#### 3.5.3.5 *Binding References to Components*

[Definition: The process of identifying the component to which a symbolic reference applies (possibly chosen from several homonymous alternatives) is called **reference binding**.]

The process of reference binding in the presence of overriding declarations is best illustrated by an example. The formal rules follow later in the section.

Consider a package Q defined as follows:

```
<xsl:package name="Q" xmlns="http://www.w3.org/1999/XSL/Transform">

  <xsl:variable name="A" visibility="final" select="$B + 1"/>
  <xsl:variable name="B" visibility="private" select="$C * 2"/>
  <xsl:variable name="C" visibility="public" select="22"/>

</xsl:package>
```

(The process is illustrated here using variables as the components, but the logic would be the same if the example used functions or named templates.)

There are three components in this package, and their properties are illustrated in the following table (the ID column is an arbitrary component identifier used only for the purposes of this exposition):

| ID | Symbolic Name | Declaring Package | Visibility | Body | Bindings |
|---|---|---|---|---|---|
| $A_Q$ | variable A | Q | final | $B + 1 | $B → $B_Q$ |
| $B_Q$ | variable B | Q | private | $C * 2 | $C → $C_Q$ |
| $C_Q$ | variable C | Q | public | 22 | none |

Now consider a package P that uses Q, and that overrides one of the variables declared in Q:

```
<xsl:package name="P" xmlns="http://www.w3.org/1999/XSL/Transform">

  <xsl:use-package name="Q">
    <xsl:override>
      <xsl:variable name="C" visibility="private" select="25"/>
    </xsl:override>
  </xsl:use-package>

  <xsl:template name="T" visibility="public">
    <xsl:value-of select="$A"/>
  </xsl:template>

</xsl:package>
```

Package P has four components, whose properties are shown in the following table:

| ID | Symbolic Name | Declaring Package | Visibility | Body | Bindings |
|---|---|---|---|---|---|
| $A_P$ | variable A | Q | final | $B + 1 | $B → $B_P$ |
| $B_P$ | variable B | Q | hidden | $C * 2 | $C → $C_P$ |
| $C_P$ | variable C | P | private | 25 | none |
| $T_P$ | template T | P | public | value-of select="$A" | $A → $A_P$ |

The effect of these bindings is that when template T is called, the output is 51.

In this example the components in P are established in three different ways:

- $A_P$ and $B_P$ are modified copies of the corresponding component $A_Q$ and $B_Q$ in the used package Q. The symbolic name, declarating package, and body are unchanged. The visibility is changed according to the rules in section 3.5.3.2 Accepting Components: specifically `private` changes to `hidden`. The references to other named components are rebound as described in this section.
- $C_P$ is the overridden component. Its properties are exactly as if it were declared as a top-level component in P (outside the <xsl:override>), except that (a) it must adhere to the constraints on

overriding components (see 3.5.3.3 Overriding Components), and (b) the fact that it overrides $C_Q$ affects the way references from other components are rebound.

- $T_P$ is a new component declared locally in P.

The general rules for reference binding can now be stated:

1. If the containing package of a component $C_P$ is P, then all symbolic references in $C_P$ are bound to components whose containing package is P.
2. When a package P uses a package Q, then for every component $C_Q$ in Q, there is a corresponding component $C_P$ in P, as described in 3.5.3.2 Accepting Components. [Make this a defined term.] Note: If $C_Q$ is overridden in P, then the corresponding component is the overriding component; in other cases it is a component that has the same symbolic name, declaring package, and body as $C_Q$, but which in general has different visibility, and different bindings for its symbolic references.
3. If the declaring package of component $C_P$ is the containing package P, there will always be exactly one non-hidden component $D_P$ within P whose symbolic name matches the name of the symbolic reference (a static error will have been reported, as described elsewhere in this specification, if this is not the case), and the reference is then bound to $D_P$.
4. If the declaring package of $C_P$ is some other package R  P  (that is,$C_P$ is present in P by virtue of an `<xsl:use-package>` declaration), then the component bindings in $C_P$ are derived from the component bindings present in its corresponding component $C_Q$ as follows: If the component binding within $C_Q$ is to a component $D_Q$ in Q, then the new binding in the $C_P$ will be to the component $D_P$ in P whose corresponding component is $D_Q$.

When reference binding is performed on a package that is intended to be used as a stylesheet (that is, for the top-level package), there must be no symbolic references referring to components whose visibility is `abstract` (that is, an implementation must be provided for every abstract component).

[ERR XTSE3080] It is a static error if a top-level package (as distinct from a library package) contains symbolic references referring to components whose visibility is `abstract`.

**Note:**
This means that abstract components must be overridden in a using package either by a component that supplies a real implementation, or by a component with visibility="absent" (see 3.5.3.2) whose effect is that any invocation of the component results in a dynamic error.

**Note:**
Unresolved references are allowed at the module level but not at the package level. A stylesheet module can contain references to components that are satisfied only when the module is imported into another module that declares the missing component.

**Note:**
The process of resolving references (or linking) is critical to an implementation that uses separate compilation. One of the aims of these rules is to ensure that when compiling a package, it is always possible to determine the signature of called functions, templates, and other components. A further aim is to establish unambiguously in what circumstances components can be overridden, so that compilers know when it is possible to perform optimizations such as inlining of function and variable references.

Suppose a public template *T* calls a private function *F*. When the package containing these two components is referenced by a using package, the template remains public, while the function becomes hidden. Because the function becomes hidden, it can no longer conflict with any other function of the same name, or be overridden by any other function; at this stage the compiler knows exactly which function *T* will be calling, and can perform optimizations based on this knowledge.

The mechanism for resolving component references described in this section is consistent with the mechanism used for binding function and variable references described in the XPath specification. XPath requires these variable and function names to be present in the static context for an XPath expression. XSLT ensures that all the non-hidden functions, global variables, and global parameters in a package are present in the static context for every XPath expression that appears in that package, along with required information such as the type of a variable and the signature of a function.