validate the assembled DES with respect to both functional correctness and QoS
Key Research Issues

- Architecture-based Interoperability
  - Automation, standardization, mappings and tools

- Distributed Resource Discovery
  - Specification, publication, distribution, selection

- Validation of Quality Requirements
  - Vocabulary and associated metrics, composition, monitoring

Main Challenge: Heterogeneity
UniFrame Knowledge Base

- Developed by domain experts for specific application domains
- Describes service-oriented architecture for the application
- Specifies functional and QoS properties of components that make up the architecture
- Discovers and matches components to the requirements
- Automatically generates code for interoperation of components
- Predicts and empirically measures vulnerability properties of the integrated system
Formal Methods

- Language for describing rules for integrating components – Two-Level Grammar (TLG)
- Automated scenario generation from environment models – Attributed Event Grammar (AEG)
Two-Level Grammar

- TLG consists of two context-free grammars corresponding to the set of type domains and the set of logical rules operating on those domains.
- The first level of the grammar, called *meta-rules*, defines the structure of the domain, including the syntactic interfaces of components.
- The second level of the grammar, called *hyper-rules*, defines the rules for composing components, performing static evaluation of QoS constraints, and generation of connector code.
**TLG Example**

ClientUMM, ServerUMM :: UniframeMetaModel.
ClientOperations, ServerOperations :: \{Interface\}*. 

```plaintext
generate Application system
   from ClientUMM and ServerUMM with QoS :
ClientOperations := ClientUMM get operations,
ServerOperations := ServerUMM get operations,
OperationMapping := map ClientOperations into ServerOperations using Application domain,
ComponentModel :=
   ServerUMM get component model,
generate java code for OperationMapping using ComponentModel with QoS.
```
TLG Glue/Wrapper Generation

Java RMI Client

CORBA Server

TLG specification for Java RMI Client

Connector

Proxy client

Knowledge base
Attributed Event Grammar

-Attributed event grammar (AEG) provides a uniform approach for automatically generating, executing, and analyzing tests.

-Quantitative and qualitative risk assessment can be performed based on statistics gathered during automatic test execution.

-AEG provides automated testing of distributed real-time embedded software systems, based on modeling the environment in which a system will operate.
AEG Example

- Shoot ::= Fire
  \[
  ( p(0.3) \text{Hit } /\text{Send\_input\_to\_SUT} (\text{Hit} . \text{time})/ \mid p(0.7) \text{Miss} )
  \]

- Large number of Shoot scenarios can be generated.

- Each event trace will satisfy the constraints imposed by the event grammar.
AEG Validation

Environment model represented as an event grammar

Generator

Run time monitor

How to create test cases

Test driver (in C or assembly language)

How to run test case

How to monitor the results

SUT
Case Study – Mobile Augmented Reality

GPS

Trackers

Soldier

HMD

Interaction

Wireless Device

Computation

Environment Model

Environment Data

Store

Battlefield

Assign Strategies

DoD

Rifle

Hand Tracker

Department of Defense
Conclusions

- Development and reuse of existing software components for embedded systems in a manner that fosters high-confidence
- Partially automates the software design and validation process for embedded systems, thereby increasing reliability
- Assists in the development of standards for software component descriptions in embedded domains
Future Work

- Expand case studies to include other domains
- Develop prototype tool suites to further validate framework
Further Information

http://www.cs.iupui.edu/uniFrame