Techniques for Embedding Executable Specifications in Software Component Interfaces

Ross McKegney
Queen's University & IBM Canada
Toronto, Ontario Canada
mckegney@cs.queensu.ca

Dr. Terry Shepard
Royal Military College of Canada
Kingston, Ontario Canada
shepard@rmc.ca

Concepts

- A **component** is an executable unit of deployment and composition
- **Semantic integrity** means that components are used in the way in which they were intended (clients do not violate the assumptions made by the component developer)
- **Contracts** are formal specifications interleaved with source code

Goal

- To investigate techniques and tools that can be used to improve *semantic integrity* in component-based systems
  - For now, the scope is limited to techniques for specifying functional properties using contracts on component interfaces

3

Interface Contracts

- Component interfaces can be thought of as the mediating agent between interacting components, essentially providing a contract between them

- Current practice is to use syntax-only contracts, as illustrated by the following OMG IDL interface

4

```
#include "Item.idl"

module ShoppingCart {

  const int MAX_QTY = 100;

  struct CartEntry {
    Item item;
    short qty;
  };
  typedef sequence <CartEntry> CartEntrySeq;

  exception InvalidQtyException { string ex_info; };
  exception CartFullException { string ex_info; };
  exception CartEmptyException { string ex_info; };
  exception ItemNotInCartException { string ex_info; };

  interface IShoppingCart {

    //Add an item to the shopping cart.
    void addItem(in Item item, in short qty) raises (CartFullException,
      InvalidQtyException);

    //Remove an item from the shopping cart.
    void removeItem(in Item item, in short qty) raises (ItemNotInCartException,
      InvalidQtyException);

    //Empty the shopping cart.
    void emptyCart();

    //Retrieve the contents of the cart.
    void getItems(out CartEntrySeq entries) raises (CartEmptyException);
  };
};
```

What does this mean?

ShoppingCart Assumptions

- Type and range of parameters/return values
- Semantics of the exceptions that methods can throw
- Types and versions of the other components in the system, including other application components, as well as platform level components
- Protocol of interaction
- Effects of operations on the internal state of the component
- Performance assumptions
- Concurrency assumptions
- Valid customizations/parameterizations
- Control assumptions
- …

These assumptions all affect the correct behaviour of our component, and of any components that use its services.

Design by Contract

Design by Contract

- Design by Contract™ was developed by Bertrand Meyer in the early 1990s, and implemented in the Eiffel programming language
  - Interleaves formal specification (pre-/post-conditions and invariants) with source code
  - Machine executable specifications, allows for less ambiguous interface definitions and when used correctly can aid implementation and testing

DbC Implementations

- DbC has been implemented for a variety of programming languages, including Java, C++, Ada, Smalltalk, Python, and Lisp
- iContract (Java)
  - Source code pre-processor, accepts contracts written as comments and generates the runtime checker code
  - Constraints specified using an executable subset of OCL
- Biscotti (Java RMI)
  - Similar syntactically to iContract and other Java DbC tools, but implemented as extensions to the Java language rather than as a pre-processor
  - Designed specifically for RMI, and supports runtime pre/post-condition and invariant evaluation, and assertion reflection

Behavioural Interface Specification Languages

BISLs

- Behavioural Interface Specification Languages (BISLs) were proposed by Jeannette Wing in 1987
  - Two-tiered approach: the lower tier specifies the mathematical notation to be used in specifications and a domain model, while the upper tier specifies interfaces to components in terms of pre-/post-conditions and invariants, using the language of development
  - E.g. Larch, VDM, JML, (AsmL)

11

IDL Extensions

- Larch/Corba (Sivaprasad, 1995)
  - Two-tiered approach: lower tier is the Larch Shared Language (LSL), upper tier is an extension of OMG IDL
  - Supports pre/post-conditions, invariants, and a simple synchronization primitive (requires, when, modifies, ensures)
- IDL++ (D'Souza & Wills, 1995)
  - Response to an RFI by the OMG for adding semantics to IDL
  - Four tiers: Traits and Theories, Fundamental Object Model, Methodology Constructs, and End-User Models

12

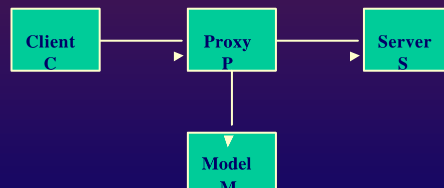
JML [Leavens 1999]

- "More expressive than Eiffel, easier to use than Larch or VDM"
- Uses a slight extension of the Java syntax to allow developers to embed specifications in Java classes and interfaces
- Available tools allow static analysis, verification, automated document generation, and run-time debugging

13

AsmL and COM/.NET [MSR]

- Applications of AsmL within Microsoft include modeling, rapid prototyping, analyzing and checking APIs, devices and protocols
- Allows for heterogeneous systems, with some components implemented and others executable specifications



14

Conclusions & Opportunities

Contracts for Components?

- Design by Contract was developed for objects, and it is worth commenting on whether the technique can scale to components
  - Heterogeneity of components and infrastructures makes contract runtime evaluation more difficult
  - Systems may be composed dynamically, meaning that we may wish to leave contracts turned on even in a deployed system
  - Issues of component granularity; it may be too difficult to model the abstract state as the size/complexity of components increase
  - Components can exhibit complex interaction mechanisms (e.g. event broadcasting or callbacks) which should be explicitly supported
  - Components will be reused based not only on their specification, but also on their implementation
  - Non-functional attributes (e.g. performance, security) are also important aspects of the contracts for components

Opportunities

- Extending contract notations to support the additional types of constraint that are important at the level of components (e.g. quality of service, different interaction mechanisms, protocol specification, etc.)
- Transitioning between contracts and other formal specification techniques
- Integrating contract evaluation tools with standard middleware architectures
- Developing tools that can automatically analyze a component and generate and/or confirm its contracts
- Reuse by contract
- Contract negotiation
- Static analysis of composed systems, using contract information
- Investigating scalability/performance attributes of existing DbC/BISL techniques
- …

17

for more information…

Contact:

Ross McKegney: mckegney@cs.queensu.ca

Dr. Terry Shepard: shepard@rmc.ca

Eiffel:	http://www.eiffel.com
iContract:	http://www.reliable-systems.com/tools/iContract/iContract.htm
JML:	http://www.cs.iastate.edu/~leavens/JML.html
ASML:	http://research.microsoft.com/foundations

18