Programming Models for Dependable Network Software Development

Agusa Laboratory, Department of Information Engineering

Our research group has been working on the topics about software development. We have been focusing on the issues about network software development, where the component-based architecture is getting more popularity. Aiming at gaining more reliability, we present the following two topics of our research.

Communicating Processes for Functional Programming Language

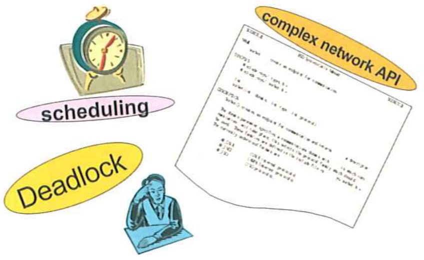
The Real World is Concurrent.

- Graphical User Interface
- Network Service
- Multiprocess Operating System

…However, concurrent programming is difficult.

- Concurrency Problem
  - Deadlock, Race Condition, Scheduling, etc…
- Based on informal model

Difficult to assure the reliability of program components



Especially, conventional Network Programming

- requires detailed understanding of internal effects along with parameters

Complex and less abstract API usually leads to error, also hard to analyze.

Formally Defined & Consistent concurrency support is required.

Our Approach:

π-calculus:

Formal Model of Concurrent System

Process Algebra for "mobility":

$\overline{x}v$ …send v via channel x

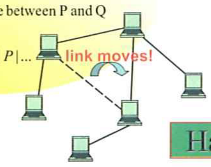
$x(y)$ …receive a name via channel x, binding it to the name y.

$P \mid Q$ …concurrent execution of P and Q

$P + Q$ …nondeterministic choice between P and Q

νzP …restrict usage of z in P

$!P$ …infinite composition, $P \mid P \mid P \mid \ldots$



Haskell:

Purely Functional Programming Language

```
sieve n = reverse (sievelist [] [2..n])

sievelist ps [] = ps
sievelist ps (n:ns) | n `candiv` ps = sievelist ps ns
                    | otherwise     = sievelist (n:ps) ns
 where
  candiv n [] = False
  candiv n (k:ks) | n `rem` k == 0 = True
                  | otherwise      = candiv n ks
```

ex. Eratosthenes' sieve

I/O system exists, but not based on concurrency theory

•x.fx

powerful, industrial strength purely functional language

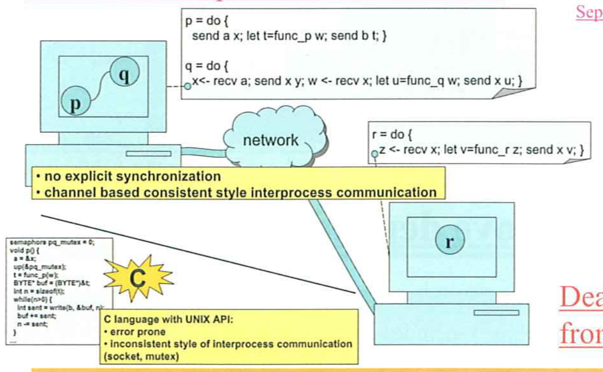
$Y F = F (Y F)$

Haskell-π Framework :

π-calculus style interprocess communication framework for Haskell

Haskell-π Framework provides reasonable solutions for following issues in concurrent programming:

Consistent Interprocess Communication



```
p = do {
  send a x; let t=func_p w; send b t; }

q = do {
  x<- recv a; send x y; w <- recv x; let u=func_q w; send x u; }
```

```
r = do {
  z <- recv x; let v=func_r z; send x v; }
```

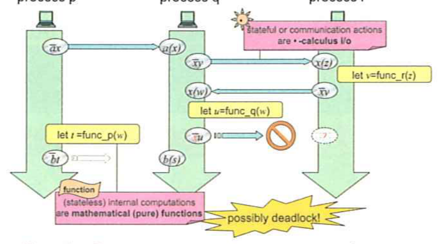
- no explicit synchronization
- channel based consistent style interprocess communication

```
semaphore pq_mutex = 0;
void p() {
  a = &x;
  up(&pq_mutex);
  t = func_p(w);
  BYTE* buf = (BYTE*)&t;
  int n = sizeof(t);
  while(n>0) {
   int sent = write(b, &buf, n);
   buf += sent;
   n -= sent;
 }
```

C language with UNIX API:
- error prone
- inconsistent style of interprocess communication (socket, mutex)

Separation of Concerns :

Separate communication Part From Purely (Mathematical) Function Part



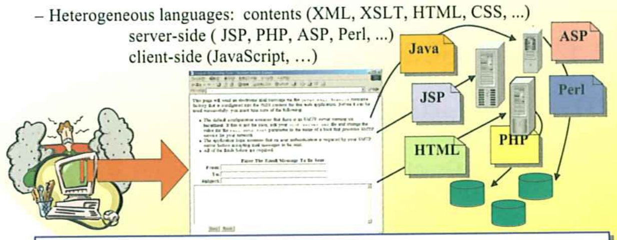
Deadlock freeness as a composite property from communication part and function part

Haskell-• aims at achieving high-reliability by formal verification of concurrency property

Validation and Verification for WebWare

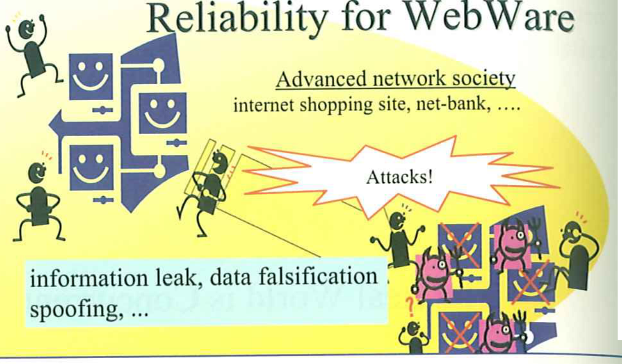
Features of WebWare Development

- Heterogeneous environment
  - Heterogeneous components: client, server, DB, WSDL, ...
  - Heterogeneous languages: contents (XML, XSLT, HTML, CSS, ...)
    server-side (JSP, PHP, ASP, Perl, ...)
    client-side (JavaScript, …)



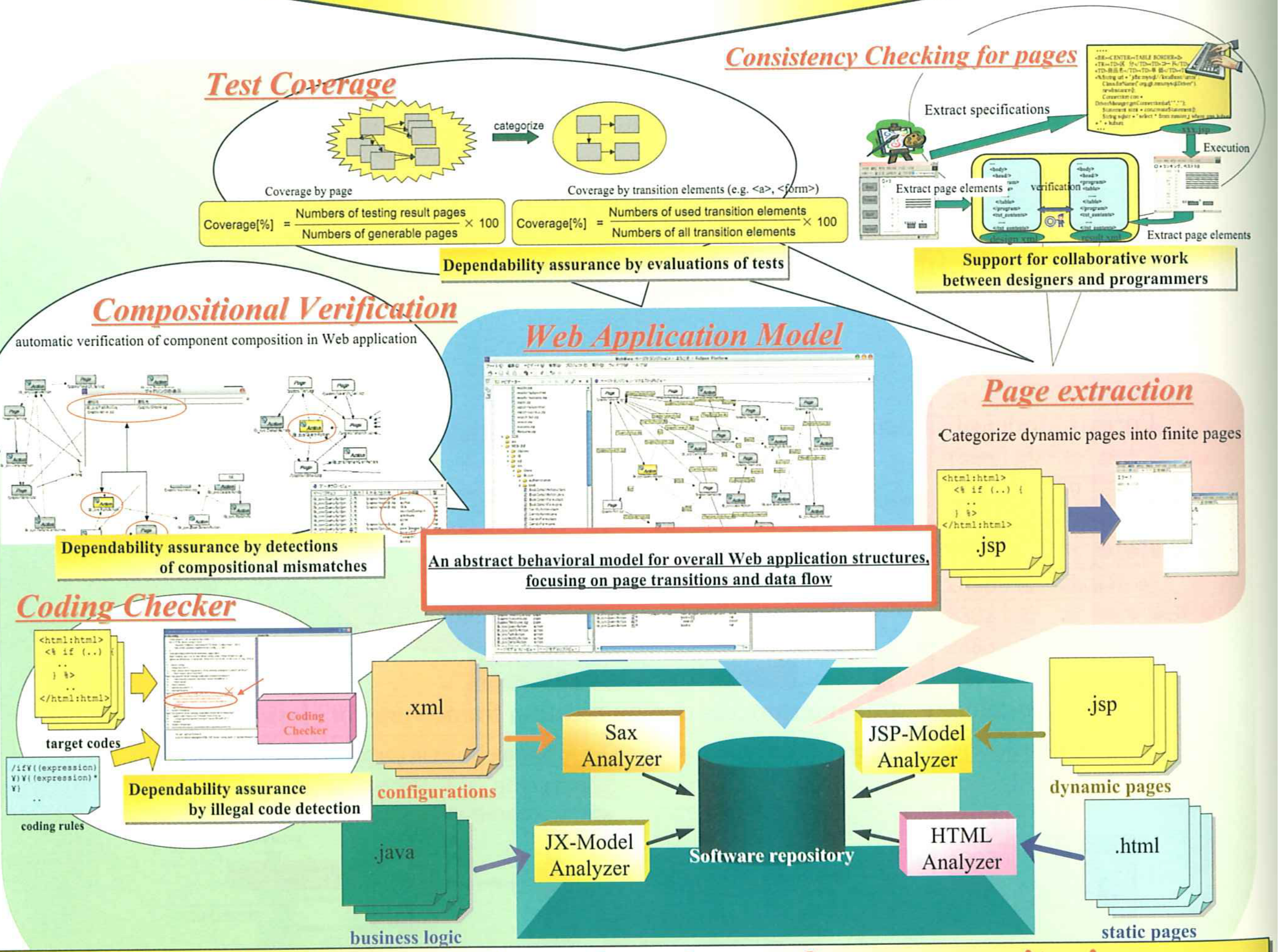
Dynamic composition of heterogeneous elements causes difficulty of WebWare testing.

Reliability for WebWare



Advanced network society
internet shopping site, net-bank, ….
Attacks!
information leak, data falsification
spoofing, ...

Dependability assurance for WebWare is one of social problems.



$$\text{Coverage}[\%] = \frac{\text{Numbers of testing result pages}}{\text{Numbers of generable pages}} \times 100$$

$$\text{Coverage}[\%] = \frac{\text{Numbers of used transition elements}}{\text{Numbers of all transition elements}} \times 100$$

Dependability assurance by evaluations of tests

Support for collaborative work between designers and programmers

automatic verification of component composition in Web application

Dependability assurance by detections of compositional mismatches

An abstract behavioral model for overall Web application structures, focusing on page transitions and data flow

Categorize dynamic pages into finite pages

```
<html:html>
  <% if (..) {
    ..
  } %>
</html:html>
```

```
/if¥((expression)
¥)¥((expression)*
¥)
  ..
```

Dependability assurance by illegal code detection

Sophisticated testing techniques based on software repositories

improve dependability of WebWare

Our results :

- Nomura Research Institute: ObjectWorks (http://works.nri.co.jp/) includes composition checker for WebWare.
- Nomura Research Institute: Ridual (http://www.ridual.jp) will include JavaScript analyzer.
- Fujitsu: Interstage (http://interstage.fujitsu.com/) includes HTML analyzer.

Agusa Lab., Graduate School of Information Science, Nagoya Univ. *http://www.agusa.i.is.nagoya-u.ac.jp/*