Extending CSCM to Support Interface Versioning

By

Hamdan Msheik

Alain Abran

Agenda

- Background on Components
- CSCM components model
- Interface versioning problem
- An approach to tackle the versioning problem
- Current and Future Work
- Conclusion

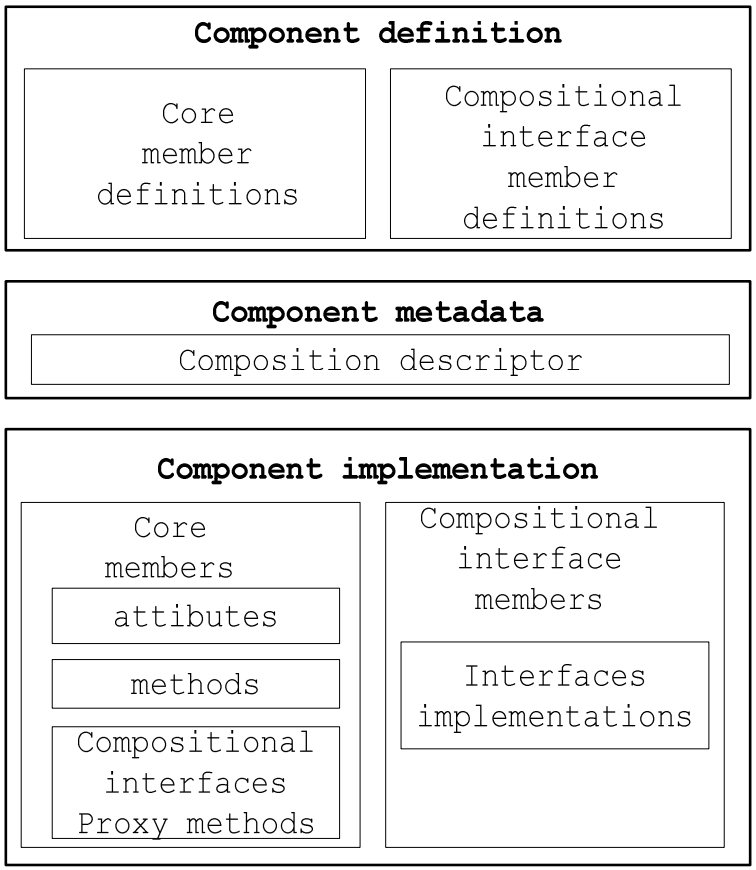
Background on Components

- Dates back to 60's, currently considered as a computing revolution [1]
- Reuse, quality, easier, faster development
- Provides public interfaces
- Components act as containers for functionalities
- Component (class) structure results in unwanted functionalities [2]
- Reuse requires wrapping, adaptation and customization

CSCM Model

- Handles the issue of unwanted functionalities
- Allows a declarative selective composition (inclusion and exclusion) of functionalities
- Provides an easier and more flexible SD approach for software customization, adaptation and reuse

CSCM Component Structure




```
import java.io.File;

Component Compressor {
  public String
   displayUsage(){…}

   // compositional Interfaces
   // zip algorithm
   Compositional public File
   zip (File f,

   String oper){}
   // gzip algorithm
   compositional public File
   gzip(File f,

   String oper){}
}
```

Interface Versioning Problem

- CSCM components and interfaces are subject to change
- Component interfaces express functional aspects only
- No guarantee that newer components and interfaces (methods) versions still compatible with older ones
- Under the mercy of the supplier's release notes
- What if you upgraded to an incompatible version?

An Approach to Tackle Components Interface Versioning

- Decorate components and their interfaces with versioning metadata
- Use Java5 annotation feature
- Load-time time detection of components versions incompatibilities
- Prevents
  - Unexpected application failure
  - Data loss or corruption
- Fine-grained detection of incompatibilities on the component and interface (method) levels

An Approach to Tackle Components Interface Versioning

- Developer responsible for metadata versioning information
- Following industrial conventions and standards

Versioning Scheme

- Associate each component and each of his interfaces with versioning data
  - Major version
  - Minor version
  - Micro version
- Each interface is associated with metadata annotation specifying all component versions with which it is compatible

Example Component Definition

```
import java.io.File;
@ComponentVersion( name="Compressor",
                   major = 1,
                   minor = 2,
                   micro = 1)
Component Compressor {
  public String displayUsage(){…}

  // compositional  members  //
  zip algorithm
  Compositional public File zip(File f, String oper){}
}
```

Core component partial implemenation

```
@ComponentVersion (
   name = "Compressor",
   major = 1,
   minor = 2,
   micro = 1)
public class Compressor {
  /* this code is generated by CSCM
     compiler */

// core method
  public String displayUsage(){
  return new String();
  }
  // compositional interfaces
  Private File zip(File f,String oper){
  return((zipClass)
      compositionalInterfaces.get(
             "zip")).zip(f, oper);
   }
 }
```

A Component Version Annotation

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
@interface ComponentVersion {
   String name()
   int mainVersion ();
   int minorVersion();
   int microVersion();
}
```

Interface Version Annotation

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
@interface InterfaceVersion {
 ComponentVersion interfaceVersion();
 ComponentVersion []
            compatibleComponentVersion();
}
```

Partial interface implementation

```
/* The code is generated by the CSCM
     Compiler */
@InterfaceVersion(
   interfaceVersion =
     @ComponentVersion (name = "zipClass",
        major = 3, minor = 3, micro = 1),
     compatibleComponentVersion = {
        @ComponentVersion ( name = " zipClass",
           major = 3, minor = 3, micro = 1)}
  )
Public class zipClass {
   public File zip(File f, String oper) {
  _  _This.displayUsage();
      return new File("testTextFile.txt");
   }
}
```

Conclusion

- Scheme for components interface versioning
- Based on standard conventions
- Exploit the Java 5 annotation feature
- Safe application termination
- Prevents data corruption or loss
- Provide a reliable method for detecting interface incompatibilities

Thank You !

hamdan.msheik.1@ens.etsmtl.ca, aabran@ele.etsmtl.ca,

References

1. P. Maurer, *Component-Level Programming*: Pearson education Inc., 2003.
2. M. S. Al-Hatali and H. G. Walton, "Smart Features for Compositional Wrappers," ICSR7 2002 Workshop on Component-based Software Development Processes, Austin, Texas, 2002.