Component-Based Development

Jim Browne

Contents

1. Overview
2. What is a component?
3. What is an interaction?
4. Component-based Compositional Development
5. Component-based verification
6. Conclusion
7. Postscript on Internet and Grid Computing
Overview

This report is on research done by Kevin Kane, Nasim Mahmood, Fei Xie and some of their predecessors.
Overview

Hardware development is component-based

    Design as units and interactions

    Implementation (FPGAs) as cells/macrocells

Software development is migrating to component-oriented development

    Aspect-oriented, Feature-oriented, Product-line architectures, Common-component architecture

Software verification requires component-based development
What Is a Component?

A component is:

• One or more logical functions of an algorithm or computation implemented as one or more sequential computations,

• An interface which specifies the information used for selection and matching of components and

• A state machine which manages the interface, the interactions with other components and the invocation of the component functions.
What is a component? - Continued

1. A component is NOT a class or object.
   Inheritance, polymorphism, etc. are not defined

2. The purpose of components is to be composed, not to be extended.

3. All relationships are established through interfaces.
What is an interface?

Interface – A specification of the properties, behaviors and requirements of the component.

Signatures are the simplest specifications of behaviors

Components may have internal state -> signatures are inadequate -> state machines

Properties – What, how, etc.

Requirements – Specification of the components which this component must have in order to implement its functionality.
Examples of Component-based Systems and Development Processes

Web Services – XML Interfaces encapsulating functionality

Grid Services – Virtual Organization as composition of organizations.

Peer to peer systems – Each peer has the structure of a “component.” Must be self-contained and interact only in conformance to protocol.
Interfaces for Composition

Accepts – Properties and Behaviors

Profile

{Transactions}

Protocol

Requires – Implementation Requirements

{Selector

Transactions

Protocol}
Accepts Interface for Matrix_Partition Component

profile:

string domain = "matrix";
string function = "distribute";
string element_type = "complex";
bool distribute_by_row = true;

transaction:

int get_matrix(in mat2 grid_re,in mat2 grid_im, in int n, in int m, in int p);

protocol: dataflow;
Requires Interface for Matrix_Partition - Continued

{selector:
    string domain == "fft";
    string input == "matrix";
    string element_type == "complex";
    string algorithm == "Cooley-Tukey";
    bool apply_per_row == true;
    transaction:
        int get_grid_n_m(out mat2 out_grid_re[],out mat2 out_grid_im[], out int n/p, out int m);
    protocol: dataflow;
} index [ p ]
Matching and Composition

Match Selector Clause with Accepts Clause

a. Can be done at compile time, link time or runtime.

b. Compile time match is not much more than sophisticated type matching

c. Link time match is no more than qualified “make.”

d. Run time match is similar to Unix dynamic binding or peer to peer file search.
Output of Compositional Compiler

Compiled to a parallel program implemented in MPI or Posix Threads
Verification by Model Checking

Model Checking – Verification that a temporal property of a system holds for ALL execution paths.

Temporal Property Example – The value of a given variable never exceeds a specified value.

Problem – State Space Explosion

Solution – Component-based development and model checking by compositional reasoning.
Model Checking of a Component-Based System

1. Establish the temporal properties of “primitive” components.
2. Compose primitive components.
3. Establish Properties for “composed” components.
4. Continue composition until system is attained.
Model Checking of a Component-Based System

Establish the temporal properties of a “primitive” component.

a. Specify an execution environment for a component
b. Specify the properties to be checked
c. Model check the properties on the component.
d. Add the verified properties to the interface of the component.
Establish Properties for a “composed” component

a. Compose the components to create a new component by matching the interfaces of the composing components.
Establish Properties for a “composed” component

a. Compose the components to create a new component by matching the interfaces of the composing components.

b. Specify an execution environment for the component

(The execution environments for each component separately minus the requirements satisfied by the matching of composing component interfaces.)
Establish Properties for a “composed” component

a. Compose the components to create a new component by matching the interfaces of the composing components.

b. Specify an execution environment for the component

(The execution environments for each component separately minus the requirements satisfied by the matching of composing component interfaces.)

c. Specify the properties to be checked
Establish Properties for a “composed” component

a. Compose the components to create a new component by matching the interfaces of the composing components.

b. Specify an execution environment for the component
   (The execution environments for each component separately minus the requirements satisfied by the matching of composing component interfaces.)

c. Specify the properties to be checked

d. Model check the properties on the composed component using the verified properties to represent the individual behaviors of the composing components.
Establish Properties for a “composed” component

a. Compose the components to create a new component by matching the interfaces of the composing components.

b. Specify an execution environment for the component
   (The execution environments for each component separately minus the requirements satisfied by the matching of composing component interfaces.)

c. Specify the properties to be checked

d. Model check the properties on the composed component using the verified properties to represent the individual behaviors of the composing components.

e. Add the verified properties to the interface of the composed component.
Establish Properties for a “Composed” Component

a. Compose the components to create a new component by matching the interfaces of the composing components.

b. Specify an execution environment for the component
   (The execution environments for each component separately minus the requirements satisfied by the matching of composing component interfaces.)

c. Specify the properties to be checked

d. Model check the properties on the composed component using the verified properties to represent the individual behaviors of the composing components.

e. Add the verified properties to the interface of the composed component.
Result

1. Properties of total systems derived from properties of components.

2. State space explosion avoided

3. Possible breakthrough for reliability and security of software systems.
Conclusions

Component-based development is the future of software development.

But like all futures, who knows when!!
Postscript on Internet and Grid Computing

Integration of SETI@home and peer to peer interactions.
Enable intranet and Internet scale execution of general parallel computations.
Example – Peer to peer computation of “Google” pageranks Create “Gaia!!”