CS6201 Software Reuse

Lecture Notes: Set #2:

XVCL Briefing

XML-based Variant Configuration Language

http://xvcl.comp.nus.edu.sg

XVCL concepts

- **an x-frame**: a generic adaptable meta-component
  - program building block, at any level: architecture, subsystem, component, class, etc.
- **x-framework:** x-frames organized into a hierarchy, a PLA
- **x-frame "composition with adaptation"**
  - this is how XVCL Processor synthesizes a program from x-framework based on SPeCifications (SPC)

XVCL notation

- XVCL commands are XML tags:

```
<adapt x-frame="B.xvcl">
      <insert break = “x”>
        xAB
      </insert>
</adapt>
```

- we will use XML-free short notation:

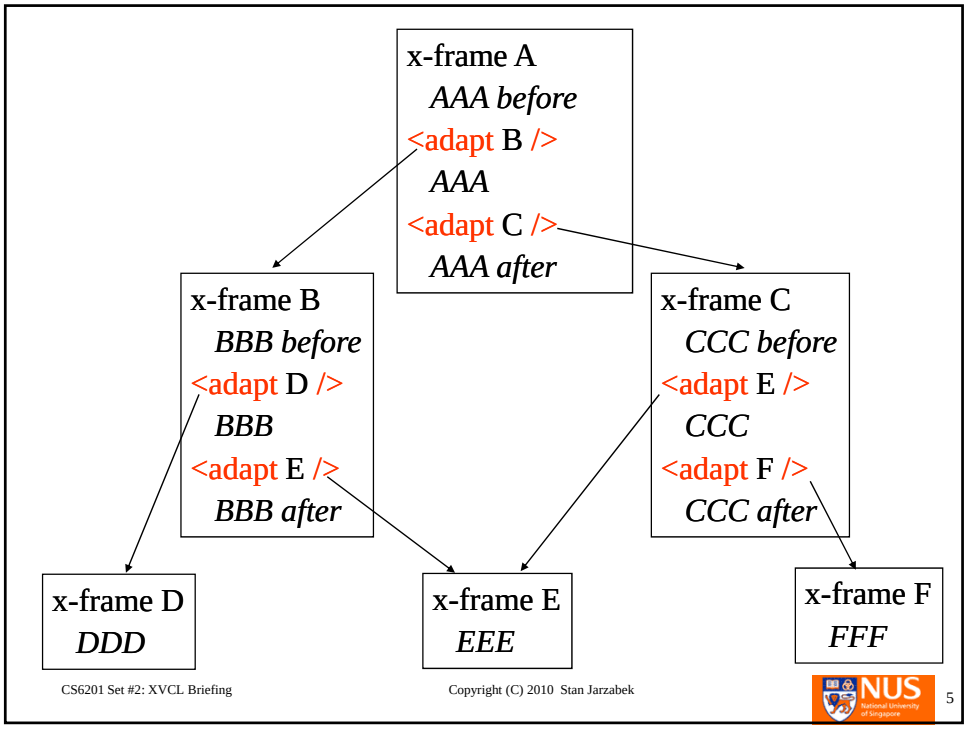
short notation is summarized on the previous slide

```
<adapt B >
      <insert x>
   xAB
```


Example of an x-frame

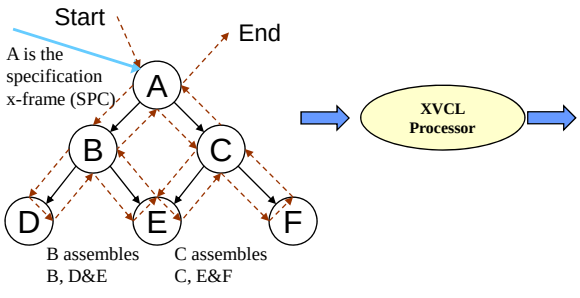
an x-frame contains base code instrumented with XVCL commands:

```
x-frame A
      AAA before        // some base code
   <adapt  B />
      AAA               // some base code
   <adapt C />
      AAA after         // some base code
```



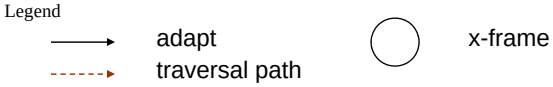
Composition of x-frames

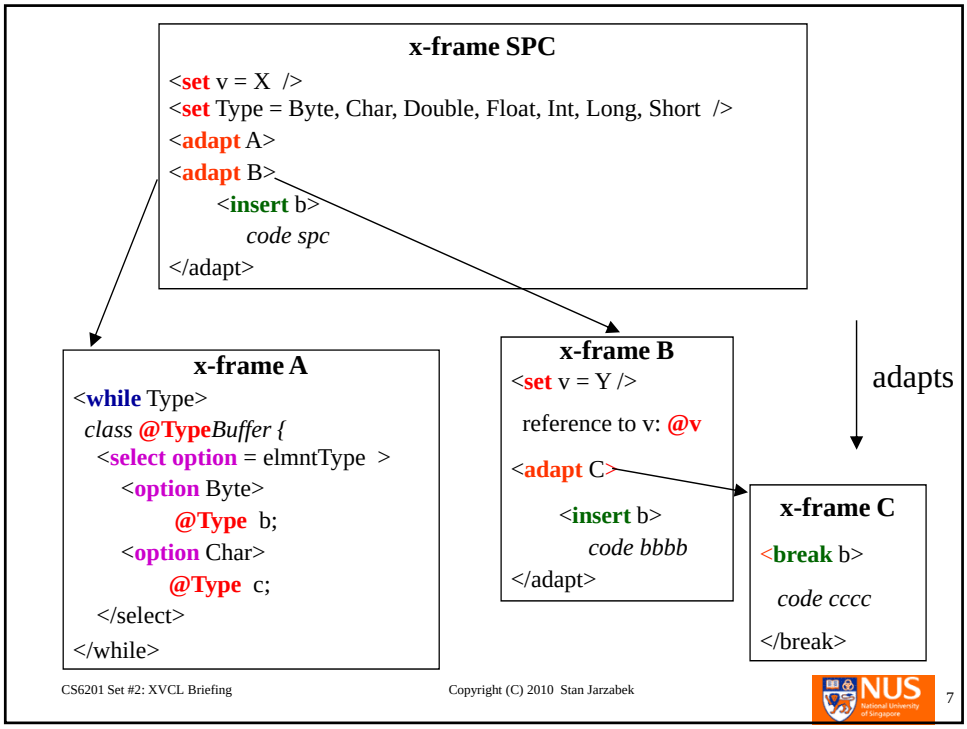
Order of Assembly = (ABDBEBACECFCA)



Result

AAA before
BBB before
DDD
BBB
EEE
BBB after
AAA
CCC before
EEE
CCC
FFF
CCC after
AAA after





Processing rules

- the processor traverses x-framework in depth-first order, as dictated by <**adapt**>s embedded in x-frames
- the processor interprets XVCL commands embedded in visited x-frames and emits a custom program into one or more files
- x-frames are read-only. The processor creates and modifies a copy of the adapted x-frame and never changes the original x-frame
- customization commands are specified for each <**adapt** A>
- recursive adaptations are not allowed

Generic design mechanisms of XVCL

- parameterization of x-frames and instantiation of parameters
- XVCL's parameters can be anything you can think of
  - composition of x-frames by <**adapt**>
  - generic names: variables and expressions
  - <**select**>ion among many given options
  - <**insert**>ions at <**break**>s
  - iteration with <**while**> (code generation)

Top-down propagation of customizations

- customizations specified in upper x-frames propagate down to <**adapt**>ed x-frames
  - customization propagate across levels of <**adapt**>ed x-frames
- overriding rule: customizations specified in upper x-frames override customizations specified in lower <**adapt**>ed x-frames

Details of XVCL mechanisms

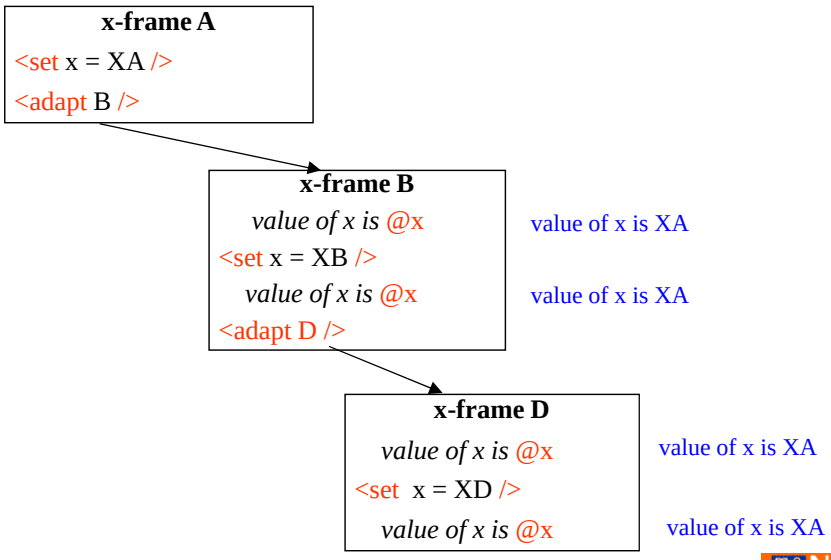
XVCL variables and expressions

- generic programs require generic names for program elements such as names of packages (Java), classes, methods, attributes

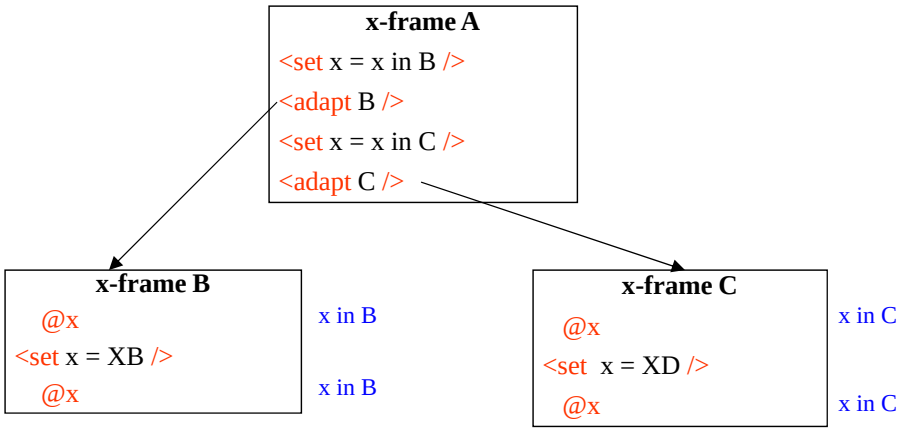
  <**set** color = Black />
  - assigns value "Black" to variable "color"

  <**set** colors = Black, White, Orange />
  - assigns a list of values to a multi-value variable "colors"
  - ',' is a separator, unless you use '\,' as an escape
- references to variables embedded in code: *xxxx*@color*yyyy*
  - XVCL Processor outputs: xxxxBlackyyyy
- values are type-less (with one exception, please check specs)

Variable propagation: example 1



Variable propagation: example 2



Variable propagation/scoping rules

- during processing, values of variables propagate top-down across x-frames visited by the processor
- <set> in an ancestor x-frame overrides <set> commands in all the subsequently processed descendent x-frames
  - each x-frame may set default values for variables
  - an ancestor x-frame (a "reuser") may override defaults set in <adapt>ed descendent x-frames

<select>

x-frame SPC

```
<set platform = Unix  />
<adapt A>
```

x-frame A

```
<select option = platform  >
      <option Unix >
            Unix-specific code here
      <option Windows >
            <adapt W-f> //Windows-specific code
      <option J2EE >
            J2EE-specific code here
      <otherwise>
            common code here
</select>
```

<select>

- <option>s are labeled with values
  - in general: conditions
- processor selects an option based on the value of the control variable
  - the first <option> whose value matches the value of the control variable is processed
- <option> may contain other XVCL commands:
  - <break>,
  - <adapt>,
  - <set>, <select>, <while>etc.

<while>

- <while> iterates over its body and generates many similar program structures based on a generic structure
  - e.g., subsystems, components, classes, …
- i^{th} iteration of <while> uses i^{th} value of a control variable

```
<set elmntType = Byte, Char, Double, Float, Int, Long, Short  />
…….
<while elmntType>
  class @elmntTypeBuffer {
    … }
</while>
```

Using <while> with <select>

- <while> is often used with <select>
- <option>s of <select> define differences among program structures to be generated in each iteration
- Example:
  - We want to generate a group of similar Buffer classes
  - We define common class parts in GenericBufferClass
  - We place <select> in <while> generation loop
  - Each <option> specifies customizations for a given classs
  - Classes that require the same or no customizations are generated in <otherwise>


```
<set elmtType = Byte, Char, Double, Float, Int, Long, Short/>
<while elmtType >
class @elmtTypeBuffer {
    <select option = elmtType>
    <option Byte>
        <adapt GenericBufferClass >
            customizations for class ByteBuffer
        </adapt>
    <option Char>
        <adapt GenericBufferClass >
            customizations for cass CharBuffer
        </adapt>
    <otherwise>
        <adapt GenericBufferClass  />
            customizations for the remaining Buffer classes
        </adapt>
    </select>
</while>
```

processor output:

```
class ByteBuffer { …
class CharBuffer { …
class DoubleBuffer { …
class FloatBuffer { …
class IntBuffer { …
class LongBuffer { …
class ShortBuffer { …
```

More about <while>

- <while> can use many variables
  - all the variables in <while> must have same number of values

```
<set elmntType = Byte, Char, Double, Float, Int, Long,
   Short  />
<set number = 1,2,3,4,5,6,7 />
.............
<while elmtType, number >
@elmntType@number
</while>
```

processor output:
Byte1
Char2
Double3
Float4
Int5
Long6
Short7

<ifdef> and <ifndef>

- execute if body if variable XXX is defined:

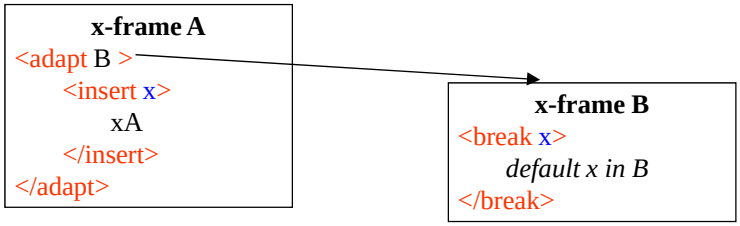
```
<ifdef var = XXX>
        if body
</ifdef>
```

- execute if body if variable XXX is not defined:

```
<ifndef var=XXX>
        if body
</ifndef>
```

<insert> into <break>s

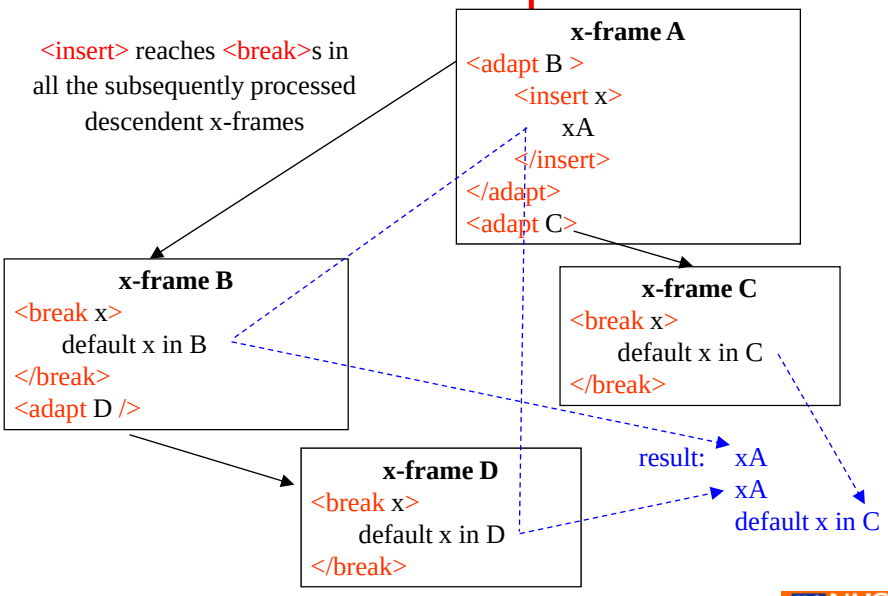
- <break> marks a variation point
- <insert> replaces the contents of matching <break>s



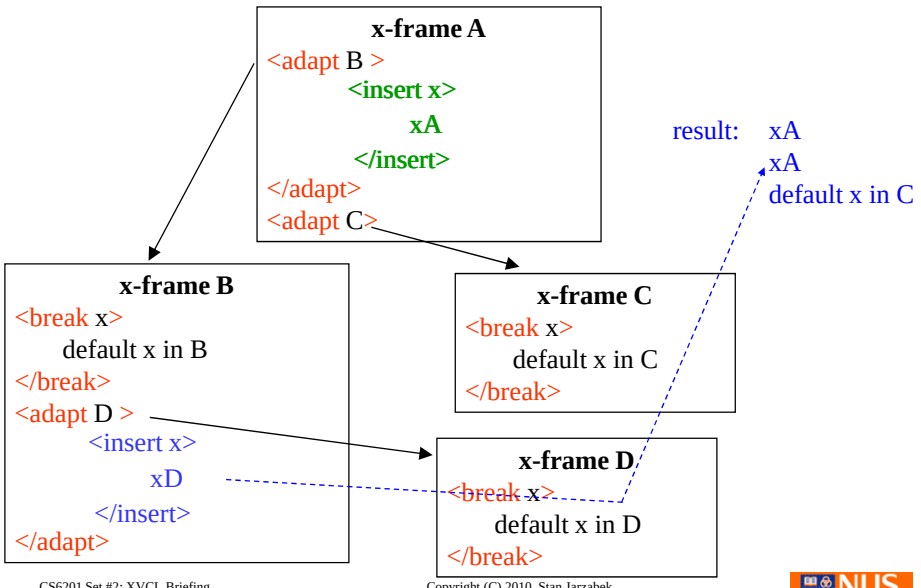
result : xA

- if there is no <insert> matching a given <break>, the default code contained in the <break> is processed

<insert> into multiple <break>s



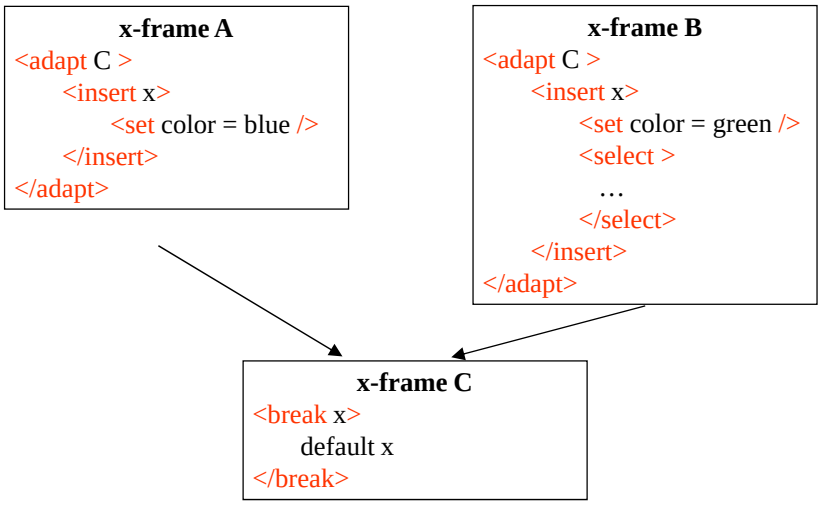
Upper <insert> overrides lower <insert>s



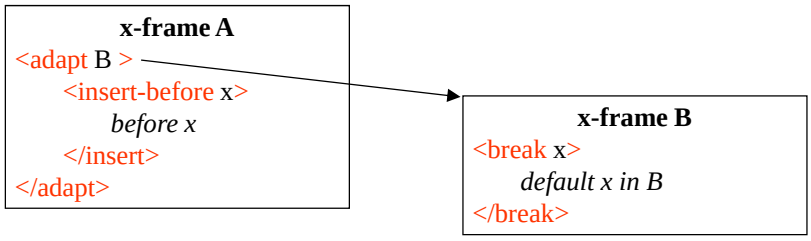
<insert> propagation/scoping rules

- during processing, <insert> is propagated to all the subsequently adapted x-frames
- only the <insert> command executed first in the processing flow matches (affects) a <break>
  - it overrides <insert> commands that may appear in all the subsequently adapted x-frames

x-frame reuse in context

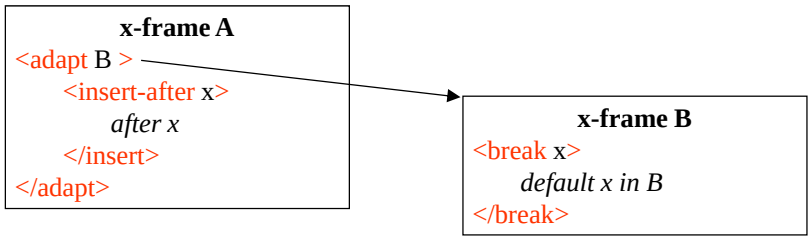


<insert-before>



result : *before x*
default x in B

<insert-after>

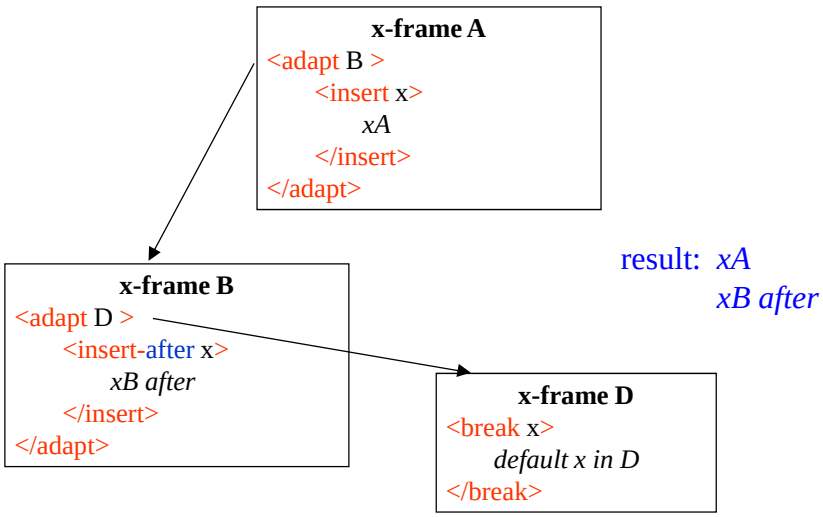


result : *default x in B*
after x

Comments

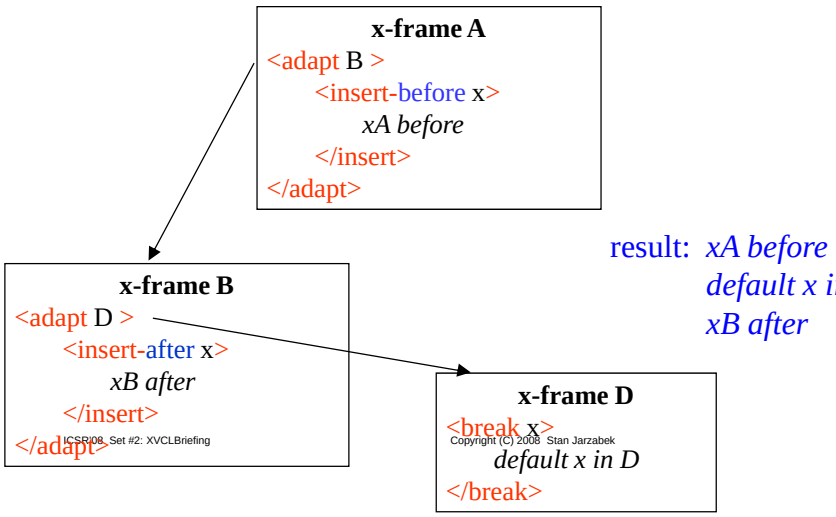
- propagation, scoping and overriding rules for <insert-before> and <insert-after> are the same as for <insert>
  - first <insert-before> overrides any subsequent ones
  - first <insert-after> overrides any subsequent ones
- each of the <insert>, <insert-before> and <insert-after> commands may match (affect) the same <break>, yielding accumulative result (see next slides)

Example (1): accumulative result



result: *xA*
xB after

Example (2): accumulative result



result: *xA before*
default x in D
xB after

Multiple <insert>s in <adapt>

x-frame A

```
<adapt B >
    <insert-before x >
        xAB before
    </insert>
    <insert x >
        xAB
    </insert>
    <insert-after x >
        xAB after
    </insert>
    <insert x >
        xAB again
    </insert>
</adapt>
```

x-frame B

```
before break x in B
<break x>
    default x in B
</break>
after break x in B
```

result and explanation on the following slide

Multiple <insert>s in <adapt>, cont.

result:

```
before break x in B
xAB before
xAB
xAB again
xAB after
after break x in B
```

Comments (from XVCL specs): multiple <insert>, <insert-before> and <insert-after> commands into the same <break> may be intermixed in the body of a single <adapt>.

This results in concatenation of the contents of all the <insert>s, before any insertion is conducted.

Referring to above example, x-frame A has two <insert>s with the body "xAB" and "xAB again" that are concatenated during processing.

What can we <insert>?

- code
- other XVCL commands:
  - <set x = v>
  - <select>
  - <while>
  - <adapt A>
  - etc

Comments on <insert> <break>

- nested <break>s are not allowed
- recursive adaptations are not allowed

Using <select> and <insert>

- we should use <select> to cater for anticipated variant features
- we should use <insert> only if we want to address unexpected new features without changing existing x-frames

Top-down propagation of <set> and <insert>

- customizations specified in upper x-frames propagate down to <adapt>ed x-frames
  - customization propagate across levels of <adapt>ed x-frames
- overriding rule: customizations specified in upper x-frames override customizations specified in lower <adapt>ed x-frames

XVCL expressions

- direct and indirect references to variables

  given a variable C:

  @C – value-of (C)

  @@C – value-of (value-of (C))

  @…@C

- name expression – a simple form of an expression

  @x@y@C - value-of (“x” | value-of (“y” | value-of (C)))

  symbol ‘|’ means concatenation

Name expressions: Example

Name	Value
A	X
X	Y
Y	Z
C	U
BU	V
AV	W

steps in evaluating name expression @A@B@C:

- get value-of (C) which is U
- get value-of (BU) which is V
- get value-of (AV) which is W

  Result: W

General form of expressions

Evaluation of ?@A@B@C?P?@X?

in XML syntax, ? are used as separators

1. evaluate name expression @A@B@C
   - the result is: W
2. replace @A@B@C with W
   - partially evaluated result is: ?W?P@X?
3. evaluate @X
   - the result is: Y
4. replace @X with Y
   - the final result is: WPY

Comments on expressions

Variables and expressions provide powerful means for creating generic names and for controlling x-framework customization process

Expressions may contain any number of name expressions intermixed with character strings. To evaluate a expression, we evaluate all the name expressions and concatenate resulting values with character strings. The result replaces the corresponding expression.

XVCL processor stores all the existing variables in the Symbol Table along with their current values, as assigned to variables in <set> commands.

You can't underestimate the role of expressions in designing XVCL representations

- anything except x-frame name can be defined by XVCL expression :

```
<set x = 1, @x, 2 />
<adapt @x>
<insert @x>
<break @x>
```

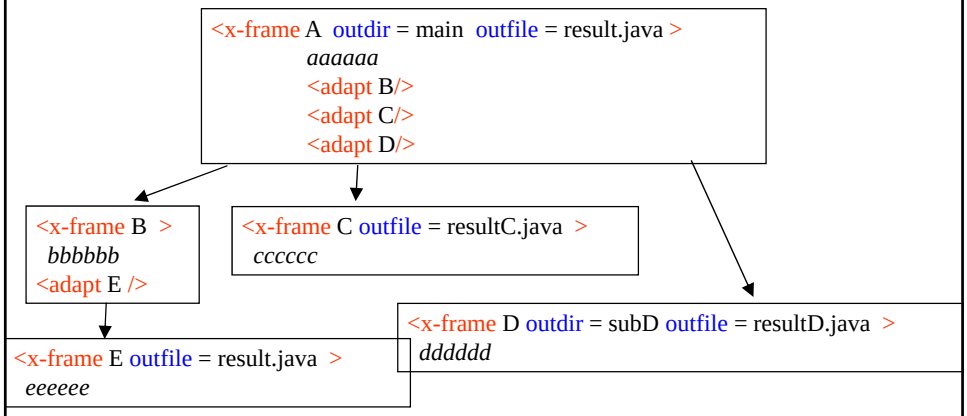
- check Notepad example on xvcl.comp.nus.edu.sg for advanced use of XVCL expressions

Attributes outdir and outfile

<x-frame A outdir = main outfile = result.java >

- outdir and outfile specify the output file to which the XVCL processor will emit the result
- values of outdir & outfile may be given by an expression
- outdir can be a either absolute path or a partial path
- outfile can be a file name or absolute path
  - relative path is not allowed for the outfile

Attributes outdir and outfile cont.



output:
File: \main\result.java
aaaaaa
bbbbbb
eeeeee

output:
File: \main\resultC.java
cccccc

output:
File:
\main\subD\resultD.java
dddddd

Example : outdir and outfile

X-framework

```
<x-frame SPC outdir = accounts
         outfile = account.java>

<set className = SavingsAccount, LoanAccount />
<set outputFile = savings.java, loan.java/>
<set outDir = savings, loan />

<while className, outputFile, outDir >
    <adapt Account outdir =@outDir
           outfile = @outputFile />
</while>
</x-frame>


<x-frame  Account>
class  @className  {
  public static void main(String[] args)  {
   }
}
</x-frame>
```

Result

file accounts\savings\savings.java:

```
class  SavingsAccount  {
  public static void main(String[] args)  {
   }
}
```

File accounts\loan\loan.java:

```
class  LoanAccount  {
  public static void main(String[] args)  {
   }
}
```

XVCL Processor options

- options modify Processor's behavior

option -A : avoids deleting the outfile

normally, when the processor emits output to outfile for the first time in a given run, and the outfile exisit, it is deleted before the output is emitted

option –A tells the processor not to delete outfile, but to append emitted output to outfile even if it already existed before the processor was invoked

option – A has no impact on further processing, that is, any further output is directed to outfile is appended to outfile

option -B : output beautifier: trims extra white spaces that otherwise appear on the output

Misc

- you can prefix XVCL commands with <xvcl: > to avoid name collision

e.g., instead of <adapt> use: <xvcl:adapt>

- including non-XVCL files into the generated system

<adapt A src=yes >

  - tells the processor to output file A without processing it. The default value is "no", meaning a proper x-frame.

Advanced features

- attributes **samelevel** (in <adapt>) and **defer-evaluation** (in <set>) may affect the value of meta-expression – see XVCL specs
- attributes **samelevel** of <adapt>
  - to change variable scoping rules
- attribute **defer-evaluation** of <set>
  - to evaluate value of a meta-variable at the point of reference
- arithmetic expressions

Summary of important XVCL commands in XML format

XVCL Command	Description
<x-frame **name**= ”name” [**outdir**=”dir-name”] [**outfile** = ”file-name”] > x-frame body: mixture of code and XVCL commands </x-frame>	An x-frame is a generic, adaptable component. An x-frame contains code instrumented for ease of adaptation with XVCL commands
<adapt **x-frame**=”name” [**outdir**=”dir-name”] [**outfile** = ”file-name”] adapt-body : mixture of <insert> commands </adapt> (or) <adapt **x-frame**=”name” [**outdir** =”dir-name”] [**outfile** =”file-name”]/>	When <adapt> command is encountered, the processor: 1. adapts the x-frame “name” by executing the commands listed in the adapt-body 2. includes the adapted x-frame “name” into the current x-frame 3. resumes processing of the current x-frame

<**break** **name** =”break-name”> break-body </break>	<break> marks a break point at which an x-frame can be adapted by other x-frames via <insert> commands; the break-body defines the default code that may be replaced or extended by <insert> commands
<**insert break** = ”break-name”> insert-body </insert> <**insert-before break** = ”break-name”> insert-body </insert-before > <**insert-after break**=”break-name”> insert-body </insert-after >	Replaces break point “break-name” in the adapted x-frame with the insert-body. Inserts the insert-body **before** the break point “break-name” in the adapted x-frame. Inserts the insert-body **after** the break point “break-name” in the adapted x-frame.

Command	Description
<**set var** = ”var-name” **value** = ”value” />	Assigns “value” to variable “var-name”
<**set-multi var**=” var-name” **value**=”value1, value2, …” />	Assigns values to a multi-value variable.
<**value-of expr** = ”expression” />	The value of the expression is evaluated and the result is inserted in place of the <value-of> command

Command	Description
<**select option** = ”var-name”> select-body: includes options listed below </**select**> list of options: <**option-undefined**> (optional) option-body </**option-undefined**> (optional) <**option value** = ”value”> (0 or more) option-body </**option**> <**otherwise**> (optional) option-body </**otherwise**>	Select from a set of options based on variable “var-name” as follows: <**option-undefined**> - if the multi-value variable “var-name” is undefined <**option**> - if value of “var-name” matches option’s “value” <**otherwise**> - if none of the <option>’s “value” is matched

Command	Description
<**ifdef** var = ”var-name”> if-body: below </**ifdef**> <**ifndef** var = ”var-name”> if-body: below </**ifndef**>	These two commands allow us to execute the if-body based on the presence or absence of the specified variable.
<**while using-items-in**=”multi-var”> while-body </**while**>	Iterates over while-body: i’th iteration uses i’th value of the multi-valued variable “multi-var”


XVCL Workbench Demo

End of XVCL briefing