

EXECUTIVE SUMMARY

The integration of heterogeneous software components is commonplace in software development. Using pre-existing in-house components, COTS, and GOTS to build new software systems should speed up development time. However, interoperability conflicts often appear within integrated systems that hinder decreases in development time and cost.

Interoperability conflicts are defined as conflicting communication needs or expectations among independent components that form an integrated system. So that they do not further complicate the development process, interoperability conflicts should be traceable to the design-based reasons why they occur. Many of these reasons are due to data and control interactions and may be detected by examining the respective software architecture of each component. Identifying these problems reduces the development time and cost associated with integration and ensures that the implemented middleware is reliable.

Collidescope is an interactive interoperability assessment tool based on software architecture properties and analytical analysis of component integration expectations. Collidescope identifies high-level interoperability problems. It then maps these conflicts to individual integration strategies that can be composed to form the architecture of the necessary middleware. The history of an assessment can be saved in an XML file for future retrieval.

TECHNICAL OVERVIEW

Collidescope is composed of 4 components: a graphical user interface (GUI), a database, an XML Library, and a rule engine (Figure 1). The developer uses the GUI to describe the characteristics of an application to Collidescope. The information the developer inputs is then written to an XML file that is stored within the Library.

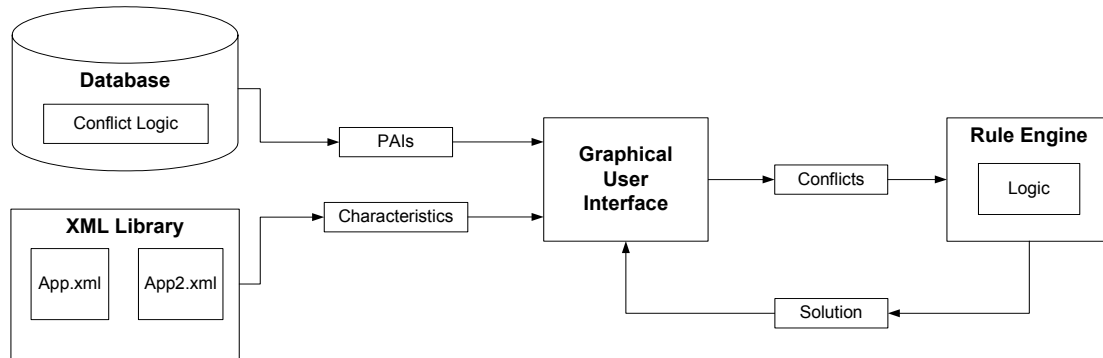


Figure 1. The Collidescope Architecture

Below is a simplified version of the XML file that Collidescope uses to describe an application. The application has a set of characteristics defined, a list of components, and a set of links that describe communication between the components.

```
<application name="Example Application">
  <characteristics>
    ...
  </characteristics>
  <components>
    <component name="Component A">
    <component name="Component B">
    ...
  </components>
  <links>
    <link>
      <component name="Component A" />
      <component name="Component B" />
    </link>
    ...
  </links>
</application>
```

To support modular components that can be re-used, the characteristics of components are actually stored in a separate file as seen below.

```
<component name="Component A">
  <characteristics>
```

```
...  
</characteristics>  
</component>
```

Once all of the information required to describe an application has been entered into the GUI the developer can ask Collidescope to perform a set of tests that determine if any problematic architecture interactions (PAIs) will occur. These problematic interactions are sets of conflicting characteristics within the application that could lead to communication failure. Finding the PAIs involves querying the database with the characteristics of the application and components therein. The database is a Microsoft Access database that contains a complex system of conflict logic that returns PAIs to the GUI.

If Collidescope is able to find any problematic interactions a set of conflicts are sent to a rule based engine to determine if a proposed solution can be generated. The rule system is implemented using JESS, the Java Expert Shell System and it contains 13 different rules that can be triggered by PAIs. These rules are used to propose integration enablers that can be inserted between components to help aid communication within the application. When the rule engine is done inserting enablers the solution is returned to the GUI for display to the developer.