XCD

Design-by-Contract for Reusable Components & Realizable Architectures

Mert Ozkaya & Christos Kloukinas

http://staff.city.ac.uk/c.kloukinas/Xcd
Two main approaches:

- Components & Connections
  - Darwin 1996

- Components & Connectors
  - Wright 1997

http://staff.city.ac.uk/c.kloukinas/Xcd
Current State

- Malavolta et al., IEEE TSE v. 39, n. 6 “What Industry Needs from Architectural Languages: A Survey” (“needs” or “asks for”?)
- No “steep learning curve” (UML), performance, reliability, . . .

http://staff.city.ac.uk/c.kloukinas/Xcd
Issues

- Formal process algebras not practitioners’ cup of tea...
  - Practitioners care about Performance/Reliability/etc.

  But Performance/Reliability @ Deadlocked States = ?

- Components + Connections
  - No Connectors
  - Components must describe the protocols they’ll be used with

    Modularity, Reusability, Architectural Exploration?

    Architectural Mismatch?

http://staff.city.ac.uk/c.kloukinas/Xcd
“All systems can be specified without connectors, therefore connectors are not needed”

Alice (Monday):  $\sum \sqrt{x}$

Bob (Friday):  $\prod x^2$

Yes, but what about map-reduce? (reduce R (map M x))

Why re-invent the wheel each time?
- We’ll not get it right each time

Define it once and Reuse-by-Calling it not Reuse-by-Copying it
Connectors

- Specify protocols once, reuse them by calling them
- Components become protocol-independent
- Components are more modular, easier to specify
  - Less work $\leadsto$ more specs $\leadsto$ architectural mismatch less likely
- Increase reusability of components & connectors (vector + bubble/quick/merge/...sort)
- Easier to verify each independently
- Easier to understand the system structure
- Easier to explore alternatives

http://staff.city.ac.uk/c.kloukinas/Xcd
Wright defined the base notions: Roles + Glue
Components implement Roles
System = Components \parallel Glue
All ADLs supporting connectors follow Wright but . . .
Glue adds global constraints
⇒ Unrealizable in general
  ◆ If we want to preserve “communication integrity”
  ◆ And we do – most analyses depend on it

http://staff.city.ac.uk/c.kloukinas/Xcd
Unrealizable Architecture – A Nuclear Plant

(a) A decentralized architecture

(b) The plant’s (unrealizable) MSCs

(c) An unavoidable bad behaviour


http://staff.city.ac.uk/c.kloukinas/Xcd
Problem
Beginnings
State
Issues
No Connectors?
Connectors
Realizability
Unrealizable Architecture
Unrealizable Wright Connector
Req vs Arch
Realizable Plant
Xcd
Conclusions
Extras

connector Plant_Connector =
role P1 = ur → na → P1. // increase both
role P2 = ur → na → P2. // double both
role UR = inc → UR □ double → UR.
role NA = inc → NA □ double → NA.

 glue G = P1.ur→UR.inc → P1.na→NA.inc
  → P2.ur→UR.double → P2.na→NA.double
  → G
  □ P2.ur→UR.double → P2.na→NA.double
  → P1.ur→UR.inc → P1.na→NA.inc
  → G. // → link, → global constraint


Wright’s (unrealizable) connector for the nuclear plant

■ Property is verified!
■ But no way to realize it while preserving comm. integrity . . . :-(
■ Architect needs to be told requirement isn’t satisfied
⇒ Global constraints are requirements
■ Architecture shouldn’t simply repeat the requirements

http://staff.city.ac.uk/c.kloukinas/Xcd
Requirements vs Architecture

Requirement  “I want infinite stairs”

Architecture

- Architecture needs to be realizable
  *In a way that respects communication integrity*

- Otherwise, performance/reliability/etc analyses are invalid

M. C. Escher  
(1898 - 1972)

http://staff.city.ac.uk/c.kloukinas/Xcd
**Realizable Plant**

connector Realizable_Plant_Connector =
role P1 = ur → na → P1. // same
role P2 = ur → na → P2. // same
role UR = inc → UR □ double → UR. // same
role NA = inc → NA □ double → NA. // same

**glue** G = (G1 || G2 || G3 || G4). // Real glue
// where Gi’s are simple links:
G1 = P1.ur → UR.inc →G1.
G2 = P1.na → NA.inc → G2.
G3 = P2.ur → UR.double → G3.


GlueProperty = UR.inc → NA.inc
→ UR.double → NA.double
→ GlueProperty
□ UR.double → NA.double
→ UR.inc → NA.inc
→ GlueProperty.

- **Of course, GlueProperty is no longer satisfied...**
- At least now we *know* that it doesn’t work!
- And so do the designers/developers/clients ...
XCD: Main Ideas

- Support arbitrary connectors
  ⇒ Modular specifications
  Simpler component specifications
  ⇒ Reusable components & reusable connectors

- Only local constraints can be imposed
  ⇒ Realizable architectures + communication integrity

- Programming language-like syntax
  ⇒ Not (too) scary (?)

- Formal (extended Design-by-Contract approach – JML-inspired)
  ⇒ Can verify (with SPIN):
    1. Are provided services (local) interaction constraints satisfied?
    2. Are provided services functional pre-conditions complete?
    3. Are there race-conditions?
    4. Do event buffer sizes suffice? and
    5. Are there (global) deadlocks?

  Plus, general LTL properties (new)
Cleanliness is next to Godliness…

Machine A

```java
@interaction {
  waits: ! washing; }
void open_door();
```

A blocks my call till it’s safe

Machine B

```java
@interaction {
  accepts: ! washing; }
void open_door();
```

B may reject it with undefined behaviour (might electrocute me!)

http://staff.city.ac.uk/c.kloukinas/Xcd
Java Thread as an XCD Component

1. component Thread {
2.   bool started := false; // component data.
3.   bool died := false;
4.   aliveP() {return started && !died;} // helper function
5. }

6. provided p {
7.   @functional {ensures: \result := aliveP();}
8.   bool isAlive();
9. }
10. @interaction {waits: !aliveP();}
11. void join();
12. @interaction {accepts: ! started;}
13. @functional {ensures: started := true;}
14. void start();
15. // ... other methods
16. }
17. }

- Constraints are more modular (*JML allows but doesn’t enforce it*)
- Functional constraints must be complete! Call already accepted!!!

http://staff.city.ac.uk/c.kloukinas/Xcd
Software Components & DbC

- Software Components have:
  - Provided method ports ✔
  - Required method ports ❌
  - Consumer event ports ❌
  - Emitter event ports ❌

- Separation of interaction/functional contracts

  **Restaurant** Provide a service from 7pm-11pm – Italian menu
  
  **Client** Require a service from 9pm-12pm – pizza or Peking duck

- Extension for required methods & event consumption/emission

```java
provided p {
    @interaction { waits/ accepts : \(\phi_1\); }
    @functional {
        requires : \(\phi_2\);
        ensures : d := f(d,p);
    }
    void service(p); }
}

required p {
    @interaction { waits: \(\phi_1\); }
    @functional {
        promises : p := f1(d);
        requires : \(\phi_2\);
        ensures : d := f2(d);
    }
    void service(p); }
}
```

DbC created for objects
- No events
- No required methods
- Ignores clients’ needs
Some Choices

- Ensures are assignments (possibly non-det), not formulæ:
  \[ z := \frac{x}{y} \quad \text{vs} \quad x = y \times z \]
  \[ z \in [0, 255]: \quad \text{1 state} \quad \text{vs} \quad 256 \]
  \[ x \in [0, n]; y \in [0, n - x]; z \in [0, n - x - y]; \quad \text{vs} \quad 0 \leq x + y + z \leq n \]
  \[ (n + 1)(n^2 + 5n + 6)/6 \quad \text{states} \quad \text{vs} \quad (n + 1)^3 \]
  \[ n = 255: \quad 2.8 \text{ M states} \quad \text{vs} \quad 16.7 \text{ M} \]

- \( \text{req}_1/\text{ens}_1 \ \text{\textbf{\textbackslash XXX}} \ \text{req}_2/\text{ens}_2 \equiv \)
  \( \text{JML} – \ \text{\textbf{\textbackslash also}} \ (\text{req}_1 \rightarrow \text{ens}_1) \land (\text{req}_2 \rightarrow \text{ens}_2) \)
  \( \text{XCD} – \ \text{\textbf{\textbackslash otherwise}} \ (\text{req}_1 \rightarrow \text{ens}_1) \lor (\text{req}_2 \rightarrow \text{ens}_2)) \land (\lor_i \text{req}_i) \)
  - Why otherwise instead of also:
    - Avoid conflicting/circular assignments
    - Introduce frames for race-condition checks
  - \( (\lor_i \text{req}_i) \): action is accepted from interaction constraints
    \( \Rightarrow \) \text{req}_i \text{ must be complete} \n
- Roles cannot reject an action, only disable it (\textit{waits}, no \textit{accepts})
Basic Component: Type Param* Data* HelperFunction* Port*;
  Port: Provided | Required | Emitter | Consumer;
P/R/E/C: Action*;
Action: IntCon? FunCon? Signature;

Connector: HOParam+ Param* Role* ConInst*;
  Role: Type Param* Data* HelperFunction* Port*; // Like Component

Composite Component: Param* CompInst* ConInst*;

Higher-Order Composite Components

Connectors = +
Interaction Constraints

- XCD roles add extra data and port constraints to components
- Component port must have all role port actions
  Wright: Component ports *implement* role ports
  XCD: Component ports *interpret* role ports
- *Role’s not a wrapper – it’s a script to be interpreted*
  - Wrappers cannot disable active (required/emitter) actions
Translator to SPIN’s ProMeLa

CompA aInst;  CompB bInst;
ConnC cInst(aInst, bInst);

- Each basic comp. instance (aInst, bInst), a ProMeLa process
  do
    :: port_I_atomic_action_J_translation
    :: port_K_nonatomic_action_M_translation_partA
    :: port_K_nonatomic_action_M_translation_partB
  od

- Ports allow at most one action to be active at any time
- Each action is guarded by its own waits/accepts guard, and by all the waits guards of its roles
- Event emission/consumption actions are atomic
- Required methods are two atomic blocks (send request, receive response)
- Provided methods are either atomic (req/comp/res) or two atomic blocks (req/comp, comp/res)
  - Non-atomic provided method used for call chaining
- Non-atomic methods encoded so as to catch race-conditions

http://staff.city.ac.uk/c.kloukinas/Xcd
Component ProMeLa Structure

::atomic { // a provided port method
    port.Channel_req ? method.Args
        : roleWaits && method.Waits && method.Accepts ->
        assert(FunctRequires); // Ensure functional completeness
        calcData(...);
        port.Channel_res ! method.Args;
    }

::atomic { // same provided port method
    port.Channel_req ? method.Args
        : roleWaits && ! method.Accepts ->
        assert(False); // Request rejected - CHAOS
    }

...::atomic {
    // sending a request - a required port
    selectParams(method.Args,
        roleWaits && methodWaits && ! port.Lock, ...) ->
    port.Lock = method;
    port.Channel_req ! method.Args;
    }

::atomic { // receiving a response - same required port
    port.Channel_res ? method.Args
        : port.Lock == method ->
        calcData(...);
        port.Lock = 0;
    }

http://staff.city.ac.uk/c.kloukinas/Xcd
Centralized Plant

```java
enum ordr := {none, incFirst, dblFirst};

role roleController {
  ordr order := none;
  bool p1_incNARcvd := false;
  bool p1_incURRcvd := false;
  bool ur_incUREmtd := false;
  bool na_incNAEmtd := false;
  all_received(){return
    p1_incURRcvd && p1_incNARcvd && p2_dblURRcvd && p2_dblNARcvd;}
}

provided port_variable P1toUR {
  @interaction{
    waits: !p1_incNARcvd;
    ensures: p1_incNARcvd := true; }
  void incUR(); }

required port_variable CtoURinc {
  @interaction{
    waits: all_received() && !ur_incUREmtd && (order==incFirst ||
      order==dblFirst && dbl_emitted());
    ensures: ur_incUREmtd := true; }
  void incUR(); }

required port_variable CtoNAinc {
  @interaction{
    waits: ur_incUREmtd && !na_incNAEmtd;
    ensures: // clear flags if dblFirst
      p1_incURRcvd := !(pre(order) == dblFirst);
      ...
      ur_dblUREmtd := pre(order) == dblFirst ? true : pre(ur_dblUREmtd);
      ...}
    void incUR(); }
```

http://staff.city.ac.uk/c.kloukinas/Xcd
<table>
<thead>
<tr>
<th>Case Study</th>
<th>Issues</th>
<th>State-vector (Bytes)</th>
<th>States</th>
<th>Memory (MB)</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decentralized Nuclear Plant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centralized Nuclear Plant</td>
<td>glue</td>
<td>240</td>
<td>137</td>
<td>73</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td></td>
<td>424</td>
<td>168349</td>
<td>407776</td>
<td>186</td>
</tr>
<tr>
<td>Lunar Lander v. 1</td>
<td>Ovrlw</td>
<td>372</td>
<td>118</td>
<td>78</td>
<td>131</td>
</tr>
<tr>
<td>Lunar Lander v. 2</td>
<td></td>
<td>392</td>
<td>4223125</td>
<td>8072166</td>
<td>3793</td>
</tr>
<tr>
<td>Gas Station (1 customer)</td>
<td></td>
<td>188</td>
<td>1003</td>
<td>1401</td>
<td>130</td>
</tr>
<tr>
<td>Gas Station (2 customers)</td>
<td></td>
<td>288</td>
<td>1136214</td>
<td>2793961</td>
<td>382</td>
</tr>
<tr>
<td>Gas Station (3 customers)</td>
<td></td>
<td>368</td>
<td>25056808</td>
<td>89254880</td>
<td>7024</td>
</tr>
<tr>
<td>BITSTATE (3 customers)</td>
<td></td>
<td>368</td>
<td>62792292</td>
<td>207452380</td>
<td>24</td>
</tr>
<tr>
<td>BITSTATE (4 customers)</td>
<td></td>
<td>456</td>
<td>66989014</td>
<td>289982810</td>
<td>25</td>
</tr>
<tr>
<td>BITSTATE (5 customers)</td>
<td></td>
<td>544</td>
<td>69607515</td>
<td>356984080</td>
<td>26</td>
</tr>
<tr>
<td>Aegis v. 1</td>
<td>L-DDLCK</td>
<td>620</td>
<td>13834057</td>
<td>71301546</td>
<td>7024†</td>
</tr>
<tr>
<td>BITSTATE v. 1</td>
<td>L-DDLCK</td>
<td>620</td>
<td>64408848</td>
<td>266469200</td>
<td>37</td>
</tr>
<tr>
<td>BITSTATE v. 2</td>
<td></td>
<td>548</td>
<td>63568962</td>
<td>268078040</td>
<td>35</td>
</tr>
<tr>
<td>English auction v. 1 (1 part.)</td>
<td>DDLCK, Ovrlw</td>
<td>140</td>
<td>296</td>
<td>295</td>
<td>130</td>
</tr>
<tr>
<td>English auction v. 2 (1 part.)</td>
<td>Ovrlw</td>
<td>144</td>
<td>776</td>
<td>1642</td>
<td>130</td>
</tr>
<tr>
<td>English auction v. 2 (2 part.)</td>
<td>Ovrlw</td>
<td>232</td>
<td>1293488</td>
<td>3732650</td>
<td>367</td>
</tr>
<tr>
<td>English auction v. 2 (3 part.)</td>
<td>Ovrlw</td>
<td>312</td>
<td>27315867</td>
<td>96797687</td>
<td>7024†</td>
</tr>
<tr>
<td>BITSTATE v. 2 (3 part.)</td>
<td>Ovrlw</td>
<td>312</td>
<td>57105380</td>
<td>189090640</td>
<td>20</td>
</tr>
</tbody>
</table>

“States Stored” unique global states.
“States Matched” states revisited during the search.
† run out of memory (7024 MB).

All case studies available at the XCD web site
Conclusions

Xcd – a new ADL

- Support for connectors
  - Arbitrary, complex connectors
  - Always realizable

- Formal
  - Architectures can be model-checked
  - Reasonable results for a number of case studies

- A less steep learning curve (?)
  - Closer to a programming language than process algebras
  - Extended DbC

http://staff.city.ac.uk/c.kloukinas/Xcd
Future Work

- Model optimization (generalize SPIN’s conditional message reception)

- Tool
  - Support recursive connectors
  - Support comp/port arrays in connectors

- Verification of components (OK) & connectors (??) in isolation
  - Construct testing environments to complete a sub-system

- Compute interface of a composite component
  - Spin + learning? Not so good – complex interfaces...
  - SMT solvers?

- Property language
  - LTL not so easy, even with KSU’s SAnToS lab’s LTL patterns
  - Observer processes can be used

- Extend
  - Timed, Probabilistic, Hybrid automata

Thank You!

http://staff.city.ac.uk/c.kloukinas/Xcd
Extras – Verifying Arbitrary Connectors in Isolation

http://staff.city.ac.uk/c.kloukinas/Xcd
Verifying Connectors – A generic C++ Algo

```cpp
#include <iostream>
using namespace std;

template <typename T>
void myswap(T & x, T & y) { T tmp = x; x = y; y = tmp; }

int main() {
    int i = 3, j = 5;
    cout << i << ' ' << j << endl; // 3 5
    myswap(i, j);
    char c; cin >> c;
    cout << i << ' ' << j << endl; // 5 3 myswap swaps!

    return 0;
}
```

http://staff.city.ac.uk/c.kloukinas/Xcd - p. 25
Algo: A set of ordering constraints on functions

```cpp
template <typename T>
void myswap(T & x, T & y) { T tmp = x; x = y; y = tmp; }
```

T { provides: Value copy(); }  
{ requires: Value copy(); }

tmp: request a copy from (x) -> accept copy request from (y);
	x: accept copy request from (tmp) -> request a copy from (y);

y: accept copy request from (x) -> request a copy from (tmp);
#include <iostream>
#include <string>
using namespace std;

template <typename T>
void myswap(T & x, T & y) { T tmp = x; x = y; y = tmp; }

class person {
    string name; int age;
public:
    person(string n, int a) : name(n), age(a) {}
    ostream &print(ostream &o) const;
};

ostream &operator<<(ostream &o, const person &p) {
    return p.print(o);
}

int main() {
    person p1("alice", 30), p2("bob", 19);
    cout << p1 << endl << p2 << endl;
    myswap(p1, p2);
    char c; cin >> c;
    cout << p1 << endl << p2 << endl;

    return 0;
}
ostream &person::print(ostream &o) const {
    return o << name << " is " << age << " years old."
};

person & person::operator=(const person &o) {
    name = o.name + " (not really)";
    age = age + o.age/2;
    return (*this);
}
template <typename T>
void myswap(T & x, T & y) { T tmp = x; x = y; y = tmp; }

T { provides: Value copy(); }
{ requires: Value copy(); }

tmp: request a copy from (x) -> accept copy request from (y);

x: accept copy request from (tmp) -> request a copy from (y);

y: accept copy request from (x) -> request a copy from (tmp);

■ Algo: A set of ordering constraints on functions

■ No functional guarantees
  ◆ Function implementations?
  ◆ Other unconstrained functions not in T?
  ◆ Multi-threading?

■ So which property should myswap satisfy?