A primary goal of object-oriented (OO) program design is to minimize the impact of future change. This is enabled through the OO language mechanism of inheritance in which original program components can be *extended* without being modified. However, users of components are not necessarily the producers of them. Despite the benefits offered by OO design, adaptation of components by those with access to documented interfaces only can be extremely error-prone and arduous. Binary Component Adaptation (BCA) has been proposed recently to automate this process. BCA enables component users to adapt components for interoperability with and compatibility across all other components in an application. BCA rewrites components to reflect the necessary changes without access to the source code or requiring recompilation of the application. Rewriting takes place prior to or during dynamic loading of a program. As a result, fewer errors are made and component interoperation is enabled with little effort by the user.

BCA enables modification prior to execution of the component. However, it may prove to be economically infeasible for application execution to be halted while component integration or update is performed. For such cases, we propose to extend BCA to enable adaptation to occur *while* the program is running. Our extension, *Runtime Binary Component Adaptation* (RBCA) will provide software administrators with a choice as to when BCA should occur to be most effective.

To implement RBCA, we will use the Java Virtual Machine as our experimental language infrastructure; however, our ideas are general and we believe that they will extend easily to other dynamically loaded languages for which symbolic information is made explicitly available by the program transfer format, e.g., Microsoft Intermediate Language (MSIL) and the Common Language Runtime (CLR). We will extend the Open Runtime Platform (ORP) from Intel Corporation to enable runtime adaptation of Java program components. The design of RBCA is shown in Figure 1 and consists of a graphical user interface (GUI) to enable application adaptation, and a RBCA-aware runtime system within which the executing program is internally represented to facilitate component rewriting at runtime.

We exemplify the use of this RBCA design with a small example. Assume that we have a class A that has already been dynamically loaded as part of an executing application (and therefore is represented in the runtime system). In addition, Class A has a method, called *output*, that outputs program state. While the program is running, we acquire new components that we intend to incorporate into our running application. The components also report program state but use the Printable interface to do so. To enable class A and the new components to uniformly accept future print requests, class A must be modified. The following BCA delta can be generated to reflect this change (the partial class is on the left and the delta is on the right):

```
class A {
    public void output(PrintStream os);
    . . .
}
```

```
delta class A {
    add interface Printable;
    add method
       public void print() {
           output(System.out);}
}
```

The delta is the same regardless of whether or not class A has been loaded yet. With the current implementation of BCA, a component cannot be modified once the class in which it is contained has been loaded. However, we can perform the adaptation using the runtime extension (RBCA) that we propose. The application administrator uses the RBCA GUI to generate the above delta for class A and to submit an update request to the runtime system. The internal representation of A is updated to reflect that A implements the Printable interface and the method print is added. The bytecode for the method (submitted as part of the delta) is stored with the internal class file representation of A and is interpreted when invoked. At which time, verification is performed on the method to ensure appropriate (well-formed) execution behavior. Compatibility is ensured since all previously loaded code will use the original *output* method (that still exists) and any newly loaded code can use either mechanism for printing. If changes to bytecode instructions within loaded methods are required (for naming and call site changes, etc.), all stored bytecode that is affected by the change is modified to reflect the changes; future invocations of the methods use the modified versions of the code.

We plan to use an interpreter for program execution initially to simplify the RBCA implementation. However, we also plan to investigate the interactions of RBCA and dynamic compilation as part of this project. Dynamic compilation can substantially improve execution performance but additional overhead and system complexity are needed for RBCA implementation. For example, checks must be inserted (via dynamic instrumentation) into the native code stream to ensure that updates are accurately reflected by the program execution. We believe that by using ORP, our task will be simplified. ORP is an adaptive compilation environment for which multiple compilers are used to dynamically monitor and re-compile code to improve execution performance (and minimize compilation overhead). We plan to extend this system to dynamically monitor update events and to use recompilation to incorporate all updates to the executing program. We will empirically evaluate both protoypical versions of RBCA (interpreter- and compiler-based) to identify the strengths and weaknesses of this approach.

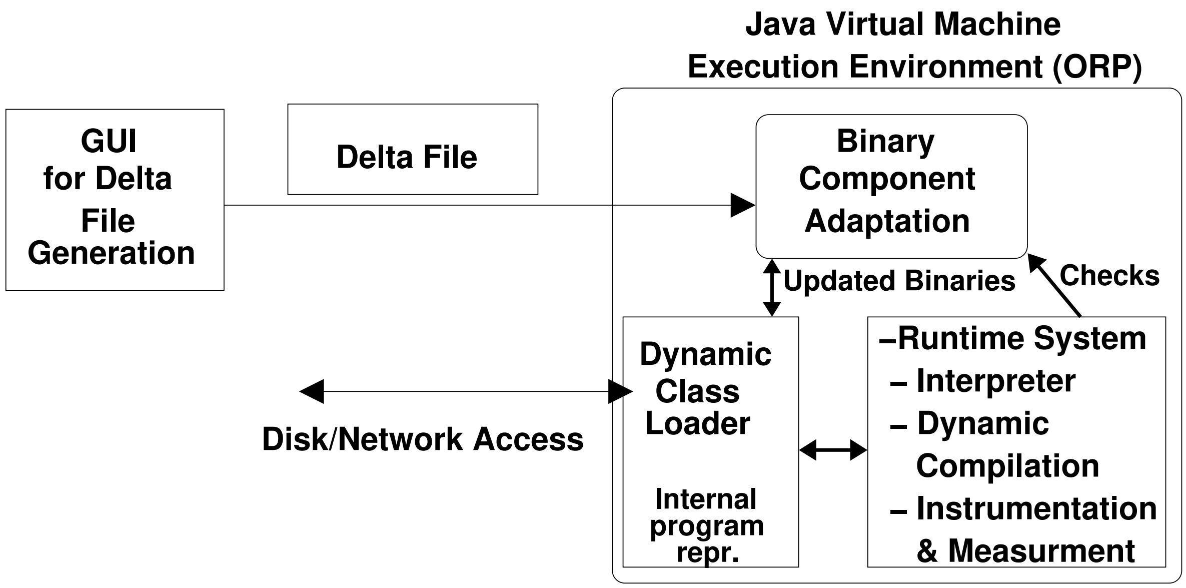

Figure 1: The proposed design of Runtime Binary Component Adaptation (RBCA). A Java Virtual Machine execution environment called the Open Runtime Platform (ORP) from Intel Corporation will be extended to enable RBCA. Normal execution proceeds using existing dynamic class file loading technology. Programs can be interpreted or dynamically compiled. Our initial RBCA work will be performed using an interpreter for execution. When an administrator wishes to adapt a component that has already been loaded as part of the executing application, he/she can use the RBCA GUI to generate a delta file of the changes and to submit the request to the extended runtime system. The delta file is used to adapt the affected portion of the loaded program (the program's internal BCA representation is modified). The interpreter (or dynamically compiled and instrumented executing program) checks for updates at method entry. When modifications have been made, the class loader generates a new representation of the modified binary and passes it to the runtime system for incorporation and execution continues.