1. **Adaptive Instrumentation Process**
   1. **Design**
      1. **Inputs**

**Figure1** shows an overview of the inputs and the output of our Instrumentation Process. It receives three Inputs. First of all, the source code of the system which we want to monitor. secondly, The Correspondence Model which contains correspondences between the services and their SEFFs as well as correspondences between the SEFF elements and the corresponding JaMoPP statements. Finally, the Instrumentation Model which contains the instrumentation points that we should use to instrument the source code.

* + 1. **Output**

The output of the Instrumentation process is an instrumented version of the source code of the system. This instrumentation is based on the monitoring probes defined in (section Probes in the chapter approach). Moreover, there are two ways to instrument the source code. The first alternative consists of using the Aspect Oriented Programming which separate the instrumentation code from the source code of the system. The second alternative consists of mixing the instrumentation code with the source code of the system. Furthermore, we showed in (section Kieker in the chapter foundation) that the first alternative can not be used in our approach because of the fine-grained monitoring probes that we defined in our approach (section monitoring probes in the chapter approach). Therefore, we decided to use the second alternative in our instrumentation approach.

**Alternative of the instrumentation process outputs**

As mentioned above, in order to do the source code instrumentation, we decided to mix between the source code and the instrumentation logic. Here, we distinguish also between two alternatives for instrumenting the source code of the system. The first alternative consists of instrumenting the original source code of the system. The second alternative consists of making a copy of the original source code of the system and instrument it. We decided to use the second alternative in our instrumentation process.

**Instrumentation of a copy of the source code**

The reason why we decided to use the second alternative to instrument the source code was due to the readability of the original source code. Moreover, if we instrumented the original source code of the system, we will reduce its readability and make the maintenance more difficult in the feature for programmers. **Figure 2** shows an instrumented version of the source code in **(Figure in foundation)**. As we can see, the readability of the source code has become difficult and the maintenance will be also hard to do. Therefore, we decided the instrument a copy the original system in order to receive the monitoring information.

Instrumentation Process

The Source Code of the system

Instrumentation Model

Correspondence Model

Instrumented Copy of the Source Code

Figure 1:The inputs and the output of the instrumentation process, the output is an instrumented copy of the original source code of the system



Figure 2: Instrumented version of the source code in figure ...

* 1. **Architecture**

**Figure 3** describes the architecture of our Instrumentation approach. The component diagram shows the dependencies between the components that we used to implement our approach.

**Two Main Concerns**

We distinguish between two main concerns, which are executed in diverse contexts. The first concern is related to the instrumentation of the source code. The second concern consists of the logic used to extract the services parameters and to log the monitoring information.

**The Source Code Instrumentation Concern**

In order to instrument the source code, we provided an instrumentation process which is implemented in the component *Source Code Instrumentation*. This component depends on the component *Probes Provider*, which provides the fine-grained probes (section Probes in approach chapter) we defined for our instrumentation approach. The component Source Code Instrumentation uses these probes in combination with information from the Correspondence Model to accomplish the instrumentation of the source code. we provide more details on the instrumentation process in section (call sequence of the instrumentation pr.)

**The Monitoring Source Code Concern**

This concern is related to the source code that has to be executed to log the monitoring information. it can be divided into two tasks. The first one consists of the extraction of the service parameters. This task is implemented by the components *Service Parameters Extractor* and Service *Parameters Factory*. The first component is responsible for extracting the service parameters values and their names and pass them to the second component. The second component receives the services parameters and put them in a defined format, currently it produces a JSON file, which contains the service parameters names as the keys that their values as the values of the keys. The second task consists of the logic used to collect and log the monitoring information, which is implemented in the component *Monitoring*. The component Source Code of the System represents the source code that we want to instrument, it has dependencies to the components *Monitoring* and Service *Parameters Factory*.

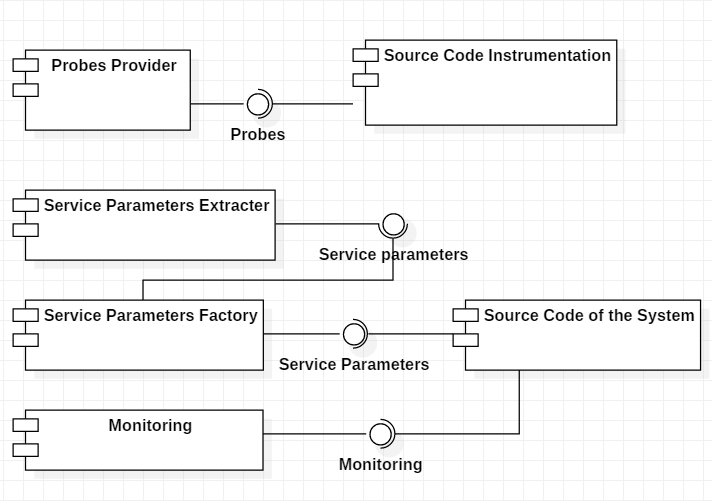
****

Figure 3: The Component Diagrams shows the dependencies between the components we used to implement our approach

* 1. **Call Sequence of the Instrumentation Process**

Figure 4 depicts the steps used in the Instrumentation Process to instrument the source code. we will describe each step in the following.

**Step 1: Execute the Instrumentation Process**

This step consists of starting the instrumentation process. Developers can execute this process at any time. Once the process is executed, it takes that have been generated from the Probes Generation Probes (**section** or chapter Probes generation process) and accomplish the instrumentation of the source code.

**Step 2: Copying the Source Code of the System**

Here, we create a copy of the source code, which we will instrument. As we showed in (section output), we decided to instrument a copy of the source but not the original source code. Moreover, since we used Eclipse for our development purpose, we created a functionality that clones the original project of the system with its dependencies and properties. Therefore, the instrumentation will find place in cloned project.

**Step 4: Parse the Copied Source Code via JaMoPP**

In this step, we create the Java model of the source code that we want to instrument, which is the copied source code. For parsing the Java source code, we used JaMoPP (section JaMoPP). Moreover, the parsing of the source code will give us the possibility to manipulate it, like referencing the corresponding statements of a probe or the insertion of the instrumentation code in a defined position in the source code.

**Step 5: Returning the Monitoring Probes**

In this step, the Instrumentation Process calls the component *Probes Provider* to receive the probes that we want for the instrumentation.

**Step 6: Finding the Probes Statements**

As we mentioned before, we will instrument a copy of the source code, which means we will need find the statements in the copied source code that correspond to our probes. However, the inputs that we have for the Instrumentation Process at this stage are the probes and the corresponding statements in the original source code. That means, in order to instrument the copied source code, we will need to search the corresponding statements of the probes in the copied source code. To do this, we compared the statements of the probes in the original source code with the statements of the copied source code.

In order to optimize the searching for corresponding statements of the probes in the copied source code, we used three parameters to identify equal statements. The first parameter is the service in which statement is written. The second parameter is the class in which the statement find place. The third parameter is the location of the statement in the source code. The combination of these parameters is unique for every statement in the source code. Therefore, it can be used to search for the probes statements in the copied source code based on probes statements in original source code.

**Step 7: Instrument the Source Code**

In this step, the copied source code will be instrumented based on the probes and their corresponding statements that has been mapped in the pervious step. Moreover, the probes are injected in the source code based on their types (section Probe). Here, we distinguish between two kind of instrumentation, namely coarse-grained instrumentation and fine-grained instrumentation. Fine-grained instrumentation is based on the probes that are represented by the SEFF elements that we defined in (Probes). Coarse-grained instrumentation consists of instrumenting the whole service without taking into account the its fine-grained probes. In this step, we execute both fine-grained and coarse-grained instrumentation.

**Step 8: Coarse-grained Instrumentation**

This step consists of the coarse-grained instrumentation of all services in the source code that have been not instrumented in the previous step. This step is required because of the adaptive instrumentation approach. As we mentioned before, in adaptive instrumentation, we collect probes only for the changed parts of the source code in the current iteration. That means, services that have been not changed in the current iteration will not be related to any probes. Thus, they will not be instrumented in step 7. However, even tough they have been not changed in the current iteration, if they are called in the changed parts of the source code, we will need their monitoring information. Therefore, we finalized our instrumentation process by the coarse-grained instrumentation of the rest of the services of the system.

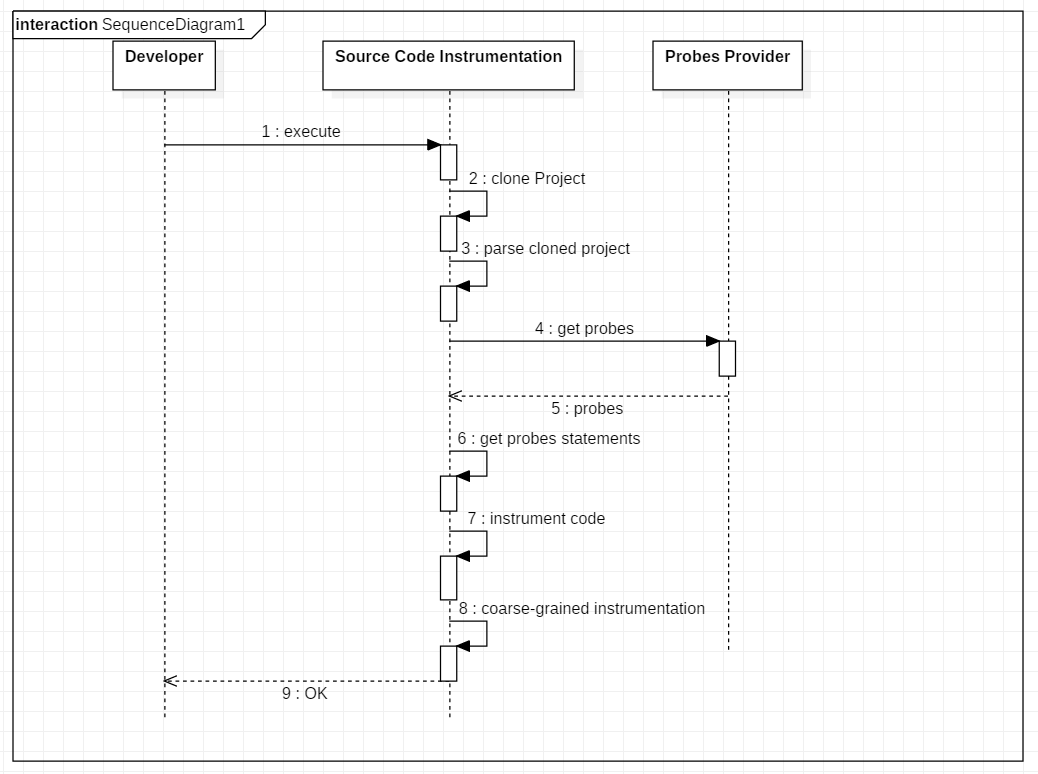
****

Figure 4: Sequence Diagram that illustrates the activities of the Instrumentation Process

* 1. **Service Parameters Extraction**

Manar and Koziolek showed that Performance Model parameters can strongly depend on the parameters of the services. Therefore, we decided to enrich our monitoring information by extracting the service parameters and their values.

**Figure 5** shows our approach for extracting the service parameters. We limited our approach to five parameter types, namely the type Map, Collection, Array, Primitive and Data Types. Data Types are the types that are defined by the user and are given as parameters to a service. For example, a DataType can be Java Bean Class named FileType, which contains the attributes file name of type string and file content of type Array of byte.

For service parameters extraction, we return for each parameter type only the property that has impact on the performance model. For collections and maps we extract their size. The same thing for arrays, we return their length. As for primitive types, we return the value of the parameter. Furthermore, Data types are handled via Reflection. In Data types we use reflection to search for attributes of the types Map, Collection, Array or Primitive in order to extract their defined properties.

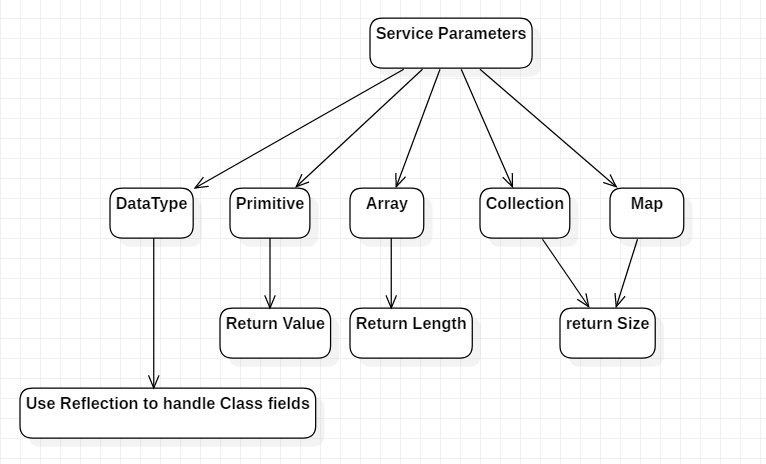
****

Figure 5: Service parameters types and how they are extracted