SFDebugger Manual

Introduction

The SmartFrog debugger is a tool designed to allow debugging of SmartFrog component descriptions at various lifecycle methods at runtime. It takes SmartFrog description file as input and analyzes it to find a set of predefined breakpoints. The breakpoints are defined as the nodes of the component hierarchy. It provides an interface to set the breakpoints and the life cycle methods at which the component should be analyzed. Also, a mechanism is provided to modify the description attributes at run time by adding, replacing and removing to the component.

Design of SmartFrog Debugger

This section describes how the SmartFrog debugger is designed to debug the component descriptions. The following steps are involved in the debugging process:

1. Find the static set of breakpoints for the given description.
2. Provide an interface to set the breakpoints and also the life cycle methods for each of the selected breakpoints.
3. Provide an interface to deploy the application.
4. Stop the application at the specified breakpoints.
5. Provide an interface to analyze the component with its current attributes and their values. Also allow to modify the description attributes at this point.
6. The program must resume after the modification of the description attributes.
7. The whole process has to be repeated till the user wishes to quit.

• *Finding the static set of breakpoints*

The input file is parsed using the SmartFrog parser API to generate various phases. These phases are then resolved and a component description consisting of a hierarchy of components related by parent-child relationships is constructed.

This hierarchy obtained is a tree of components of which all the leaves are prims. This component tree is then traversed to get a Vector of predefined breakpoints.

• *Setting the breakpoints and deploying the application*

An interface like a simple shell is provided where the user can see the static set of breakpoints and set any of the breakpoints and the lifecycle method at which the deployment process should stop. Also, the application can be deployed on any host.

• *Stopping and resuming the application deployment*

The deployed application should stop at the selected breakpoints and the lifecycle methods. There are three places at which the application can be stopped. One is before the deployment phase i.e. at sfDeploy(..) method. Second is after deployment and before start of the particular component i.e. at sfStart(...) and the third is after the component has started and before the termination of the component i.e. At sfTerminateWith(...) for each

of the selected node in the component tree. At any stopping point, the user can see the existing atrributes of the component node and modify them.
After the modification of the attributes at a component node and any of the lifecycle methods, the deployment process should be resumed. If no further breakpoints are set, it must execute the application till the end.

Implementation of SmartFrog Debugger

This section discusses the implementation details of SmartFrog Debugger. It consists of the following classes and interfaces to provide the required functionality.

• *Breakpoints Class*

This class has methods that are used to find the set of static breakpoints. It implements the CDVisitor interface. The breakpoints are maintained as a Vector.

The following methods are part of this class:

➢getBreakpoints(String url): it returns the Vector of breakpoints for a given component description. It traverses the component hierarchy and visits every node and invokes actOn() method which takes the required action on each node.

➢actOn(): this method is part of CDvisitor interface which is implemented by Breakpoints class. Its function is to add the component name to the breakpoints Vector if it qualifies as a breakpoint.

➢getCompDesc(): it returns the Component Description for the given url.

• *Debugger Class*

This class is the main class to launch the debugger. It provides the debugger shell acting as the user interface for the debugger. It implements the Command interface.

The debugger shell provides following set of commands:

- **debug:** used to specify the description file to be debugged. It takes as input the description file, analyzes it to display the set of predefined breakpoints.

- **Break:** used to set a particular break point from the available list. It has the following options to specify the life cycle method at which to apply the selected breakpoint.

    -d: to stop before deployment of the component

    -s: to stop before starting the component

    -t: to stop before terminating the component.

- **Help:** displays the command summary.
- **Run:** used to deploy the given description. It takes the hostname as parameter to specify the hostname of the node on which to deploy the description.
- **Exit:** used to exit from the debugger.

The user inputs the commands on the debugger shell prompt and the shell executes them.

Debugger class has the following methods:

➢execute(): this method provides the main loop of the debugger shell.

➢parseCommand(): this method parses the command given by the user. If the command is valid it stores it in a string else it shows an appropriate error message.

➢processCommand(): this method processes the parsed command and creates an instance of required classes and call the appropriate methods in those classes.

➢printWelcomeMessage(): this method prints the welcome message at the start of the shell.

➢printPrompt(): this method prints the command prompt for the debugger shell.

➢printHelp(): this method prints the summary of the commands that can be used in debugger shell.

• *Deployer Class*

This class provides the functionality to enable the deployment of the given description through debugger interface. Also, it provides methods to set all the breakpoints at the appropriate life cycle methods. To set the breakpoint at a specific component node, it adds certain Boolean attributes to its context before starting the deployment process. It implements the CDVisitor interface.

Deployer class has the following methods:

➢deployCompDesc(HashMap map, String hostname): it deploys the given component description on a specified host. The component is deployed in the rootProcess compound of that host.

➢putHook(ComponentDescription comp, Object metName): it adds the necessary Boolean attributes to the component context corresponding to the selected lifecycle method. It is a helper method used in setting a breakpoint.

➢actOn(): this method is part of CDvisitor interface which is implemented by Breakpoints class. Its function is to invoke putHook(...) method for the component description for which breakpoint is set.

• *SimpleShell Class*

This creates an inner shell which runs in the SmartFrog console. It provides a simple interface to examine and modify the context of the component description which has stopped in a particular lifecycle method. It supports the following commands.

- **print:** to print the component attributes.
- **add:** to add an attribute to the component context. It takes following parameters:
  - ➔ Attribute name to add
  - ➔ Value to be assigned
- **replace** – to replace an attribute value. It takes following parameters:
  - ➔ Attribute name to replace
  - ➔ Value to be assigned
- **remove** – to remove an attribute from the component context.

SimpleShell class has the following methods:

➢parseCommand(): this does the task of parsing the commands given by the user. If the command is valid it stores that in a string else prints an appropriate message.

➢processAttributes(): this method gets the command from the user, parses it and takes appropraite actions.

➢printAttributes(): this method prints the component attributes.

➢addAttribute(Object atr, Object val): this method adds the attribute 'atr' with the corresponding value 'val' to component.

➢replaceAtrribute(Object atr, Object val): this method replaces the value of the given attribute 'atr' with the value 'val'.

➢removeAttribute(Object atr) – this method removes the given attribute 'atr' from the component description.

➢help(): this method prints the summary of the commands that can be used in shell.

• *SFTrace Class*

This class creates a TraceDebugger component which when deployed with the SmartFrog Daemon adds debug hooks to the deployed components for all lifecycle methods. The hook action takes place based on the attributes added to the component context at the time of setting the breakpoints. The hook action creates the SimpleShell for analyzing and modifying the component description at runtime.

Using SmartFrog Debugger

1. Build Debugger by running ant in core/extras/sfDebugger.
2. Step 2 creates debugger jar file in directory core/extras/sfDebugger/dist/lib.
3. Add TraceDebugger component to default.sf to deploy with Daemon.
4. Build the SmartFrog framework by running ant dist in core/smartfrog.
5. Copy the debugger jar file from core/extras/sfDebugger/dist/lib to core/smartfrog/dist/lib or update class path for the SmartFrog daemon so that debugger classes are available at run time.
6. Start the SmartFrog Daemon.
7. Start the Debugger by executing the class org.smartfrog.services.sfDebugger.Debugger.