University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Experimental SOFA Implementation

Research Report

Petr Hanč, Jan Rovner, Jan Valdman

Technical Report No. DCSE/TR-2001-02
November, 2001

Distribution: public

Technical Report No. DCSE/TR-2001-02
November 2001

Experimental SOFA Implementation

Petr Hanč, Jan Rovner, Jan Valdman

Abstract

This technical report report describes current stage of an experimental SOFA implementation created at the Department of Computer Scienence and Engineering, University of West Bohemia. The information hereafter is intended mainly for new SOFA team members and students of master degree that participate on the project. However, it could be interesting also for external spectators that are interested in the SOFA framework or in component architectures in general.

This work was supported by the Grant Agency of the Czech Republic (GACR), project No. 201/99/0244 “Developing software components for distributed environment”

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

1. Introduction to SOFA

1.1 Requirements for SOFA

The only requirement for running SOFA application is a computer with operating system capable of running Java Virtual Machine version 1.2 or higher. Current SOFA implementation is written in Java 2 language.
SOFA was tested to run in Windows 98 and in Linux Red Hat 6.3, both with JVM 1.2.

1.2 Running SOFANode in Windows easily

To run SOFA Node in Windows, first you need to set the CLASSPATH environment variable to contain the parent directory of the sofa directory. However, most Java programs require this step and over time the CLASSPATH variable becomes rather long. Here is a better solution: create a directory named *classes*, and set CLASSPATH to contain this directory and a dot (which means current directory).
SET CLASSPATH=C:\classes;. (Both semicolon and dot should be there!)

Now copy the sofa directory into the classes directory. Next time, if any other Java program would want the classpath to contain its directory, it is not necessary to modify classpath, just copy the program into the *classes* directory.
Second thing is that the SOFAnode needs to be told how to access template repository. This can be done either via local file system or via Java RMI. Currently there is no configuration option, the information is hard-wired into the code.

1.3 SOFA Backgrounds

The SOFA project was started at the Department of Software Engineering at Charles University in Prague. The goal of the project was to design a software environment to support software provider - user (consumer) relation. The abbreviation SOFA means Software Appliances.
The purpose of described SOFA design and implementation is to work out some parts of SOFA with respect to its intended implementation in Java, to test the viability of SOFA architecture and its suitability for practical use and to make basic implementation of SOFA node for testing purposes.
Our work is supported by the Grant Agency of the Czech Republic, project no. 201/99/0244 „Developing software components for distributed environment“.

1.4 Intended SOFA Contributions

The key issues addressed by SOFA are:

- DCUP (Dynamic Component Update) – a mechanism for changing a component without having to stop the whole application
- component trading
- component licensing and billing
- component versioning
- security support

From these issues, we implemented only the DCUP ability. Other issues may be subjects of master theses at our department.

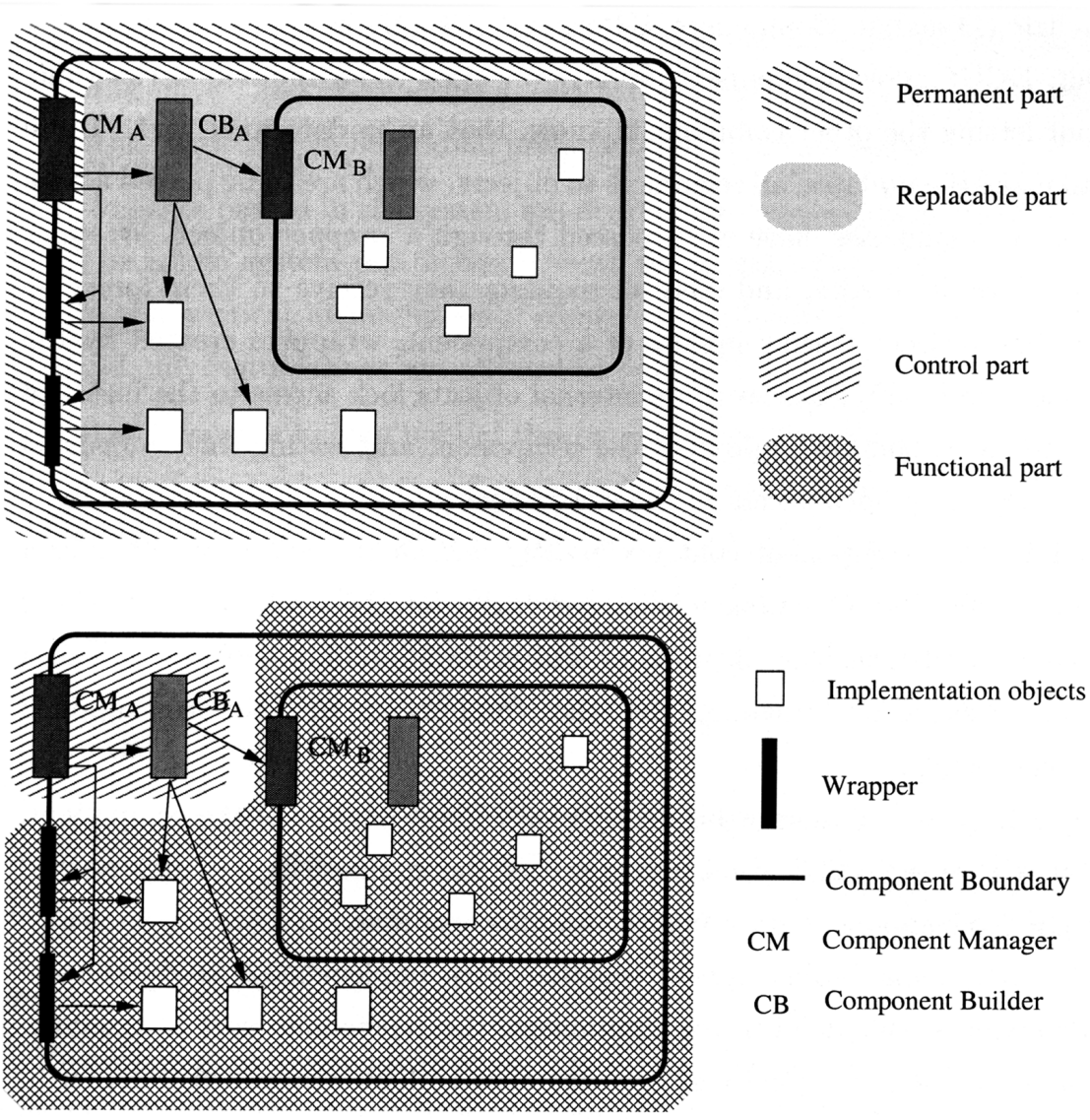
1.5 SOFA Concepts

A SOFA application is a hierarchy of mutually interconnected software components. An application can be created just by a composition of bought components perhaps with great

reuse of already owned components. The DCUP (Dynamic Updating) feature allows to upgrade or exchange components at application runtime.

SOFA components are described in Component Description Language. CDL is a high level structured language capable to describe interfaces, frames, architecture, bindings and protocols. CDL is used both at compile and run time.

A SOFA component is a black box of specified type with precisely defined *interfaces*. Each component is an instance of some component type. At runtime a component consists of a permanent and a replaceable part. The permanent part creates a ``border'' of the component. It contains the Component Manager(CM) and wrapper objects that intercept calls on component's interfaces. The replaceable part is formed by the Component Builder (CB), internal implementation objects and subcomponents. In other words, CM and CB form the control part of a component while the rest is a functional part.



An important feature of SOFA architecture is support for electronic market of components. That is why SOFA introduces SOFAnet and SOFAnode concepts that reflect various roles that companies play on a market---they are producers, retailers, end-users or a combination of these.

SOFAnet is a homogeneous network of SOFA nodes. In contrast to other component architectures, SOFA covers not only the computer oriented aspects (objects, call schemes, operating systems) but it also faces real-word problems like manufacturing, trading, distribution and upgrading of components. From this point of view SOFA is a complex system and not only a component-based middleware.

1.6 SOFA General Overview - SOFAnode

- SOFA node, which is a basic building block of the SOFA network, is functionally divided into following parts:
- template repository (TR) – stores component binary runable images and provides them upon request
- run part (RP) - serves as a "SOFA operating system", i.e. it manages SOFA applications and provides services to component management objects (CM, CB – see further)
- user shell (SH) - provides user interface to SOFA. It shows a list of components that can be started as applications and a list of running applications. Allows the user to start an application, shut down a running application, update an updatable component.
- made part (MD), IN and OUT – not implemented; according to the design of SOFA these parts are used for component migration between SOFA nodes and between node and component manufacturer.

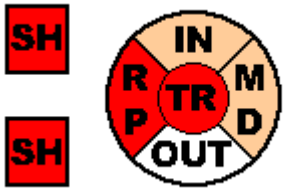


Image 1 – SOFA Node

1.7 DCUP feature

Dynamic Component Update is so far the only implemented feature of SOFA architecture. The key problems that had to be solved were:

- an update of a component must be fully transparent to the rest of the application
- transition of state from the old to the new version of a component
- references between updated component and its neighborhood need to be renewed

1.8 Component Bindings And Interface Wrappers

Input and output interfaces of components are realized by interface wrappers (IW). These allow the architecture to dynamically bind components together, to control interface's behavior and traffic.

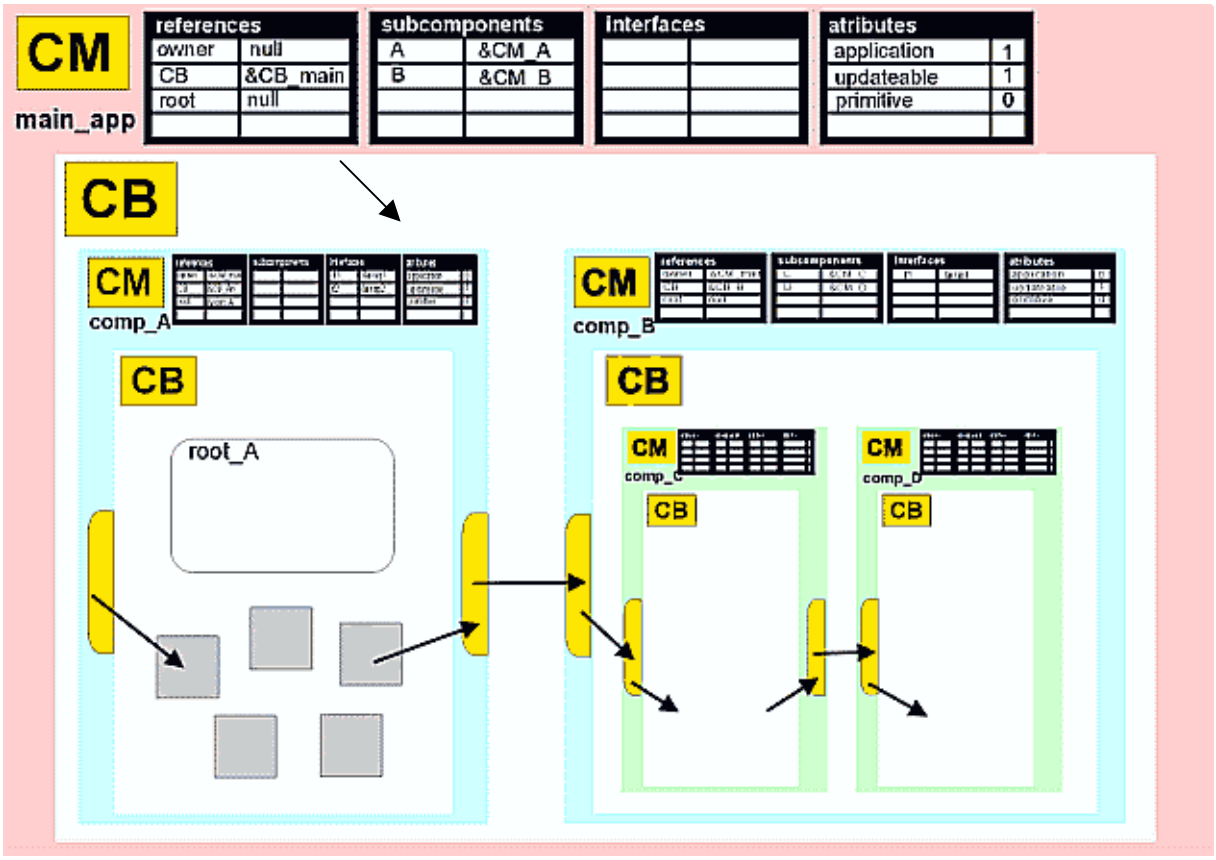


Image 2 – Component binding and wrappers

1.9 SOFA User Interface

When SOFA Test Application is started, a blue window with six buttons appears. With these buttons you can display four information windows, hide them all and exit the demo. The four information windows are:

- debug window (red) – shows debugging information, which are sent here by calling SOFASystem.fdb.debug() method
- template repository (pink) – displays the list of components, that are present in template repository
- run part (yellow) – shows the tree of components loaded in RP and additional information on their CMs
- user shell (umber) – shows running applications and components in TR. Also allows user to load, start, stop, update etc. selected components. In fact, this is the main control window for SOFA.

2. SOFA Design

2.1 SOFA Inner Interfaces

All basic SOFA entities like TR, root object, RP etc. communicate with each other through interfaces. The names of the interfaces reflect the entities, which communicate through them. For example ICB2CM interface contains methods for component builder to call component manager.

There are also several general interfaces, that do not connect only two entities, but are available to all of them. These are for example ICManager, ICBuilder and IRootObject.

Interfaces are stored in the SOFA/interfaces directory. The description of key methods contained in interfaces is in chapters that describe individual entities.

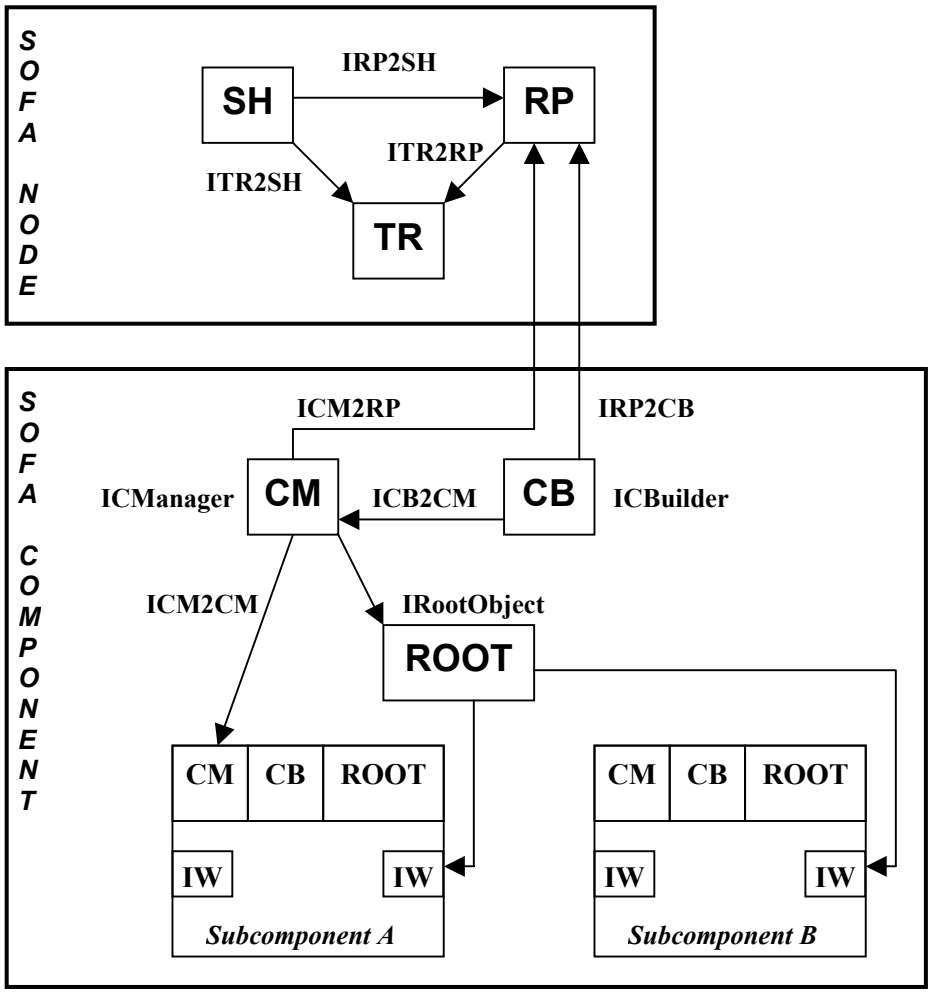


Image 3 – SOFA interface overview

2.2 Introduction

Image 3 shows example of sofa component. The scheme is not general, but it clearly shows which interfaces are used for calling which objects. Objects are in boxes, interfaces are arrows with description. Arrows are oriented in direction of calling. There are also three general interfaces, which are without arrows, because they can be called by any object.

This example component is a component with two subcomponents. Inner structure of both subcomponents is similar to the structure of the main component, so for simplicity it is not shown. Note also that the main component has no own input and

output interfaces. Both subcomponents do not require any interface (and so there is no arrow going from their input interface wrappers). Subcomponents provide some output interfaces. Their functionality is used by root object, which also does all the functionality of the main component.

2.3 Template Repository

Template repository is a place, where component images are stored and from where they are being taken and instantiated. Component binaries are stored in a hierarchy of directories, where the top level directory name means manufacturer's name and the second level directory means component's name. All classes of a component are stored packed in one .jar file. The name of this file indicates version of the component.

Template repository is implemented as sofa.repository.TemplateRepository class. Key methods of this class are:

- getComponentList() – returns list of descriptors of components which are in TR
- getStreamWithBinaryImageOfComponent() – returns a stream that contains binary image of component, which is specified by its component ID

Auxiliary classes and their roles are:

- ComponentDescriptor – this class contains information about component in TR, namely component's identifier (SOFAComponentID), producer, name and version
- ComponentDescriptorList – implements list of ComponentDescriptors
- TRList – its method getComponentList() goes through TR and gets information (producer, name and version) about all components

2.4 Runpart

Runpart contains methods for loading selected application and for creating instances of components. It also maintains a list of component managers of loaded components. The main class of runpart is RunPart, key methods are:

- registerComponentManager(), unRegisterCM() – self-explaining
- loadApplication() – loads an application specified by ComponentDescriptor argument
- makeComponent() – is similar to loadApplication(), but uses different arguments and initializes newly created instance in a bit different way

When a new component instance is being created, first the root component manager class is created, and then its init() method is called. For building up the rest of the component is responsible its component builder.
For loading applications and creating components the runpart utilises the technology of class loaders, which is covered in next subchapter.

2.5 Class Loaders

The technology of class loaders is implemented in java itself. Generally, whenever a new instance of a class is created, it is created by a class loader. Class loader is a class that loads binary image of a class into memory and returns reference to it. If a class is loaded without using an explicit class loader, it is loaded by default system class loader.
If a class is loaded with an explicit class loader, then when this class tries to create an instance of some other class, the JVM primarily tries to create this instance with the same explicit class loader. This feature is very important, because it allows to set up

specific place (direcotry) for every class, where its subclasses are looked up and loaded from.
Abstract class SOFAAbstractClassLoader is based on java's URLClassLoader class. For loading classes from files, we created SOFAFileClassLoader class.

2.6 User Shell

Shell implements user interface to SOFA node. Its appearance is described in chapter 1.8. In current state of implementation, shell allows only loading, starting and pausing of components.
These operations are implemented in components themselves, shell only does some preparation and calls them up. The following methods of the ShellFrame class are most important:

- loadBtn_actionPerformed() – loads an application
- startBtn_actionPerformed() – starts loaded application
- pauseBtn_actionPerformed() – resumes paused application

3. Components and Applications

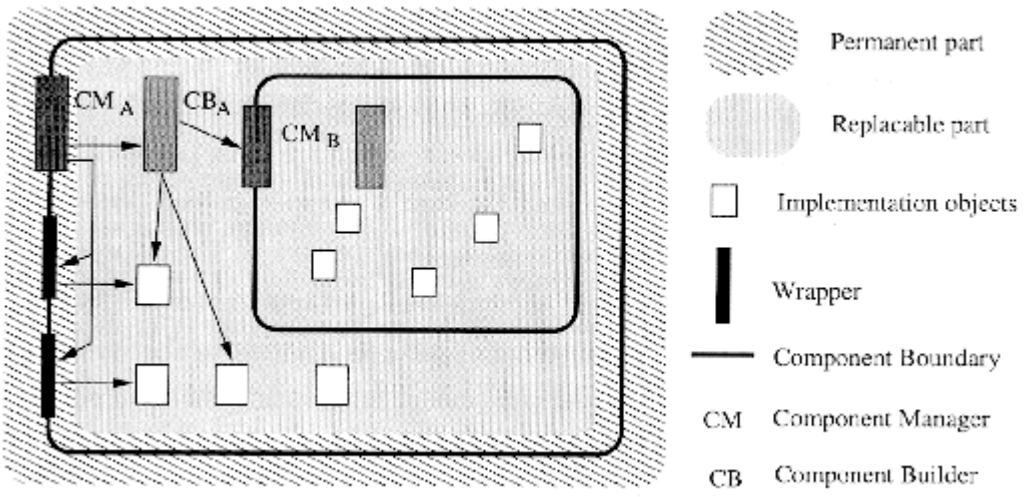
3.1 Introduction

As it was described before, a component can be of three types of behavior: library-like components without any running threads, single-thread components (similar to classic programs) and multi-thread components.

From another point of view, components can be divided into two another groups: plain components that utilize only the Root Object and compound components that instantiate subcomponents.

Finally, from yet another point of view, components can be divided into plain components (without 'main()' function) and application-like components. From a structural point of view, there is no difference between plain components and applications., Run Part treats them in the same way. The only difference is only in User Shell behavior, i.e. application can be directly loaded and started by the user in contrary of 'normal' components.

All components communicate with their neighborhood using two sets of interfaces. One set contains so-called *requires* interfaces. These are calls that the component requires from outside. The second set contains so-called *provides* interfaces. These are calls that the component offers to be called from outside. All interfaces are wrapped in interface wrappers (IW, see chapter 3.5).



3.2 Component Manager

All important functionality of component manager is implemented in SOFACMTemplate class. At current state of implementation, component managers are derived from this class; in future they will be automatically generated from CDL file.

The only method that component manager has to override is createComponentBuilderInstance(), which creates an instance of component-specific component builder.

The key features of SOFACMTemplate are:

- fCBuilder – contains reference to CB
- fInterfaceList – contains list of interface wrappers of current component (lists of interface wrappers of inner components are stored in their component managers)
- fComponentManagerList – contains list of component managers of inner components

- fApplication, pauseFlag, endFlag, updateFlag, externalizeFlag – state flags of the component
- extStream, extFile – a stream and a file for externalization, i.e. for saving internal state of component before updating (or on request)
- init() – initialization method of component manager. First it creates an instance of component builder and calls its buildComponent() method. Finally it calls CB's method bind() to bind inner components together.
- registerInterface() – adds new interface to the list of interfaces. The new interface is described by unique SOFAIID (interface identifier) and its interface wrapper.
- lookupInterface() – returns interface wrapper of interface specified by SOFAIID
- registerComponentManager() – adds new CM to the list of CMs of subcomponents
- queryInterface() – returns reference to an object that implements requested interface specified by SOFAIID
- start() – changes the state of the component to "running". First it starts root object, then it starts all subcomponent managers.
- pause() – changes the state of the component to "paused" and delegates this request to subCMs
- resume() – changes the state of the component from "paused" to "running"; again it delegates the request to subCMs
- externalize_begin() – the first phase of externalization. It prepares component for externalization. Component's state changes to "externalizing", all interface wrappers are switched off and all subcomponents are informed about upcoming externalization.
- externalize_commit() – the second phase of externalization, externalization itself. It calls root object's externalize_commit() method, which should be overriden and in which the root object should write its variables into submitted stream in format variable=value. Again, this method recursively calls all subcomponents.
- externalize_finalize() – the last phase of externalization. Informs all subcomponents about finished externalization, switches on all interface wrappers and changes component's state to "running".
- externalize() – non-recursive method, that launches the three phases of externalization.
- destroy_begin() – the first phase of destroying a component. All interface wrappers are switched off and all subcomponents are informed about upcoming destruction.
- destroy_commit() – the second phase of component's destruction. It calls root object's destroy_commit() method, which should be overriden and in which the root object should do all actions to be done before destruction. Recursively, destroy_commit() is delegated to all subcomponents.
- destroy_finalize() – the last phase of destruction. First it delegates the request to subCMs, then it destroys component builder, interface wrappers and auxiliary data structures like various lists, and then it unregisters itself from runpart. Finally it calls system garbage collector.
- destroy() – non-recursive method, that launches the three phases of destruction of a component.
- destroy_kill() – alternative method to destroy_finalize() for the case that some component is stuck and does not respond to destruction anouncement.
- update() – updates component with the component specified by given component descriptor. First it does externalization, than it destroys all subcomponents and

inner parts of the component. Then component builder of new component is created, which builds that component. Also the externalized state of the component is restored and component's state is changed to "running".

3.3 Component Builder

All important functionality of component builder is implemented in SOFACBTemplate class. At current state of implementation, component builders are derived from this class; in future they will be automatically generated from CDL file.

The only methods that component builder has to override are instantiateSubComponents(), which creates instances of inner components, createRootObject(), which creates a component-specific root object, and createInterfaceWrappers(), which creates component-specific interface wrappers and registers them in CM.

The key features of SOFACBTemplate are:

- fCB2CM – contains reference to CM that owns this CB. The reference gives access to CM's methods.
- buildComponent() – creates interface wrappers of component and creates instances of inner components. Then it creates root object and calls its init() method.
- bind() – Performs all binding in the component. Also calls root's bindInterfaces();
- setInterfaceImplementation(), getInterfaceImplementation() – methods for binding inner components together

3.4 Root Object

Root object is the main object of each component. It contains a thread, where all component-level activities are done. Template class for root object is SOFARootTemplate. Here are its key methods and variables:

- start() – creates new main thread and starts it
- pause(), resume(), externalize() – these methods receive requests for corresponding operation. Some operations have two phases, for example externalization itself is done in the do_externalize() method.
- init() – initialization of root object
- init(FileInputStream fi) – initialization of root object, during which the internal state is read from externalized data file. Reading of the data is component-specific and needs to be programmed by the creator of the component.
- bindInterfaces() – binds interfaces to inner objects and components
- externalize_begin(), getExternalizeAck(), externalize_commit(), externalize_finalize() – methods used during externalization. Externalize_commit() should be overriden so as it writes component's variables into submitted stream in format variable=value.
- destroy_begin(), getDestroyAck(), destroy_commit() – methods used during component's destruction. Destroy_commit() may contain some cleanup code.
- run() – this method contains inner program of the component (and thus the programmer should override it). As the root object is a thread, this method is the place where the component lives. Some components have no internal life, because they act as function libraries, so they do not need run() method at all (because SOFARootTemplate already implements run() as an empty method).

3.5 Interface Wrappers

Interface wrappers are special classes, that encapsulate provide and require interfaces of the component and allow the system to enable/disable communication on the

interface. An interface wrapper can contain request queue. Every component‘s interface has its specific interface wrapper.
Basic class for interface wrappers is SOFAIWTemplate with the following key elements:

- isReady, isOnFlag – flags that indicate, whether the underlaying interface is ready to use and whether the communication on the interface is enabled.
- implObj – contains reference to the object, which implements the interface. The problem is that interface entities in java cannot be overtyped, so using objects is the way how to get around this.
- getIID() – returns unique interface identifier. This method needs to be laid over to return interface-specific information.
- setInterfaceImplementation() – sets the implObj variable to specific object
- enterInterfaceFunctionCall() – intended for tracing and administrative purposes
- off(), on() – disable/enable traffic on interface

4. Component's Lifecycle (Scenarios)

This chapter describes scenarios of operations that can be done upon component. As it is natural, more operation are designed than implemented.

4.1 Loading a Component

Loading starts when users clicks the „load" button in user shell, or when loading is requested by higher component. In the first case, the RunPart.loadApplication() method is called, in the second case it is RunPart.makeComponent(). Both loadings are done in the same way, except that in the second case the component manager of newly loaded component is registered at parent's component manager. Here is the scenario of what happens when a component is being loaded:

1. a classloader is created
2. component manager (CM) is created (it is a class, which is loaded using previously created classloader)
3. init() method of the CM is called
4. CM creates component builder (CB) of the component
5. CM calls CB's buildComponent() method to build the component
6. CB first creates interface wrappers (IWs), then it creates instances of inner components (which are done by makeComponent() method, so this scenario starts for them from the beginning). Of course, in case of primitive component, this step does nothing, as a primitive component has no inner components.
7. finally CB creates root object of the component and calls its init() method
8. during initialisation, the root object instantiates and binds together its inner objects
9. the execution point exits CB's buildComponent(). CM calls CB's bind() method and it delegates this to root object
10. root object binds internal objects to IWs, it also binds components together
11. now the CM's initialisation ends and the component is in „loaded" state

4.2 Starting of a Loaded Component

This scenario is quite simple.

1. CM's start() method is called
2. CM calls root's start() method
3. root creates its internal threads (if there are any) and starts them
4. CM calls start() method of all sub-component managers
5. component's state changes to „running"

4.3 Pausing and Resuming

This scenario is also quite simple.

1. CM's pause() or resume() is called
2. CM calls root's pause() or resume()
3. CM delegates request to all subCMs
4. component's status is changed to „paused" or „running"

4.4 Externalization

1. CM's externalize() is called
2. it runs the first phase of externalization – externalize_begin(), where component's state is changed "externalizing", all interface wrappers are switched off and all subcomponents are informed about upcoming externalization.

3. CM waits in a loop for all components to acknowledge they are ready for externalization
4. externalize_commit() is called. This is the second phase of externalization, externalization itself. It calls root object's externalize_commit() method, which should be overriden and in which the root object should write its variables into submitted stream in format variable=value. Again, this method recursively calls all subcomponents.
5. externalize_finalize() is called, which is the last phase of externalization. Informs all subcomponents about finished externalization, switches on all interface wrappers and changes component's state to "running".
6. externalize() finishes

4.5 Destruction

1. CM's destroy() is called
2. it runs the first phase of destruction – destroy_begin(), where all interface wrappers are switched off and all subcomponents are informed about upcoming destruction.
3. CM waits in a loop for all components to acknowledge they are ready for destruction
4. destroy_commit() is called. This is the second phase of component's destruction. It calls root object's destroy_commit() method, which should be overriden and in which the root object should do all actions to be done before destruction. Recursively, destroy_commit() is delegated to all subcomponents.
5. destroy_finalize() is called, which is the last phase of destruction. First it delegates the request to subCMs, then it destroys component builder, interface wrappers and auxiliary data structures like various lists, and then it unregisters itself from runpart. Finally it calls system garbage collector.
6. destroy() finishes

5. Using SOFA – A Sample Application

SOFA application classes can be divided into two categories. Classes in one category implement application's functionality, classes in the other category integrate and encapsulate component for use in SOFA framework.

Classes related to SOFA architecture, like CM, CB and root object can be derived from their templates, which are in sofa.abstr package. They implement most SOFA - related things, only some methods need to be overridden. In some future version of SOFA, these classes will be generated from CDL file by CDL compiler.

When creating an application, first its design should be done. It is up to programmer to design modularity of the application and functionality of individual building blocks (components). So the application's structure should be designed. Also components should be specified, with respect to their reusability, functionality, etc.

5.1 SOFA Application Example – Calculator

SOFA was designed with respect of simple usage. Under normal situation, a SOFA programmer declares components via CDL and a CDL compiler creates a skeleton of the code in given programming language. The programmer start writing Root.run() method in the same way a C programmer starts writing the main() function. As far, the SOFA infrastructure is hidden.

On the other hand, every SOFA component should respond to some events raised by Run Part (i.e. externalization, upgrade) in a 'nice' way. This is done by overriding few methods of the Root class. The application logic of such responses (thread synchronization etc.) is left on the programmer. The basic idea of SOFA is:

Take care about all resources that you created by yourself and override corresponding methods to respond to control calls on CMs.

5.1.1 Introduction

One example can say more than pages of description. For the reason this chapter describes a sample SOFA application. The example is very simple and, in fact, it does nothing useful. However, because it is simple, SOFA things can be seen there very clearly.

The example should work as a calculator. It has no user input. It starts at zero and increments the value by five every second.

The structure of the example consists of three components:

- calculator cpu – CalcCPU, library-type component with basic mathematical functions
- display – responsible for showing numbers
- main component – CalcDemo, which contains the cpu and the display

From interface point of view, both subcomponents have only "provides" interfaces.

5.1.2 CalcCPU Component

- The basic class of this component is CalcCPU, which contains all the functionality of the component. It contains four basic mathematical functions. Note that CalcCPU is ordinary java class and that there are no signs of SOFA's presence.
- ICalcCPU contains interface for the CalcCPU class. This interface is in fact the interface that the component provides.
- CalcCPUIW contains interface wrapper of ICalcCPU. It extends SOFAIWTemplate, some component-specific things are added. CalcCPUIW implements ICalcCPU, so these are compatible, however IW allows to do some SOFA things. IW holds its own private reference to object that itself implements ICalcCPU, this reference is set by

setInterfaceImplementation() call. Method getIID() is overriden so as it returns the right interface identifier (in this case IID_ICalcCPU). Other methods are the same as in ICalcCPU and in fact they just forward calling there, they work as wrappers.

- CalcCPUCM contains component manager. It extends SOFACMTemplate. The only overriden method is createComponentBuilderInstance(), which creates and returns a new component-specific component builder class instance (in this case it is CalcCPUCB).
- CalcCPUCB contains component builder. It extends SOFACBTemplate. Only two methods are overriden – one for creating component-specific root object, and the second for creating and registering interface wrappers. Their names are createInterfaceWrappers() and createRootObject(). Guess, which of them does which function…
- Finally, there is CalcCPURoot, which contains the root object. It extends SOFARootTemplate. Root object holds private instance of CalcCPU. The instance is created during init() method. Method bindInterfaces() is also component-specific, so it is overriden.

It may look like that a really simple component needs lot of programming. However, all SOFA-related classes contain only few lines, the rest of dirty work is already programmed in class templates. Moreover, most of SOFA-related classes will be in future generated from CDL file.

5.2 Display Component

This component is much like the CalcCPU component. It has no "requires" interfaces, and it provides just one interface. It has also only one inner implementation object. The structure of this component is the same as CalcCPU's, so it will not be described here. However, something is new in this component. The CalcCPU had no internal state, it behaved as functional library. The Display has display window and the display has some value. When the component is externalizing, it needs to write this value into stream, and it should be also able to read the value from stream. Before destruction, this component should release its window.

- DisplayRoot contains besides init() and bindInterfaces() also methods externalize_commit(), where saving of display value is done, destroy_commit(), which releases display window, and do_restore(), which reads display value from stream.

5.3 CalcDemo Component

This component is a bit different from previous two, because it has no inner implementation object, it contains two subcomponents and it does not require nor provide any interface.

- CalcDemoCM contains component manager. In comparison with CalcCPUCM, there is nothing new here.
- CalcDemoCB contains component builder. Besides already mentioned createRootObject() it contains method instantiateSubComponents(). This method was not in previous components, because they contained no subcomponents. This method creates instances of subcomponents by calling runpart's method makeComponent() and by submitting required information about the components.
- CalcDemoRoot contains root object. This class is a bit more complicated. Two private variables hold references to interface wrappers of subcomponents (k for CalcCPU and d for Display). In bindInterfaces() method, these variables are filled with valid references. Methods like externalize_commit() and do_restore() are overridden to provide correct behavior of the component. Other methods are specific to this

component and perform its functionality. Note, that method run() is present in this component's root object. Previous two components had no own life, so they did not contain this method. This main component has its own life (the value increments by five every second) – it is done in the run() method.

6. Conclusion

This techreport describes current stage of SOFA implementation but either the implementation or this report are subjects of sustained modification.

Besides this techreport, another important reference document describing our SOFA implementation exists. It's auto-generated standard Java HTML (javadoc) documentation, extracted from source code javadoc comments. The generated document presents all of project's packages, classes, interfaces and methods.

However, the information provided here or even the Javadoc reports can be partly out-of-date or obsolete – in doubts, please consult the source code.

6.1 Future Work

Current SOFA implementation is purely experimental and has many serious limitations. It was coded especially to verify and evaluate ideas and techniques of SOFA/DCUP. The SOFA framework should be re-implemented to make it at-least public laboratory-usable and the required re-implementation would require still lots of work.

7. References

[1] Plasil F., Balek D., Janecek R.: SOFA / DCUP Architecture for Component Trading and Dynamic Updating. In: Proceedings of ICCDS '98, Annapolis, IEEE CS, 1998.

[2] SOFA group at The Charles University, Prague.
http://nenya.ms.mff.cuni.cz/thegroup/SOFA/sofa.html

[3] SOFA group at The University of West Bohemia in Pilsen.
http://www-kiv.zcu.cz/groups/sofa

[4] SOFA Template Repository Implementation. Master thesis of Stanislav Dobry. Department of Computer Science And Engineering, University of West Bohemia, 2001

8. Appendices

8.1 Javadoc Summary

There is a Javadoc – generated summary of the source code available for this project. There is a HTML version that contains all cross-references, an index etc.

8.2 Printed Javadoc Summary

There is also a shorter printed version generated through LaTeX that provides a comprehensive listing of all packages, classes , intertfaces and exceptions. It is a part of this technical report.

.

8.3 Full Source Code

There is also an archive with full source code available on web pages of the SOFA group at UWB or it is available at e-mail request.

SOFA Implementation at UWB

Petr Hanc, Jan Rovner, jan Valdman

November 30, 2001

Contents

Chapter 1

Package sofa.interfaces

Package Contents *Page*

Contains all "SOFA" interfaces. These interfaces are the basic interfaces between various subsets of the SOFA infrastructure. Currently this package contains only interfaces between Template Repository, Run Part, User Shell and interfaces of some fundamental objects in the Run Part.

1.1 Interfaces

1.1.1 Interface **ICB2CM**

Interface between component builders and managers. Allows component builders to access data stored in component managers.

Declaration

public interface ICB2CM

Methods

- *getCMList*
  `public SOFAComponentManagerList` **getCMList**`( )`
  - **Usage**
    - Utility function, returns list of all subcomponent's component managers

- *getRPRef*
  `public IRP2CB` **getRPRef**`( )`
  - **Usage**
    - Utility function, returns reference to Run Part

- *lookupInterface*
  `public IIfaceWrapper` **lookupInterface**`( sofa.common.SOFAIID` **iid** `)`
  - **Usage**
    - Utility function, returns reference to interface wrapper for interface identified by iid.

- *registerInterface*
  `public void` **registerInterface**`( sofa.common.SOFAIID` **iid,**
  `sofa.interfaces.IIfaceWrapper` **refIfaceWrapper** `)`
  - **Usage**
    - Component builder registers interface's (idenfied by iid) wrapper object to component's component manager.

1.1.2 Interface **ICBuilder**

Interface of Component Builder. Allows to bulid and destroy the replaceable part of a component and handles some binding.

Declaration

public interface ICBuilder

METHODS

- *bind*
  `public void` **bind**`( )`
  - **Usage**
    - Component manager calls this method during component startup to set-up internal bindings between components.

- *buildComponent*
  `public void` **buildComponent**`( sofa.interfaces.ICB2CM` **cm2cb**`, boolean` **createWrappers** `)`
  - **Usage**
    - Builds the internal part of component. Creates interface wrappers, instantiates subcomponents and the root object.
  - **Parameters**
    - `cm2cb` - reference to parent Component Manager
    - `createWrappers` - if true then wrappers would be created. This is set true when building the component and to fasle while updating the component.

- *getInterfaceImplementation*
  `public Object` **getInterfaceImplementation**`( sofa.common.SOFACMID` **cmID**`, sofa.common.SOFAIID` **iid** `)`
  - **Usage**
    - Components requiring interfaces calls back this method during build time to get reference to interface wrapper implementing interface named iid.

- *getRootObject*
  `public IRootObject` **getRootObject**`( )`
  - **Usage**
    - Helper function, returns reference to component's root object.

- *setIntefaceImplementation*
  `public void` **setIntefaceImplementation**`( sofa.common.SOFAIID` **iid**`, java.lang.Object` **oRef** `)`
  - **Usage**
    - Components providing interfaces calls back this method during build time to set reference to interface wrapper implementing interface named iid.

1.1.3 INTERFACE ICM2CM

Part of Component Manager's functionality necessary for comunication with parent or child Component Managers.

DECLARATION

public interface ICM2CM

METHODS

- *getCMList*
  `public SOFAComponentManagerList` **getCMList**`( )`
  - **Usage**
    - Helper function, returns lists of all registered component mangers of subcomponents.

- *getInfo*
  `public String` **getInfo**`( )`
  - **Usage**
    - Helper function, returns string information about component manager.

- *getRPRef*
  `public IRP2CB` **getRPRef**`( )`
  - **Usage**
    - Helper function, returns references to Run Part instance.

- *registerComponentManager*
  `public void` **registerComponentManager**`( sofa.common.SOFACMID` **cmID**`, sofa.interfaces.ICManager` **refCM** `)`
  - **Usage**
    - Called back by subcomponent to register its component manager (indentified by cmID) to its "owner" component

1.1.4 INTERFACE ICM2RP

Purpose of its interface is to register root component manager to Run Part's global component manager table.

DECLARATION

public interface ICM2RP

METHODS

- *registerComponentManager*
  `public void` **registerComponentManager**`( sofa.common.SOFACMID` **cmID**`, sofa.interfaces.ICManager` **refCM** `)`
  - **Usage**
    - Registers component manager identified by cmID to Run Part's component manager table.

1.1.5 INTERFACE **ICManager**

Component manager's main interface (control interface of Component Manager), Allows to create, control and manage component. Most of interaction between component and the rest of SOFA system is executed here.

DECLARATION

public interface ICManager

METHODS

- *destroy_begin*
  `public void` **destroy_begin**`( )`
  - **Usage**
    - Starts phase 1 of component shutdown. Multicasts the message to all subcomponent's managers.

- *destroy_commit*
  `public void` **destroy_commit**`( )`
  - **Usage**
    - Starts phase 2 of component shutdown. Multicasts the message to all subcomponent's managers.

- *destroy_finalize*
  `public void` **destroy_finalize**`( )`
  - **Usage**
    - Starts phase 3b of component shutdown. Multicasts the message to all subcomponent's managers.

- *destroy_kill*
  `public void` **destroy_kill**`( )`

    * Starts phase 3a of component shutdown. Multicasts the message to all subcomponent's managers.

- *destroy*
`public void` **destroy**`( )`
  - **Usage**
    * Destroys component. Multicasts the message to all subcomponent's managers.

- *externalize_begin*
`public void` **externalize_begin**`( )`
  - **Usage**
    * Starts phase 1 of externalization. Multicasts the message to all subcomponent's managers.

- *externalize_commit*
`public void` **externalize_commit**`( java.io.OutputStream` **stream** `)`
  - **Usage**
    * Starts phase 2 of externalization. Multicasts the message to all subcomponent's managers.

- *externalize_finalize*
`public void` **externalize_finalize**`( )`
  - **Usage**
    * Starts phase 3 of externalization. Multicasts the message to all subcomponent's managers.

- *externalize*
`public void` **externalize**`( )`
  - **Usage**
    * Externalizes component. Multicasts the message to all subcomponent's managers.

- *getCMID*
`public SOFACMID` **getCMID**`( )`
  - **Usage**
    * Utility function, returns cm identifier.

- *getCMState*
`public int` **getCMState**`( )`
  - **Usage**
    * Returns current state of cm.
  - **See Also**
    * `sofa.common.Const` (in 10.1.1, page 72)

- *getComponentDescriptor*

  - **Usage**
    * Utility function, gets component desriptor.

- *getDestroyAck*
`public boolean` **getDestroyAck**`( )`

  - **Usage**
    * Auxiliary method, returns the value of ackFlag.

- *getExternalizeAck*
`public boolean` **getExternalizeAck**`( )`

  - **Usage**
    * Auxiliary method, returns the value of externalizeFlag.

- *init*
`public void` **init**`( sofa.interfaces.ICManager` **parentCM,**
`sofa.interfaces.IRP2CB` **runPartForBuilder** `)`

  - **Usage**
    * Initializes component. First method called by Run Part on freshly-loaded component. Instantiates component builder and starts building and binding process.

- *isApplication*
`public boolean` **isApplication**`( )`

  - **Usage**
    * Boolean function, returns true for case of application's root component manager, otherwise returns false.

- *pause*
`public void` **pause**`( )`

  - **Usage**
    * Pauses component. Multicasts the message to all subcomponent's managers.

- *queryInterface*
`public Object` **queryInterface**`( sofa.common.SOFAIID` **iid** `)`

  - **Usage**
    * Returns reference to given (by iid) interface implemented by component. Returned reference is indirect (to the interface wrapper proxy object).

- *resume*
`public void` **resume**`( )`

  - **Usage**
    * Resumes paused component. Multicasts the message to all subcomponent's managers.

- *setCMID*

  - **Usage**
    * Utility function, sets cm identifier.

- *setComponentDescriptor*
  `public void` **setComponentDescriptor(**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Usage**
    * Utility function, sets component desriptor.

- *start*
  `public void` **start( )**
  - **Usage**
    * Starts component. Multicasts the message to all subcomponent's managers.

- *update*
  `public void` **update(** `sofa.node.repository.ComponentDescriptor` **newDescriptor**
)
  - **Usage**
    * Updates component. Multicasts the message to all subcomponent's managers. Stops component and starts update process.

1.1.6 Interface **IIfaceWrapper**

Control interface of SOFA Interface Wrappers. Handles some binding, interface behavior and traffic control. proxy object for indirect call through SOFA interfaces. There is one wrapper per one interface.

Declaration

public interface IIfaceWrapper

Methods

- *getIID*
  `public SOFAIID` **getIID( )**
  - **Usage**
    * Returns internal interface identifier (a string).

- *getWrapperObjectInstance*
  `public Object` **getWrapperObjectInstance( )**
  - **Usage**
    * Helper function. Returns reference to self.

- *off*

  - **Usage**
    * Blocks all communication through the wrapper.

- *on*
  `public void` **on**`( )`
  - **Usage**
    * Enables communication though the wrapper.

- *setInterfaceImplementation*
  `public void` **setInterfaceImplementation**`( java.lang.Object` **oRef** `)`
  - **Usage**
    * Sets target object that implements this SOFA/Java interface.

1.1.7 Interface **IRootObject**

Control interface of the Root Object of each component. Controls component's lifecycle.

Declaration

public interface IRootObject

Methods

- *bindInterfaces*
  `public void` **bindInterfaces**`( sofa.interfaces.ICBuilder` **cb** `)`
  - **Usage**
    * Binding SOFA component interfaces to Java implementation objects.

- *destroy_begin*
  `public void` **destroy_begin**`( )`
  - **Usage**
    * Prepare for component shutdown (phase 1).

- *destroy_commit*
  `public void` **destroy_commit**`( )`
  - **Usage**
    * Shutdown of component (phase 2).

- *do_restore*
  `public void` **do_restore**`( java.io.InputStream` **fi** `)`
  - **Usage**
    * Reads component state information from a stream. Used after component update.

    * `fi` - stream with state informastion

- *externalize_begin*
`public void` **externalize_begin**`( )`
  - **Usage**
    * Prepare for externalization (phase 1).

- *externalize_commit*
`public void` **externalize_commit**`( java.io.OutputStream` **fo** `)`
  - **Usage**
    * Externalize state information into a stream (phase 2).

- *externalize_finalize*
`public void` **externalize_finalize**`( )`
  - **Usage**
    * Cleanup and recovery after externalization (phase 3).

- *getDestroyAck*
`public boolean` **getDestroyAck**`( )`
  - **Usage**
    * Returns the state of destroyAck flag. used to test whether a component is ready for shutdown.

- *getExternalizeAck*
`public boolean` **getExternalizeAck**`( )`
  - **Usage**
    * Returns the state of externalizeAck flag. Used to test whether a component is ready to externalize its state information.

- *init*
`public void` **init**`( )`
  - **Usage**
    * Initialization of Root Object during component creation.

- *init*
`public void` **init**`( java.io.InputStream` **fi** `)`
  - **Usage**
    * Initialization of Root Object after component update.

- *pause*
`public void` **pause**`( )`
  - **Usage**
    * Stop component's internal threads (if any) until resume.

- *resume*
  `public void` **resume**`( )`
  - **Usage**
    - Resumes paused component's internal threads (if any).

- *start*
  `public void` **start**`( )`
  - **Usage**
    - Start component's internal threads (if any).

1.1.8 Interface **IRP2CB**

Interface between Run Part and Component Builder. Allows to create a pre-fetched component and to find what components implement a SOFA interface. CMID comes from CDL.

Declaration

public interface IRP2CB

Methods

- *makeComponent*
  `public ICManager` **makeComponent**`( sofa.interfaces.ICManager` **parent**`, java.lang.String` **CMID**`, java.lang.String` **producer**`, java.lang.String` **componentName**`, java.lang.String` **version** `)`
  - **Usage**
    - Implemented by Run Part. Instantiates and returns a reference to the component from Template Repository. This method is called indirectly by Component Builders when they need to instatntiate a subcomponent. A CB provides description of the subcomponent (that is hard-wired into the builder by CDL compoler) in three strings that are processed into a component descriptor. Technically, this method is similar to loadApplication() method
  - **Parameters**
    - `parent` - reference to parent component manager, i.e. the component that call this method
    - `CMID` - unique identification of the new component. Used for registration of its component manager.
    - `producer` - used to build-up a component descriptor
    - `complonentName` - used to buid-up a component descriptor
    - `version` - used to build-up a component descriptor
  - **See Also**
    - `sofa.vers.sofa.vers`

- *unRegisterCM*

- *updateComponent*
  `public ICBuilder` **updateComponent**`( sofa.interfaces.ICManager` **myCM**`, sofa.node.repository.ComponentDescriptor` **cd** `)`

1.1.9 Interface IRP2Sh

User Shell interface to Run Part. Allows user to load, start, stop etc. a component.

Declaration

public interface IRP2Sh

Methods

- *loadApplication*
  `public ICManager` **loadApplication**`( sofa.node.repository.Node2TR` **repository**`, sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    - loads an application from template repository.
  - **Parameters**
    - `repository` - references to a Template Repository (connected via RMI)
    - `cd` - specification of the "aplication" component to load

1.1.10 Interface ISOFAComponentInterface

Generic parent of all SOFA component interfaces. No functionality. (Like IUnknown in COM, maybe used in future)

Declaration

public interface ISOFAComponentInterface

1.1.11 Interface ISOFAMessages

NOT USED! Used by components to exchange messages. Used also for internal purposes by Component Managers and Root objects.

Declaration

public interface ISOFAMessages

METHODS

- *broadcast*
  `public void` **broadcast**`( java.lang.String` **msg** `)`
  - **Usage**
    - Sends a meesage to all subordinate SOFA entities of this objects.
  - **Parameters**
    - `msg` - text information. Its format will be specified later.

- *msg*
  `public void` **msg**`( java.lang.String` **msg** `)`
  - **Usage**
    - Sends a meesage to the superordinate SOFA entity of this object.
  - **Parameters**
    - `msg` - text information. Its format will be specified later.

1.1.12 INTERFACE ITR2RP

Interface between Template Repository and Run Part. Allows to load a binary image of a component and to find components that implement specified SOFA interface.

DECLARATION

public interface ITR2RP

METHODS

- *getStreamWithBinaryImageOfComponent*
  `public InputStream` **getStreamWithBinaryImageOfComponent**`(`
  `sofa.common.SOFAComponentID` **compID** `)`
  - **Usage**
    - OBSOLETE. Replaced by RMI in new template repository.
  - **See Also**
    - `sofa.node.repository.sofa.node.repository`

1.1.13 INTERFACE ITR2Sh

OBSOLETE. Interface between Template Repository and User Shell. Provides information for SOFA users about components stored in TR.

Declaration

public interface ITR2Sh

Methods

- *getComponentList*
  `public ComponentDescriptorList` **`getComponentList( )`**
  - **Usage**
    - OBSOLETE. Gets a list of components available in TR.

Chapter 2

Package sofa.abstr.util

Package Contents *Page*

Classes

Contains some auxiliary classes those are used by the supperior package. This includes a list of interface wrapper references and component manager references.

We decided to use string-based identifiers to provide some elementar dynamic type-checking, simple naming of interfaces, and also for debugging purposes.

Classes in this package manage a flat list of various identifiers.

2.1 Classes

2.1.1 Class SOFAComponentManagerList

Utility class, serves as a dynamic list of component managers. All functionality inherited from java.util.Vector. Possibility to look up component manager by its cmID.

Declaration

public class SOFAComponentManagerList
extends java.util.Vector

Constructors

- *SOFAComponentManagerList*
  `public` **SOFAComponentManagerList**`( )`

Methods

- *addItem*
  `public boolean` **addItem**`( sofa.abstr.util.SOFAComponentManagerListItem` **item** `)`
- *getItem*
  `public SOFAComponentManagerListItem` **getItem**`( int` **index** `)`
- *lookupItemByCMID*
  `public SOFAComponentManagerListItem` **lookupItemByCMID**`( sofa.common.SOFACMID` **cmID** `)`
- *removeItemByCMID*
  `public void` **removeItemByCMID**`( sofa.common.SOFACMID` **cmID** `)`
- *removeItemByRef*
  `public void` **removeItemByRef**`( sofa.interfaces.ICManager` **cm** `)`

2.1.2 Class SOFAComponentManagerListItem

Utility class, serves an item of dynamic list of component managers.

Declaration

public class SOFAComponentManagerListItem
extends java.lang.Object

CONSTRUCTORS

- *SOFAComponentManagerListItem*
  `public` **SOFAComponentManagerListItem**`( sofa.common.SOFACMID` **cmid**`, sofa.interfaces.ICManager` **refCManager** `)`

METHODS

- *getCMID*
  `public SOFACMID` **getCMID**`( )`
- *getrefCManager*
  `public ICManager` **getrefCManager**`( )`
- *getrefCManager2*
  `public ICM2CM` **getrefCManager2**`( )`
- *toString*
  `public String` **toString**`( )`

2.1.3 CLASS SOFAInterfaceWrapperList

Utility class, serves as a dynamic list of interface wrappers. All functionality inherited from java.util.Vector. Possibility to look up interface wrapper by its iid.

DECLARATION

public class SOFAInterfaceWrapperList
extends java.util.Vector

CONSTRUCTORS

- *SOFAInterfaceWrapperList*
  `public` **SOFAInterfaceWrapperList**`( )`

METHODS

- *addItem*
  `public boolean` **addItem**`( sofa.abstr.util.SOFAInterfaceWrapperListItem` **item** `)`
- *getItem*
  `public SOFAInterfaceWrapperListItem` **getItem**`( int` **index** `)`
- *lookupItemByIID*
  `public SOFAInterfaceWrapperListItem` **lookupItemByIID**`( sofa.common.SOFAIID`

2.1.4 Class **SOFAInterfaceWrapperListItem**

Utility class, serves an item of dynamic list of interface wrappers.

Declaration

public class SOFAInterfaceWrapperListItem
extends java.lang.Object

Constructors

- *SOFAInterfaceWrapperListItem*
  `public` **SOFAInterfaceWrapperListItem**`( sofa.common.SOFAIID` **iid**`,`
  `sofa.interfaces.IIfaceWrapper` **refIfaceWrapper** `)`

Methods

- *getIID*
  `public SOFAIID` **getIID**`( )`
- *getrefIfaceWrapper*
  `public IIfaceWrapper` **getrefIfaceWrapper**`( )`

Chapter 3

Package sofa.shell

Package Contents *Page*

Classes

User Shell. This package contains classes needed for User Shell including graphical user interface.

This is the main interface to SOFAnode for regular users. It displays components available in related Template Repository, allows loading, execution and upgrade of components. There is no other interface no other interface for SOFAnode users. On the other hand, SOFAnode administrators can use administrative interface of corresponding parts. See sofa.node.repository.shell for example.

3.1 Classes

3.1.1 Class **Shell**

Some core functionality of the user shell. In fact, there is only one function because the rest is coverred by GUI in ShellFrame class.

Declaration

public class Shell
extends java.lang.Object

Constructors

- *Shell*
  `public` **Shell**`( )`
  - **Usage**
    - default constructor - no action

Methods

- *matchForUpgrade*
  `public static boolean` **matchForUpgrade**`(`
  `sofa.node.repository.ComponentDescriptor` **a,**
  `sofa.node.repository.ComponentDescriptor` **b** `)`
  - **Usage**
    - Compares two components whether thay are compatiple for upgrade. Tests component names and versions.
  - **Parameters**
    - `a` - the first (older) component
    - `b` - the new component (upgrade candidate)

3.1.2 Class **ShellFrame**

User Shell graphical interface. Allows loading, running, upgrading and termination of components. Works with "new" Template repository via Java RMI. For now, contains hard-wired RMI naming reference to "rmi://sofa/servers/repository"

Declaration

public class ShellFrame
extends javax.swing.JFrame

Serializable Fields

- private Globals fglobals
  –

Constructors

- *ShellFrame*
  `public` **ShellFrame**`( sofa.application.Globals` **gl** `)`
  - **Usage**
    * Creates the frame.
  - **Parameters**
    * `gl` - reference to the "global object" of the application

Methods

- *processWindowEvent*
  `protected void` **processWindowEvent**`( java.awt.event.WindowEvent` **e** `)`

Chapter 4

Package sofa.util

Package Contents *Page*

Classes

Contains some suporting utility classes that have not much in common with SOFA. It is posssible that this package would be merged with some similar one. Nothing interesting here.

4.1 Classes

4.1.1 Class MessageBox

Auxiliary class for displaying various mesages. Usefull for debugging purposes.

Declaration

public class MessageBox
extends java.lang.Object

Constructors

- *MessageBox*
  `public` **MessageBox**`( )`

Methods

- *showMessage*
  `public static void` **showMessage**`( java.lang.Object` **sender**`, java.lang.String` **message** `)`
  - **Usage**
    - Displays a box with a message.
  - **Parameters**
    - `sender` - referrence to the sender object, will be displayed
    - `message` - the message, displayed 'as is'

4.1.2 Class Monitor

Generic monitor class used for thread synchronization where internal monitors are not possible (thread not owner exception).

Declaration

public class Monitor
extends java.lang.Object

Constructors

- *Monitor*
  `public` **Monitor**`( )`
  - **Usage**

METHODS

- *close*
  `public synchronized void` **close**`( )`
  - **Usage**
    - Closes the monitor as soon as possible. Other threads must wait till open.

- *open*
  `public synchronized void` **open**`( )`
  - **Usage**
    - Opens the monitor. One of waiting threads is executed.

4.1.3 CLASS StreamCopier

Reads data from InputStream to specified file.

DECLARATION

public class StreamCopier
extends java.lang.Object

CONSTRUCTORS

- *StreamCopier*
  `public` **StreamCopier**`( )`

METHODS

- *readFileFromInputStream*
  `public static long` **readFileFromInputStream**`( java.io.InputStream` **source**`, java.lang.String` **targetFileName** `)`
  - **Parameters**
    - `source` - input stream with data
    - `tragetFilename` - name of destination file that will be created

4.1.4 CLASS StringList

Auxiliary class capable of halding strings. Similar to vector, but successor of ListModel. Used to avoid casting problems when using tree views.

Declaration

public class StringList
extends javax.swing.DefaultListModel

Constructors

- *StringList*
  `public` **StringList**`( )`

Methods

- *addString*
  `public void` **addString**`( java.lang.String  s )`
  - **Usage**
    - Add new strring into the list
  - **Parameters**
    - `s` - the string

Chapter 5

Package sofa.abstr

Package Contents *Page*

Classes

Contains templates (ancessors) of various basic SOFA classes that make a backbone of SOFA infrastructure. These templates are used by CDL compiler as teplates for end-user SOFA components.

All user-written components (and automatically generated classes) are successors of these templates. This concept makes the job for CDL compiler much easier and the generated Java source files are short and comprehensive.

5.1 Classes

5.1.1 Class **SOFACBTemplate**

Generic Component Builder. The template class covers approx. half of the functionality of component builders. It creates data structures of the component, instantiates the Root object and interface wrappers and takes care about the bindings. This template class contains several virtual methods that must be overriden in real builders by code geenrated from CDL.

Declaration

public abstract class SOFACBTemplate
extends java.lang.Object
implements sofa.interfaces.ICBuilder

Fields

- public ICB2CM fCB2CM
  - auxiliary reference to parent component manager
- public IRootObject fRoot
  - auxiliary reference to the root object

Constructors

- *SOFACBTemplate*
  `public` **SOFACBTemplate**`( )`

Methods

- *bind*
  `public void` **bind**`( )`
  - **Usage**
    - Performs all binding in the component. Also calls root.bindInterfaces().
  - **See Also**
    - `sofa.abstr.SOFARootTemplate.bindInterfaces`

- *buildComponent*
  `public void` **buildComponent**`( sofa.interfaces.ICB2CM` **cm2cb**`, boolean` **createWrappers** `)`
  - **Usage**
    - Builds the component by calling all necessary methods. Creates interface

  - **Parameters**
    - `cm2cb` - reference to parent Component Manager
    - `createWrappers` - if true then wrappers would be created. This is set true when building the component and to fasle while updating the component.

- *createInterfaceWrappers*
`protected void` **createInterfaceWrappers**`( )`

  - **Usage**
    - Creates all interface wrappers. Must be overriden by code generated from CDL.

- *createRootObject*
`protected IRootObject` **createRootObject**`( )`

  - **Usage**
    - Creates the root object and returns a reference to it. Must be overriden by code generated from CDL.

- *getInterfaceImplementation*
`public Object` **getInterfaceImplementation**`( sofa.common.SOFACMID` **cmID**`, sofa.common.SOFAIID` **iid** `)`

  - **Usage**
    - Returns a reference to an object that implements specified interface. Currently it can perform lookup only in one Component Manager This method is used for required interfaces only. Teh call must be generated from CDL.
  - **Parameters**
    - `cmID` - Component Mananaget that we aks for the reference
    - `iid` - SOFA interface ID
  - **See Also**
    - `sofa.common.SOFACMID` (in 10.1.2, page 73)
    - `sofa.common.SOFAIID` (in 10.1.4, page 74)

- *getRootObject*
`public IRootObject` **getRootObject**`( )`

  - **Usage**
    - Returns a reference to the root object of this component. This is an auxiliary method.

- *instantiateSubComponents*
`protected void` **instantiateSubComponents**`( )`

  - **Usage**
    - Instatiates all subcomponents. Must be overriden by code generated from CDL.

- *registerIfW*
`protected final void` **registerIfW**`( sofa.interfaces.IIfaceWrapper` **iw** `)`

  - **Usage**

    * Registers (new) interface wrapper in parent Component Manager. This method is called by ancessors in their overriden createInterfaceWrappers This is an auxiliary method.
  - **See Also**
    * `sofa.abstr.SOFACBTemplate.createInterfaceWrappers` (in 5.1.1, page 33)

- *setIntefaceImplementation*
  `public void` **setIntefaceImplementation**`( sofa.common.SOFAIID` **iid**`, java.lang.Object` **oRef** `)`
  - **Usage**
    * Sets a reference in a interface wrapper pointing to specified implementation object. Looks up wrapper by the specified ID. This method used for provided interfaces only. The call must be generated from CDL.
  - **Parameters**
    * `iid` - SOFA interface ID
    * `oRef` - reference to an object that implements specified interface
  - **See Also**
    * `sofa.common.SOFAIID` (in 10.1.4, page 74)

5.1.2 Class **SOFACMTemplate**

Generic Component Manager. The template class covers most of the functionality of component managers. Its methods can be divided into three groups: (a) regular methods that control the component, (b) virtual methods that must be overriden by instances of real Component Builders (generated from CDL by CDLtoSOFA compiler) and (c) some auxiliary method that provide service functions like registrations of subcomponents and interface wrappers, assignment and query of various flags etc. However this template covers all desired functionality of CMs, so the real CMs are quite simple. For now they are intended only for some non-standard modifications.

Declaration

public abstract class SOFACMTemplate
extends java.lang.Object
implements sofa.interfaces.ICManager, sofa.interfaces.ICB2CM, sofa.interfaces.ICM2CM

Fields

- public ICManager parent
  - auxiliary reference to parent component manager

Constructors

- *SOFACMTemplate*

  - **Usage**
    * This constructor just creates some internal data structures and initializes same private variables.

METHODS

- *createComponentBuilderInstance*
  `protected ICBuilder` **createComponentBuilderInstance**`( )`

  - **Usage**
    * Creates component builder and returs its reference. This method is overriden by real component managers.

- *destroy_begin*
  `public void` **destroy_begin**`( )`

  - **Usage**
    * Makes a component to prepare for shutdown. Sets destroyFlag, blocks all interface wrappers and informs all subcomponents. Recursive method.

- *destroy_commit*
  `public void` **destroy_commit**`( )`

  - **Usage**
    * Actually performs shutdown actions. This call is passed to the Root object of the component should using its overriden method destroy_commit(). This method is recursivelly called on all subcomponents.

- *destroy_finalize*
  `public void` **destroy_finalize**`( )`

  - **Usage**
    * The last stage of shutdown. Recursive method, it calls all subcomponents, then destroys interface wrappers, the Componnet Builder, and auxiliary data structures like lists of IWs and sub CMs. Also calls garbage collector and pro-forma sets a new state of the component.

- *destroy_kill*
  `public void` **destroy_kill**`( )`

  - **Usage**
    * The last stage of shutdown, alternative call to destroy_finalize(). It is used in case that a component does not reply to destroy Flag by setting the destroyAck. This method destroys interface wrappers, the Componnet Builder, and auxiliary data structures like lists of IWs and sub CMs. Also calls garbage collector and pro-forma sets a new state of the component.

- *destroy*
  `public synchronized void` **destroy**`( )`

  * Destroys the component and unloads it from memory. Also releases all involved data structures. This is a main non-recursicve method that internally calls recursive destroy_begin(), destroy_commit() and destroy_finalize().
- **See Also**
  * `sofa.abstr.SOFACMTemplate.destroy_begin` (in 5.1.2, page 35)
  * `sofa.abstr.SOFACMTemplate.destroy_commit` (in 5.1.2, page 35)
  * `sofa.abstr.SOFACMTemplate.destroy_finalize` (in 5.1.2, page 35)
  * `sofa.common.Const` (in 10.1.1, page 72)

- *externalize_begin*
`public void` **externalize_begin**`( )`
  - **Usage**
    * Makes a component to prepare for externalization. Sets externalizeFlag, blocks all interface wrappers and informs all subcomponents. Recursive method.

- *externalize_commit*
`public void` **externalize_commit**`( java.io.OutputStream` **stream** `)`
  - **Usage**
    * Actually performs externalization. Root object of the component should using its overriden method externalize_commit() write to the stream any number of text lines in the format VARIABLE-NAME=VALUE. This method is recursivelly called on all subcomponents.
  - **Parameters**
    * `stream` - file (OutputStream) where to write state information if null, then a new temporary file will be created

---

- *externalize_finalize*
`public void` **externalize_finalize**`( )`
  - **Usage**
    * The last stage of externalization. Opens wrappers, restores externalizeFlag and thus allows normal operation of the componnet. Also a recursive method, so it calls all subcomponents.

- *externalize*
`public synchronized void` **externalize**`( )`
  - **Usage**
    * Extenalizes a state information of this and all subordinated subcomponents. This is a main non-recursicve method that internally calls recursive externalize_begin(), externalize_commit() and externalize_finalize(). This method throws an exception, if the component does not respond to the externalizeFlag or due to I/O error.
  - **See Also**
    * `sofa.abstr.SOFACMTemplate.externalize_begin` (in 5.1.2, page 36)
    * `sofa.abstr.SOFACMTemplate.externalize_commit`
    * `sofa.abstr.SOFACMTemplate.externalize_finalize` (in 5.1.2, page 36)
    * `sofa.common.Const` (in 10.1.1, page 72)

- *getCMID*
  `public SOFACMID` **getCMID**`( )`
  - **Usage**
    - Returns component manager ID. Auxiliary method.
  - **See Also**
    - `sofa.common.SOFACMID` (in 10.1.2, page 73)

- *getCMList*
  `public SOFAComponentManagerList` **getCMList**`( )`
  - **Usage**
    - Returns a reference to the list of (sub)component managers that is stored in each component manager.

- *getCMState*
  `public int` **getCMState**`( )`
  - **Usage**
    - Gets a global state of this component. Used many times for decisions how to threat this component.
  - **See Also**
    - `sofa.common.Const` (in 10.1.1, page 72)

- *getComponentDescriptor*
  `public ComponentDescriptor` **getComponentDescriptor**`( )`
  - **Usage**
    - Returns component descriptor of this component. Auxiliary method.

- *getDestroyAck*
  `public boolean` **getDestroyAck**`( )`
  - **Usage**
    - Returns the value of destroyFlag. Auxiliary method.

- *getExternalizeAck*
  `public boolean` **getExternalizeAck**`( )`
  - **Usage**
    - Returns the value of externalizeFlag. Auxiliary method.

- *getInfo*
  `public final String` **getInfo**`( )`
  - **Usage**
    - Fills up a string with various current state information of this component manager. This method is used for exaple by SOFA Demo Application (Runpart View).

- *getRPRef*
  `public IRP2CB` **getRPRef**`( )`

  - **Usage**
    * Returns a reference to the Run PArt. Auxiliary method.

- *init*
`public void` **init**`( sofa.interfaces.ICManager` **parentCM**`,`
`sofa.interfaces.IRP2CB` **runPartForBuilder** `)`

  - **Usage**
    * De facto creates the component. Creates component builder and calls its methods to set up the component.
  - **Parameters**
    * `parentCM` - reference to parent component manager
    * `runPartForBuilder` - reference to the runpart that is passed to component builder
  - **See Also**
    * `sofa.abstr.SOFACMTemplate.createComponentBuilderInstance` (in 5.1.2, page 35)
    * `sofa.abstr.SOFACBTemplate.buildComponent`
    * `sofa.abstr.SOFACBTemplate.bind` (in 5.1.1, page 32)

- *isApplication*
`public boolean` **isApplication**`( )`

  - **Usage**
    * Returns true if this component is "runnable". This method is currently used only by User Shell.

- *lookupInterface*
`public IIfaceWrapper` **lookupInterface**`( sofa.common.SOFAIID` **iid** `)`

  - **Usage**
    * Looks for specified SOFA interface within this component manager. Auxiliary method.
  - **Parameters**
    * `iid` - SOFA name of the interface to look for

- *pause*
`public synchronized void` **pause**`( )`

  - **Usage**
    * Sets a pauseFlag in this component and all subcomponents. All involved components are required to stop all operation at the first suitable occasion. Components with live threads can/should use a monitor object.
  - **See Also**
    * `sofa.util.Monitor` (in 4.1.2, page 28)

- *queryInterface*
`public Object` **queryInterface**`( sofa.common.SOFAIID` **iid** `)`

  - **Usage**
    * Returns a reference to the wrapper object (implementation object??) of an

- **Parameters**
  - `iid` - SOFA name of the requested interface

- *registerComponentManager*
  `public void` **registerComponentManager**`( sofa.common.SOFACMID` **cmID,** `sofa.interfaces.ICManager` **refCM** `)`
  - **Usage**
    - Registers (new) component manager (of a subcomponent) in this component Manager. Uses SOFA CMID and a reference to the new component manager. Auxiliary method.
  - **Parameters**
    - `cmID` - SOFA name of the component manager of a subcomponent
    - `refCM` - reference to the component manager of a subcomponent

---

- *registerInterface*
  `public void` **registerInterface**`( sofa.common.SOFAIID` **iid,** `sofa.interfaces.IIfaceWrapper` **refIfaceWrapper** `)`
  - **Usage**
    - Registers (new) interface wrapper by its component Manager. Uses SOFA IID and a reference to interface wrapper object. Auxiliary method.
  - **Parameters**
    - `iid` - SOFA name of the interface
    - `refIfaceWrapper` - reference to the wrapper object

- *resume*
  `public synchronized void` **resume**`( )`
  - **Usage**
    - Opposite to pause(). Allows this component and all subcomponent to perform normal operation.

- *setApplication*
  `public void` **setApplication**`( boolean` **value** `)`
  - **Usage**
    - Marks this component as "runnable". Call of this method should be generated from CDL, but CDL now lacks any support for this.
  - **Parameters**
    - `value` - true for applications, false for passive components

---

- *setCMID*
  `public void` **setCMID**`( sofa.common.SOFACMID` **id** `)`
  - **Usage**
    - Sets component manager ID. Auxiliary method.
  - **See Also**
    - `sofa.common.SOFACMID` (in 10.1.2, page 73)

- *setComponentDescriptor*
  `public void` **setComponentDescriptor(**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Usage**
    - Sets component descriptor of this method. Component descriptors are used in connection with Template Repository. Auxiliary method.
  - **See Also**
    - `sofa.repository.ComponentDescriptor` (in 15.1.1, page 112)

- *start*
  `public synchronized void` **start( )**
  - **Usage**
    - Creates and starts a thread in already loaded component. Until this call the component is "dead" and should not accept or produce any interface calls. The call on this method also activates all subcomponents.

- *toString*
  `public String` **toString( )**
  - **Usage**
    - Returns a string with a class name of the component manager class. Auxiliary method.

- *update*
  `public synchronized void` **update(** `sofa.node.repository.ComponentDescriptor` **newDescriptor)**
  - **Usage**
    - Updates the component (whole architecture) whit a new (specified) version from Template Reposiroty, i.e. it implements DCUP.
  - **Parameters**
    - `newDescriptor` - describes the new version of the component we use for update

5.1.3 Class **SOFAIWTemplate**

Generic Interface Wrapper. The template covers some basic functions of interface wrappers like setting them on and off, assigment and query of the implementation object etc.

Declaration

public abstract class SOFAIWTemplate
extends java.lang.Object
implements sofa.interfaces.IIfaceWrapper

CONSTRUCTORS

- *SOFAIWTemplate*
  `public` **SOFAIWTemplate**`( )`

METHODS

- *enterInterfaceFunctionCall*
  `protected void` **enterInterfaceFunctionCall**`( sofa.abstr.SOFAIWTemplate` **ifWrapper**`, java.lang.String` **functionName** `)`
  - **Usage**
    - Performs a check whether to pass or block an incomming call. Call to this method are in real wrappers generated from CDL.

      NOT YET IMPLEMENTED!

- *getIID*
  `public SOFAIID` **getIID**`( )`
  - **Usage**
    - Returns a SOFA name of the interface we wrap.

- *getWrapperObjectInstance*
  `public final Object` **getWrapperObjectInstance**`( )`
  - **Usage**
    - Returns the reference to the implementation object.

- *off*
  `public void` **off**`( )`
  - **Usage**
    - Sets this wraper off. Incomming calls are blocked and threated according to the policy of this wrapper (not yet implemented).

- *on*
  `public void` **on**`( )`
  - **Usage**
    - Opens this wrapper for incomming (and possibly any waiting) calls.

- *setInterfaceImplementation*
  `public void` **setInterfaceImplementation**`( java.lang.Object` **oRef** `)`
  - **Usage**
    - Sets the implementation object for this wrapper, i.e. for this SOFA interface.
  - **Parameters**
    - `oRef` - reference to the implementation object

5.1.4 Class **SOFARootTemplate**

Generic Root object. This template is the real functional body of each component. This template class covers three kinds of components, i.e. passive components (libraries), single thread components and multi-thread components. The class contains methods that are used to control the lifecycle of the component and usually are one-to-one mapped to equal-named methods of component manager or builder classes (i.e. from ICManager or ICBuilder interfaces). Each Root object creates a new thread for the component and starts its runnable.run() method during call of start() method. Thus all components run in their own threads and should not interfere each other (but components are not allowed to poerform any bussy-waiting).

Declaration

public abstract class SOFARootTemplate
extends java.lang.Object
implements java.lang.Runnable, sofa.interfaces.IRootObject

Fields

- public Monitor gate
  - A monitor used for thread synchronization.
- public boolean pauseFlag
  - indicates that the component is requested to stop all threads until resume
- public boolean destroyFlag
  - indicates that the component is rerquested to shutdown
- public boolean updateFlag
  - indicates that this component should prepare for an update
- public boolean externalizeFlag
  - indicates that this component should prepare for externalization
- public boolean externalizeAck
  - component's response to the pauseFlag - "I am ready"
- public boolean destroyAck
  - component's response to the destroyFlag - "I am ready"

Constructors

- *SOFARootTemplate*
  `public` **SOFARootTemplate**`( )`
  - **Usage**
    - default constructor. Just creates the monitor.
  - **See Also**
    - `sofa.abstr.SOFARootTemplate.gate`

Methods

- *bindInterfaces*
  `public void` **bindInterfaces**`( sofa.interfaces.ICBuilder` **cb** `)`
  - **Usage**
    - No default action. Real component that instantiate subcomponents must here map java variables to SOFA references.
  - **Parameters**
    - `cb` - reference to Component Builder
  - **See Also**
    - `sofa.abstr.SOFACBTemplate.getInterfaceImplementation`

- *destroy_begin*
  `public void` **destroy_begin**`( )`
  - **Usage**
    - Default action at the beginning of component shutdown. Just sets destroyFlag true. Real components would like to finish their work here. Then destroyAck is set true.
  - **See Also**
    - `sofa.abstr.SOFARootTemplate.destroyFlag`
    - `sofa.abstr.SOFARootTemplate.destroyAck`

- *destroy_commit*
  `public void` **destroy_commit**`( )`
  - **Usage**
    - Called by Component Manager immediately before the root object is destroyed. Default is no action but this is a good place for cleanup code.

- *do_restore*
  `public void` **do_restore**`( java.io.InputStream` **fi** `)`
  - **Usage**
    - Restores component's state from a given stream. Should be overriden. Default is no action.

    * `sofa.abstr.SOFARootTemplate.externalize_commit`

- *externalize_begin*
  `public void` **externalize_begin**`( )`
  - **Usage**
    * Default behavior at the beginning of externalization. Just set externalizeFlag and externalizeAck true.
  - **See Also**
    * `sofa.abstr.SOFARootTemplate.externalizeFlag`
    * `sofa.abstr.SOFARootTemplate.externalizeAck`

- *externalize_commit*
  `public void` **externalize_commit**`( java.io.OutputStream` **fo** `)`
  - **Usage**
    * Flushes externalization data to a stream. This should be overridden to produce some usefull data in form variable=value per each line.
  - **Parameters**
    * `fo` - open stream where to externalize to
  - **See Also**
    * `sofa.abstr.SOFARootTemplate.do_restore`

- *externalize_finalize*
  `public void` **externalize_finalize**`( )`
  - **Usage**
    * Default action at the end of externalization. Just sets pauseFlag and externalizeFlag to false.

- *getDestroyAck*
  `public boolean` **getDestroyAck**`( )`
  - **Usage**
    * Returns the state of destroyAck. Used by Component Manager to test whether the component is ready to shutdown.

- *getExternalizeAck*
  `public boolean` **getExternalizeAck**`( )`
  - **Usage**
    * Returns the state of externalizeAck. Used by Component Mananger to test whether the component is ready to externalize.

- *init*
  `public void` **init**`( )`
  - **Usage**
    * Called on the beggining of component setup. Does nothing in this template, but real Roots should instantiate here their own data, especially interface

---

- *init*
  `public void` **init**`( java.io.InputStream` **fi** `)`
  - **Usage**
    - Like init(), but gets former externalized state information from the supplied stream. Intended for DCUP. UNDER DEVELOPMENT.
  - **Parameters**
    - `fi` - stream with externalized state information

- *pause*
  `public void` **pause**`( )`
  - **Usage**
    - Called by Component Manager when the component is requested to stop. Default action is no operation. This method should be overriden if there is some work that must be done before pausing. At the end, pauseFlag should be set true.
  - **See Also**
    - `sofa.abstr.SOFARootTemplate.pauseFlag`

- *resume*
  `public void` **resume**`( )`
  - **Usage**
    - Called by Component manager when the component should resume. The method ensures that all flags are false and wakes up the main thread.
  - **See Also**
    - `sofa.abstr.SOFARootTemplate.pause` (in 5.1.4, page 45)

- *run*
  `public void` **run**`( )`
  - **Usage**
    - No default action. Should be overriden. At this point the component's programer gets control, so this method equals to the main() function in regular C programs.

- *start*
  `public void` **start**`( )`
  - **Usage**
    - Starts component that has been successfully loaded. Technicaly, it creates new thread for the run() mehod.
  - **See Also**
    - `sofa.abstr.SOFARootTemplate.run` (in 5.1.4, page 45)

Chapter 6

Package sofa.exceptions

Package Contents *Page*

Classes

Definitions of all SOFA exceptions. This package introduces a few brand new SOFA exceptions that are used mainly by Component Managers, Builders, Root Objects and Interface Wrappers. We decided to introduce SOFA exception base class (ESOFAException) to help programmers to distinguish exceptions raised by the SOFA framework from anyone's else exceptions.

6.1 Classes

6.1.1 Class **ESOFAClassLoaderException**

Specialized exception raised by classloaders. Indicates problems in Template Repository or Run Part concerning loading or running SOFA components.

Declaration

public class ESOFAClassLoaderException
extends sofa.exceptions.ESOFAException

Constructors

- *ESOFAClassLoaderException*
  `public` **ESOFAClassLoaderException**`( java.lang.String` **msg** `)`

6.1.2 Class **ESOFACMNoCBInstance**

Specialized exception that can be raised by Component Manager if it can not find Component Builder during component creation or update.

Declaration

public class ESOFACMNoCBInstance
extends sofa.exceptions.ESOFAException

Constructors

- *ESOFACMNoCBInstance*
  `public` **ESOFACMNoCBInstance**`( java.lang.String` **msg** `)`

6.1.3 Class **ESOFAComponentImageNotFound**

OBSOLETE. Specialized exception that can be raised during component creation or update. Raised by old implementation of Template Repository.

Declaration

public class ESOFAComponentImageNotFound
extends sofa.exceptions.ESOFAException

CONSTRUCTORS

- *ESOFAComponentImageNotFound*
  `public` **ESOFAComponentImageNotFound**`( java.lang.String` **msg** `)`

6.1.4 CLASS ESOFAException

Generic SOFA exception. Functionality is inherited from java.lang.Exception

DECLARATION

public class ESOFAException
extends java.lang.Exception

CONSTRUCTORS

- *ESOFAException*
  `public` **ESOFAException**`( java.lang.String` **msg** `)`

6.1.5 CLASS ESOFAExternalizationTimeout

Specialized exception that an occur during externalization of SOFA component.

DECLARATION

public class ESOFAExternalizationTimeout
extends sofa.exceptions.ESOFAException

CONSTRUCTORS

- *ESOFAExternalizationTimeout*
  `public` **ESOFAExternalizationTimeout**`( )`
- *ESOFAExternalizationTimeout*
  `public` **ESOFAExternalizationTimeout**`( java.lang.String` **msg** `)`

6.1.6 CLASS ESOFAInvalidObjectReference

Specialized exception that can be raised by Interface Wrappers, Component Builders or Component Managers while binding components during component creation or update.

Declaration

public class ESOFAInvalidObjectReference
extends sofa.exceptions.ESOFAException

Constructors

- *ESOFAInvalidObjectReference*
  `public` **ESOFAInvalidObjectReference**`( java.lang.String` **msg** `)`

Chapter 7

Package sofa.vers

Package Contents *Page*

Contains versioning support for SOFA. Used mainly in Template Repository implementation. Subject of master thesis of Stanislav Dobry (graduated in 2001). Well documented in hsi master thesis in more detail.

7.1 Interfaces

7.1.1 Interface **VersionAccess**

Version information about one element implementation. To be used in the template marker 'version' attribute

Declaration

public interface VersionAccess
implements java.lang.Comparable

Fields

- public static final int REV_PRIMITIVE
  - The possible levels of revision data.
- public static final int REV_COMPONENT
  -
- public static final int REV_TRAIT
  -
- public static final int INCOMPARABLE
  - Results of version data comparison. Must conform to @see java.lang.Comparable.compareTo() return values.

    The semantics is that of a sub/super-set relation, as follows (same for RevisionData and VariantData): A <B == A.compareTo(B) <0 == A is a subset of B

    Concerning compatibility of the components that are described by this version data, this means that [revision] B can replace A but not vice versa [variant] ??? A can replace B but not vice versa (needs research)
- public static final int EQUAL
  -
- public static final int PRECEDES
  -
- public static final int FOLLOWS
  -
- public static final int SUBSET
  -
- public static final int SUPERSET

METHODS

- *getBranchID*
  `public String` **getBranchID**`( )`
  - **Returns** - The name of the branch

- *getBranchString*
  `public String` **getBranchString**`( )`
  - **Returns** - The "branch=...", "rev=...", "var=..." strings used for URI naming; uses the REV_COMPONENT level as per TR 9/2000.

- *getRevID*
  `public String` **getRevID**`( )`
  - **Returns** - The "P.R.B" for REV_COMPONENT, "N" for REV_PRIMITIVE, 'null' if data cannot be represented by required revid level. Used for CDL inclusion and human consumption.

- *getRevID*
  `public String` **getRevID**`( int` **level** `)`

- *getRevString*
  `public String` **getRevString**`( )`

- *getTag*
  `public String` **getTag**`( )`
  - **Returns** - The unique version tag

- *getVarID*
  `public String` **getVarID**`( )`
  - **Returns** - String describing the variant expression

- *getVarString*
  `public String` **getVarString**`( )`

- *toString*
  `public String` **toString**`( )`
  - **Returns** - The complete version part of component URI name

7.1.2 INTERFACE **VersionComparison**

Operations for comparing the current version data with other one.

DECLARATION

```
public interface VersionComparison
```

METHODS

- *canConditionallyReplace*
  `public boolean` **canConditionallyReplace**`( sofa.vers.VersionData` **anotherVersion** `)`
  - **Usage**
    - (this checks the contextual compatibility)

- *canReplace*
  `public boolean` **canReplace**`( sofa.vers.VersionData` **anotherVersion** `)`
  - **Usage**
    - Attempt to determine whether this version can replace the 'anotherVersion'. Performs compatibility type checks as defined in the techreport plus variant compatibility check.

- *compareRevision*
  `public int` **compareRevision**`( sofa.vers.VersionData` **anotherVersion** `)`
  - **Returns** - -1/0/1 if this revision precedes/equals/follows the 'anotherVersion'.

- *compareVariant*
  `public int` **compareVariant**`( sofa.vers.VersionData` **anotherVersion** `)`
  - **Returns** - -1/0/1 if this variant is subset/equal/superset of the 'anotherVersion'.

7.2 Classes

7.2.1 CLASS DuplicateElementException

Thrown when attempting to add a data element which already exists in a set and must not be overwritten.

DECLARATION

public class DuplicateElementException
extends java.lang.Exception

CONSTRUCTORS

- *DuplicateElementException*
  `public` **DuplicateElementException**`( )`

7.2.2 CLASS RevisionData

Holds revision data of one component/type. The idea is to store, if possible/practical, only the trait-based revision data and compute the other levels on-demand. Concerning descriptive data, only the important

Declaration

public class RevisionData
extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable

Serializable Fields

- private String parentName
  - Parent revision of this one, so we can traverse the graph.
- private RevisionData parentRev
  -
- private RevisionElement primitiveData
  - The data itself as sets of RevisionElements. Should probably rather be a map of vectors, each vector with a name defined by the revisioning model plus some rules to manipulate it. At present, the basic (KSI TR 9/2000) SOFA model is wired in.
- private RevisionElement componentData
  - The data itself as sets of RevisionElements. Should probably rather be a map of vectors, each vector with a name defined by the revisioning model plus some rules to manipulate it. At present, the basic (KSI TR 9/2000) SOFA model is wired in.
- private RevisionElement traitData
  - The data itself as sets of RevisionElements. Should probably rather be a map of vectors, each vector with a name defined by the revisioning model plus some rules to manipulate it. At present, the basic (KSI TR 9/2000) SOFA model is wired in.

Constructors

- *RevisionData*
  `public` **RevisionData**`( )`
  - **Usage**
    - Default constructor – creates empty revision data object.

- *RevisionData*
  `public` **RevisionData**`( java.lang.String` **fragment** `)`
  - **Usage**
    - String constructor – fills in component revision data from a string containing the revision part of the URI component name (the "rev=1.2.3" or just the "1.2.3").

- *RevisionData*
  `public` **RevisionData**`( java.lang.String` **fqvParentName,**
  `sofa.vers.RevisionElement []` **primElems,** `sofa.vers.RevisionElement []`

  - **Usage**
    * Primary constructor that fills the revision data from the data arrays.

METHODS

- *compareTo*
`public int` **compareTo**`( java.lang.Object` **o** `)`
  - **Usage**
    * The comparison of two revisions is done by comparing the corresponding componentrevision data and denotes purely the historical order (does not attempt to compare for compatibility).
  - **Returns** - VersionAccess.PRECEDES/VersionAccess.EQUALS/VersionAccess.FOLLOWS if this revision precedes/equals/follows the otherRevision
  - **See Also**
    * `java.lang.Comparable`

- *getElements*
`public RevisionElement` **getElements**`( int` **level** `)`
  - **Returns** - the set of RevisionElements at the given level.

- *getParentName*
`public String` **getParentName**`( )`
  - **Returns** - The parent component name

- *getParentRev*
`public RevisionData` **getParentRev**`( )`
  - **Returns** - Reference to the parent RevisionData object

---

- *getRevID*
`public String` **getRevID**`( int` **typeOfID** `)`
  - **Usage**
    * Returns the desired form of the revision ID, computed from the "raw" data held by this object. See the technical report 9/2000. This is the place where the ordering of specification parts comes to action!
  - **Returns** - String with revision ID, `null` if REV_TRAIT is requested (no revision ID format is defined for trait level).

- *setParent*
`public void` **setParent**`( java.lang.String` **aName**`, sofa.vers.RevisionData` **aRev** `)`
  - **Usage**
    * Sets the revision object of the immediate predecessor in the revision graph. Should probably compute/check the higher-level data from the trait-level data and parent

- *toString*
  `public String` **toString**`( )`
  - **Returns** - The default string form is the component revision ID. See the technical report 9/2000.

7.2.3 Class **RevisionElement**

Holds one item in any revision data, usually of one trait.

Declaration

public class RevisionElement **extends** java.lang.Object **implements** java.lang.Comparable, java.io.Serializable

Fields

- public static final int CT_ANY
  –
- public static final int CT_NA
  –
- public static final int CT_INIT
  –
- public static final int CT_NONE
  –
- public static final int CT_SPEC
  –
- public static final int CT_GEN
  –
- public static final int CT_MUT
  –

CONSTRUCTORS

- *RevisionElement*
  `public` **RevisionElement**`( int` **aValue** `)`
  - **Usage**
    - Convenience constructor as the last resort for the lazy ones.

- *RevisionElement*
  `public` **RevisionElement**`( java.lang.String` **aName**`, int` **aValue**`, int` **aChange** `)`
  - **Usage**
    - Constructor – fill in the element data

METHODS

- *compareTo*
  `public int` **compareTo**`( java.lang.Object` **o** `)`
  - **Usage**
    - Compares this rev data to otherElement using '<' on the rev number, that is uses the historical ordering for comparison. (Does not attempt comparison for compatibility which would need the change value.)
  - **See Also**
    - `java.lang.Comparable`

- *getChange*
  `public int` **getChange**`( )`

- *getChangeString*
  `public static final String` **getChangeString**`( int` **change** `)`
  - **Usage**
    - Converts change indication value into a human-readable string form.

- *getChangeValue*
  `public static final int` **getChangeValue**`( java.lang.String` **change** `)`
  - **Usage**
    - Converts change indication string representation to the numerical value.

- *getName*
  `public String` **getName**`( )`

- *getValue*
  `public int` **getValue**`( )`

- *toString*
  `public String` **toString**`( )`
  - **Usage**

7.2.4 Class **VariantData**

The variant part of the version data. Loosely based on Gergi's model plus feature logic.

Declaration

public class VariantData
extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable

Serializable Fields

- private Vector keys
  - Scalar attributes
- private Vector dims
  - Hierarchical attributes

Fields

- public static final int AND
  - types of variant expr
- public static final int BOOLEAN
  -
- public static final int FEATURE
  -

Constructors

- *VariantData*
  `public` **VariantData**`( )`
  - **Usage**
    - Default constructor – creates an empty variant expression

---

- *VariantData*
  `public` **VariantData**`( java.lang.String` **fragment** `)`
  - **Usage**
    - String constructor – fills in the data by parsing the string containing the variant part of component URI name. The string is assumed to contain a boolean conjunctive expression. TODO: feature logic with OR and parenteses

- *VariantData*
  `public` **VariantData**`( sofa.vers.VariantElement [] newKeys, sofa.vers.VariantElement [] newDims )`
  - **Usage**
    - A stub constructor for creating the conjunctive list.

METHODS

- *compareTo*
  `public int` **compareTo**`( java.lang.Object  o )`
  - **Usage**
    - The comparison of two variants is done by comparing the constituent sets of variant elements.
  - **Returns** - VersionAccess.SUBSET/VersionAccess.EQUALS/VersionAccess.SUPERSET if this variant is-contained-in/equals/contains the otherVariant
  - **See Also**
    - `java.lang.Comparable`
- *getType*
  `public int` **getType**`( )`
  - **Returns** - the type of this variant expression; only AND (boolean conjunctive terms) can be handled at present.
- *toString*
  `public String` **toString**`( )`
  - **Returns** - the URI fragment with the variant identification, after the 'var='. Should be independent of the order of key and dim values as they were added to the expression, i.e. should generate a "normalized" variant string.

7.2.5 CLASS **VariantElement**

Contains data of one variant attribute, scalar or hierarchical.

DECLARATION

public class VariantElement
extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable

Serializable Fields

- private String name
  –
- private String value
  –
- private boolean hierarchical
  – TRUE if the element is a hierarchical attribute

Fields

- public static final int HIER_DELIMITER
  – Delimiter of the hierarchical attribute levels

Constructors

- *VariantElement*
  `public` **VariantElement**`( java.lang.String` **aName**`, java.lang.String` **aValue** `)`
  – **Usage**
    * Constructor, fills in the data and parses the aValue to see whether it is a hierarchical attribute.

Methods

- *compareTo*
  `public int` **compareTo**`( java.lang.Object` **o** `)`
  – **Usage**
    * Compares this variant element with otherElement. The two elements must have the same <tt>name</tt>part. Comparison of ordinal elements is based on lexicographical comparison of their <tt>value</tt>parts. TODO: Comparison of hierarchical elements should use the "precedes" relation defined on dimension values in the version database; because the database is not implemented at the time of writing, the comparison is textual, using lexicographical comparison of the <tt>value</tt>.
  – **See Also**
    * `java.lang.Comparable`
    * `sofa.vers.VersionAccess` (in 7.1.1, page 51)

- *getName*

- *getValue*
  `public String` **getValue**`( )`
- *isHierarchical*
  `public boolean` **isHierarchical**`( )`
- *toString*
  `public String` **toString**`( )`

7.2.6 Class **VersionData**

Holds version data (revision+variant) of one component. See KSI TR 9/2000 and Notes on implementation.

Declaration

public class VersionData
extends java.lang.Object
implements java.io.Serializable, VersionAccess

Serializable Fields

- private String tag
  –
- private String branch
  –
- private RevisionData revision
  –
- private VariantData variant
  –

Constructors

- *VersionData*
  `public` **VersionData**`( )`
  - **Usage**
    * Default constructor creates empty revision and variant objects.
- *VersionData*
  `public` **VersionData**`( java.lang.String` **fragment** `)`
  - **Usage**
    * This constructs the version data object from a URI fragment of the component

---

- *VersionData*
  `public` **VersionData**`( java.lang.String` **aTag**`, java.lang.String` **aBranch**`, sofa.vers.RevisionData` **aRev**`, sofa.vers.VariantData` **aVar** `)`
  - **Usage**
    - This constructs the version data object from pieces.

METHODS

- *compareTo*
  `public int` **compareTo**`( java.lang.Object` **o** `)`
  - **Usage**
    - Compares two version data structures. They must denote the same branch; then revision comparison takes precedence over variant comparison (i.e. 1.2.3 <1.3.4 regardless of variant properties; this is doubtful but left as is for simplicity). Ignores the version tags.
  - **See Also**
    - `java.lang.Comparable`
    - `sofa.vers.RevisionData.compareTo`
    - `sofa.vers.VariantData.compareTo`

- *getBranchID*
  `public String` **getBranchID**`( )`
  - **Returns** - the branch name

- *getBranchString*
  `public String` **getBranchString**`( )`

- *getRevID*
  `public String` **getRevID**`( )`
  - **Returns** - the REV_COMPONENT revision string

- *getRevID*
  `public String` **getRevID**`( int` **level** `)`
  - **Returns** - the revision string corresponding to the requested level.

- *getRevString*
  `public String` **getRevString**`( )`

- *getTag*
  `public String` **getTag**`( )`
  - **Returns** - the version tag

- *getVarID*
  `public String` **getVarID**`( )`

- *getVarString*
  `public String` **getVarString**`( )`

- *normalizeComparisonValue*
  `public static final int` **normalizeComparisonValue**`( int` **value** `)`

  – **Usage**
    ∗ Utility method that converts an integer value to a version comparison canonical value.

- *toString*
  `public String` **toString**`( )`

  – **Returns** - The URI fragment with version information

7.2.7 Class **VersionIncomparableException**

This exception is thrown when attempting to compare two revisions, variants or whole version data that are incomparable (e.g. revisions of different types, variants of different dimensions, etc.).

Declaration

public class VersionIncomparableException
extends java.lang.Exception

Chapter 8

Package sofa.application

Package Contents *Page*

Classes

This package is used for SOFA architecture development and testing. Currently it contains also a hard-wired SOFA user and administration interface.

Gradually, the SOFA framework would split into several independently executable modules but for now there is this monolitic application (although internally modular). the only exception is the Template Repository, that is self-executable and interconnected via Java RMI.

The application creates a window with several buttons - one per an implemented SOFAnode part. Each button opens a manager's window of corresponding SOFA subsystem.

8.1 Classes

8.1.1 Class **ApplicationMainFrame**

Visual interface to SOFA test application. Creates one button per each implemented SOFAnode part. By pressing a button a window from corresponding package is dispalyed. Uses Globals for references.

Declaration

public class ApplicationMainFrame
extends javax.swing.JFrame

Serializable Fields

- private Globals fglobals

  –

Constructors

- *ApplicationMainFrame*
  `public` **ApplicationMainFrame**`( sofa.application.Globals` **aglobals** `)`

Methods

- *processWindowEvent*
  `protected void` **processWindowEvent**`( java.awt.event.WindowEvent` **e** `)`

8.1.2 Class **ApplicationMainFrameNew**

OBSOLETE.

Declaration

public class ApplicationMainFrameNew
extends javax.swing.JFrame

Constructors

- *ApplicationMainFrameNew*
  `public` **ApplicationMainFrameNew**`( )`

8.1.3 Class **DemoApplication**

Declaration

public class DemoApplication
extends java.lang.Object

Fields

- public Globals globals
  –

Constructors

- *DemoApplication*
  `public` **DemoApplication**`( )`
  - **Usage**
    * THE MAIN CLASS OF THIS EXPERIMENTAL SOFA IMPLEMENTATION. It acrs as a loader of the SOFA framework. It instantiates implemented pars of the SOFAnode, cares about initial setup if necessary. Any visible results are product of ApplicationMainFrame class, that is called by this object.
  - **See Also**
    * `sofa.application.ApplicationMainFrame` (in 8.1.1, page 65)

Methods

- *main*
  `public static void` **main**`( java.lang.String [] args )`

8.1.4 Class **Globals**

Utility class, that contains references to the main modules of this SOFA application. There are no global variables in Java, so a static class has to be used instead. How nice!

Declaration

public class Globals
extends java.lang.Object

Fields

- public static TemplateRepository tr
  - reference to Template Repository module
- public static TemplateRepositoryFrame trFrame
  - reference to Template Repository window
- public static DebuggerFrame dbFrame
  - reference to Debugger window
- public static RunPart rp
  - reference to Run Part
- public static RunPartFrame rpFrame
  - reference to Run Part window
- public static Shell sh
  - reference to User Shell
- public static ShellFrame shFrame
  - reference to User Shell window

Constructors

- *Globals*
  `public` **Globals**`( )`

Chapter 9

Package sofa.node.repository.shell

Package Contents *Page*

Classes

Administrator's GUI into the Template Repository. This package displays a window that allows to create, delete, list, filter etc. components in several ways.

9.1 Classes

9.1.1 Class **Shell**

Window application to visualize some choosen functions of TR.

Declaration

public class Shell
extends java.lang.Object

Constructors

- *Shell*
  `public` **Shell**`( )`

Methods

- *badUsage*
  `public static void` **badUsage**`( )`
- *main*
  `public static void` **main**`( java.lang.String [] args )`

9.1.2 Class **ShellFrame**

Main frame of sofa.node.repository.shell class.

Declaration

public class ShellFrame
extends javax.swing.JFrame

Serializable Fields

- private final int LEVELS
  – Count of basic sofaname levels. (provider, component name, version)
- private final String ALL_ELEMENTS
  –
- private final String LABELS

- private final String buttonLabels

  –

- private final boolean modeButtonEnable

  –

Constructors

- *ShellFrame*
  `public` **ShellFrame**`( )`

Methods

- *closeWindow*
  `protected void` **closeWindow**`( )`
- *processWindowEvent*
  `protected void` **processWindowEvent**`( java.awt.event.WindowEvent` **e** `)`
- *refresh*
  `public void` **refresh**`( )`
- *refreshComponentList*
  `protected void` **refreshComponentList**`( )`
- *refreshFilters*
  `protected void` **refreshFilters**`( int` **fromLevel** `)`
- *setLocal*
  `public void` **setLocal**`( sofa.node.repository.Node2TR` **tr** `)`
- *setRemote*
  `public void` **setRemote**`( sofa.node.repository.Node2TR` **tr** `)`
- *showException*
  `protected void` **showException**`( java.lang.Exception` **x** `)`
- *showMessage*
  `public void` **showMessage**`( java.lang.String` **message** `)`

Chapter 10

Package sofa.common

Package Contents *Page*

Classes

Common definitions of some data types and/or classes used among other SOFA classes, for example SOFA IID and so on. This package provides mostly declaration - there is no functionality.

because there are no global variables or constants in java, we use a global class to hold global references and constants that are used widespread in the implementation.

10.1 Classes

10.1.1 Class **Const**

This class contains constats used widespread in this implementation of SOFA framework.

Declaration

public class Const
extends java.lang.Object

Fields

- public static final int CM_STATE_NULL
  - internal state of a component. Used by Component Manager class.
- public static final int CM_STATE_LOADED
  - internal state of a component. Used by Component Manager class.
- public static final int CM_STATE_RUNNING
  - internal state of a component. Used by Component Manager class.
- public static final int CM_STATE_PAUSED
  - internal state of a component. Used by Component Manager class.
- public static final int CM_STATE_EXTERNALIZING
  - internal state of a component. Used by Component Manager class.
- public static final int CM_STATE_UPGRADING
  - internal state of a component. Used by Component Manager class.
- public static final int CM_EXTERNALIZE_LOOP_COUNT
  - How many times we should try to wait for a component to respond for externalization request.
- public static final int CM_DESTROY_LOOP_COUNT
  - How many times we should try to wait for a component to respond for externalization request.

Constructors

- *Const*
  `public` **Const**`( )`

10.1.2 Class **SOFACMID**

Utility class, represents Component Manager's ID within SOFA runtime. In this implementation it is only a string wrapper. No functionality.

Declaration

public class SOFACMID
extends java.lang.Object

Constructors

- *SOFACMID*
  `public` **SOFACMID**`( java.lang.String` **ID** `)`

Methods

- *id*
  `public String` **id**`( )`
- *toString*
  `public String` **toString**`( )`

10.1.3 Class **SOFAComponentID**

Utility class, represents Component ID within SOFA runtime. In this implementation it is only a string wrapper. No functionality.

Declaration

public class SOFAComponentID
extends java.lang.Object

Constructors

- *SOFAComponentID*
  `public` **SOFAComponentID**`( java.lang.String` **ID** `)`

Methods

- *id*
  `public String` **id**`( )`

10.1.4 Class **SOFAIID**

Utility class, represents SOFA Interface ID within the SOFA runtime. In this implementation it is only a string wrapper. No functionality.

Declaration

public class SOFAIID
extends java.lang.Object

Constructors

- *SOFAIID*
  `public` **SOFAIID**`( java.lang.String` **ID** `)`

Methods

- *id*
  `public String` **id**`( )`

10.1.5 Class **SOFAInterfaceRef**

OBSOLETE? Utility class, used for type casting to avoid references to Object class. No functionality.

Declaration

public class SOFAInterfaceRef
extends java.lang.Object

Constructors

- *SOFAInterfaceRef*
  `public` **SOFAInterfaceRef**`( )`

Chapter 11

Package sofa.runpart.classloaders

Package Contents *Page*

Classes

Classloaders used to load SOFA components from JAR files. This is the only way how can the Run Part access "binary images" of SOFA comopnents.

11.1 Classes

11.1.1 Class **SOFAAbstractClassLoader**

An abstract class loader that is capable to load classes specified by URI. Internally uses a URLClassLoader class.

Declaration

public abstract class SOFAAbstractClassLoader
extends java.lang.Object

Constructors

- *SOFAAbstractClassLoader*
  `public` **SOFAAbstractClassLoader**`( )`

Methods

- *createInstance*
  `public Object` **createInstance**`( java.lang.String` **className** `)`
  - **Usage**
    - Creates an instance of a given class. This method is used internally by the JVM when it need to instantiate a "new" class.
  - **Parameters**
    - `className` - full name of the class to make - supplied by JVM
- *getURLString*
  `protected abstract String` **getURLString**`( )`
  - **Usage**
    - no function. Must be overriden.

11.1.2 Class **SOFAFileClassLoader**

SOFA class loader capable to load classes from local JAR file.

Declaration

public class SOFAFileClassLoader
extends sofa.runpart.classloaders.SOFAAbstractClassLoader

Constructors

- *SOFAFileClassLoader*
  `public` **SOFAFileClassLoader**`( java.lang.String` **aFileName** `)`
  - **Usage**
    - Default constructor.
  - **Parameters**
    - `aFileName` - JAR file name, including path and extension

Methods

- *getURLString*
  `protected String` **getURLString**`( )`
  - **Usage**
    - constructs URL from "file://" and the JAR file name

Chapter 12

Package sofa.node.repository.utils

Package Contents *Page*

Classes

An auxiliary package. It contains few helper classes used in Template Reposiory.

12.1 Classes

12.1.1 Class **DirectoryFilter**

This class provides suport for contents listing of directory. Using of this filter ensure that directories will be listed only.

Declaration

public class DirectoryFilter
extends java.lang.Object
implements java.io.FileFilter

Constructors

- *DirectoryFilter*
  `public` **DirectoryFilter**`( )`

Methods

- *accept*
  `public boolean` **accept**`( java.io.File` **pathname** `)`

12.1.2 Class **DistributionPackage**

This class provides different services over DistributionPackage format.

Declaration

public class DistributionPackage
extends java.lang.Object

Fields

- public static final String versionInfo
  - Pathname to file containing list of versions in package

Constructors

- *DistributionPackage*

METHODS

- *getComponents*
  `public static ComponentStatus` **getComponents**`( java.lang.String` **distributionPackagePathname** `)`
  - **Usage**
    - Opens versionInfo file in package and process if. Do not test component for their status so status of all components is set to UNDEFINED.
  - **Returns** - List of all components in package.
  - **Exceptions**
    - `java.io.FileNotFoundException` -
    - `sofa.node.repository.DistributionPackageCorruptedException` -

12.1.3 CLASS JarFileFilter

This class provides suport for contents listing of directory. Using of this filter ensure that JARfiles will be listed only.

DECLARATION

public class JarFileFilter
extends java.lang.Object
implements java.io.FileFilter

CONSTRUCTORS

- *JarFileFilter*
  `public` **JarFileFilter**`( )`

METHODS

- *accept*
  `public boolean` **accept**`( java.io.File` **file** `)`

Chapter 13

Package sofa.node.repository

Package Contents *Page*

Second implementation of Template Repository. Support remote operation via Java RMI and also support proper versioning provided by package sofa.vers.

13.1 Interfaces

13.1.1 Interface **In2TR**

Administrative interface for adding and removing components etc.

Declaration

public interface In2TR **implements** java.rmi.Remote

Methods

- *insertComponent*
  `public ComponentStatus` **insertComponent**`( sofa.node.repository.SerializedData` **distributionPackage**`, sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    - Insert one component from dristribution package.
  - **Parameters**
    - `distributionPackage` - is serialized representation of package.
    - `cd` - is fqvName of component to be inserted.
  - **Returns** - status of insertion.
  - **Exceptions**
    - `sofa.node.repository.DistributionPackageCorruptedException` -

- *insertComponent*
  `public ComponentStatus` **insertComponent**`( java.lang.String` **distributionPackagePathname**`, sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    - Insert one component from dristribution package
  - **Parameters**
    - `distributionPackagePathname` - is pathname to file containing package.
    - `cd` - is fqvName of component to be inserted.
  - **Returns** - status of insertion
  - **Exceptions**
    - `sofa.node.repository.DistributionPackageCorruptedException` -
    - `java.io.FileNotFoundException` -

- *insertPackage*
  `public ComponentStatus` **insertPackage**`( sofa.node.repository.SerializedData` **distributionPackage** `)`
  - **Usage**
    - Insert all components from dristribution package.

    * `distributionPackage` - is serialized representation of package.
  - **Returns** - List of all components with status.
  - **Exceptions**
    * `sofa.node.repository.DistributionPackageCorruptedException` -

- *insertPackage*
  `public ComponentStatus` **insertPackage**`( java.lang.String` **distributionPackagePathname** `)`
  - **Usage**
    * Insert all component from dristribution package.
  - **Parameters**
    * `distributionPackagePathname` - is pathname to file containing package.
  - **Returns** - List of all components with status.
  - **Exceptions**
    * `sofa.node.repository.DistributionPackageCorruptedException` -
    * `java.io.FileNotFoundException` -

- *removeComponent*
  `public void` **removeComponent**`( sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    * Deletes the component from TR.
  - **Parameters**
    * `cd` - is fqvName of component to be removed.
  - **Exceptions**
    * `sofa.node.repository.NoSuchComponentException` -
    * `sofa.node.repository.ComponentInUseException` -

13.1.2 Interface Made2TR

This interface is used by developer tools to update Template Repository.

Declaration

public interface Made2TR
implements java.rmi.Remote

Methods

- *getComponent*
  `public void` **getComponent**`( java.lang.String` **directoryPath**`, sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**

  - **Parameters**
    - `directoryPath` - name of the directory where all components files/directories will be created.
  - **Exceptions**
    - `sofa.node.repository.NoSuchComponentException` -
    - `IOException` -

- *insertComponent*
  `public ComponentStatus` **insertComponent**`( java.lang.String` **directoryPath** `)`
  - **Usage**
    - Inserts component in unpacked form.
  - **Parameters**
    - `directoryPath` - path to directory with all necessary files.
  - **Returns** - status of insertion.
  - **Exceptions**
    - `sofa.node.repository.ComponentCorruptedException` -

- *updateComponent*
  `public ComponentStatus` **updateComponent**`( java.lang.String` **directoryPath** `)`
  - **Usage**
    - Inserts component in unpacked form. Only if component already exists it is overwritten. Otherwise exception is thrown.
  - **Parameters**
    - `directoryPath` - path to directory with all necessary files.
  - **Returns** - status of updation.
  - **Exceptions**
    - `sofa.node.repository.ComponentCorruptedException` -

13.1.3 Interface **Node2TR**

Interface joining all repository parts together.

Declaration

public interface Node2TR
implements In2TR, Out2TR, Run2TR, Made2TR, QueryTR

13.1.4 Interface **Out2TR**

System interface for component trading.

DECLARATION

public interface Out2TR
implements java.rmi.Remote

METHODS

- *getPackage*
  `public SerializedData` **getPackage**`( sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    - Take out distribution package with one component.
  - **Parameters**
    - `cd` - is fqvName of demanded component.
  - **Returns** - Distribution package is serializable representation.
  - **Exceptions**
    - `sofa.node.repository.NoSuchComponentException` -

- *getPackage*
  `public SerializedData` **getPackage**`( sofa.node.repository.ComponentDescriptor` `[]` **cds** `)`
  - **Usage**
    - Take out distribution package with several components.
  - **Parameters**
    - `cds` - is array of fqvNames of demanded components.
  - **Returns** - Distribution package is serializable representation.
  - **Exceptions**
    - `sofa.node.repository.NoSuchComponentException` -

13.1.5 INTERFACE QueryTR

Querying tools. This interface work with ComponentAbstractors only.

DECLARATION

public interface QueryTR
implements java.rmi.Remote

METHODS

- *getAllComponents*

  - **Usage**
    - Return fqNames of all component in the TR.

- *getAllComponents*
  `public ComponentAbstractor` **getAllComponents**`(`
  `sofa.node.repository.ComponentAbstractor` **provider** `)`
  - **Usage**
    - Return fqNames of all components available from one provider.

- *getAllProviders*
  `public ComponentAbstractor` **getAllProviders**`( )`
  - **Usage**
    - Return names of all available providers.

- *getAllVersions*
  `public ComponentAbstractor` **getAllVersions**`( )`
  - **Usage**
    - Return all available versions

- *getAllVersions*
  `public ComponentAbstractor` **getAllVersions**`(`
  `sofa.node.repository.ComponentAbstractor` **fqName** `)`
  - **Usage**
    - Return all available versions of a component.

- *getNewestVersion*
  `public ComponentAbstractor` **getNewestVersion**`(`
  `sofa.node.repository.ComponentAbstractor` **fqName** `)`
  - **Usage**
    - Return the newest version of component

- *isAnyComponentAvailable*
  `public boolean` **isAnyComponentAvailable**`(`
  `sofa.node.repository.ComponentAbstractor []` **versions** `)`
  - **Usage**
    - Return TRUE if finds at least one version satisfying the input.

13.1.6 Interface **ResourceName**

Common resource names.

Declaration

```
public interface ResourceName
```

FIELDS

- public static final String BUILDER_CLASS
  –
- public static final String MANAGER_CLASS
  –
- public static final String MAIN_CLASS
  –
- public static final String DEPLOYMENT_DESCRIPTOR
  –
- public static final String COMPONENT_DECLARATION
  –

13.1.7 INTERFACE **ResourceType**

Supported resource types.

DECLARATION

```
public interface ResourceType
```

FIELDS

- public static final String STRING
  –
- public static final String INTEGER
  –
- public static final String FLOAT
  –
- public static final String FILE
  –
- public static final String CLASS
  –

13.1.8 INTERFACE **Run2TR**

DECLARATION

public interface Run2TR
implements java.rmi.Remote

METHODS

- *getBuilderClass*
  `public SerializedData` **getBuilderClass(**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Usage**
    - Specialized version of getResource() method.

- *getClass*
  `public SerializedData` **getClass(** `java.lang.String` **className,**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Parameters**
    - `className` - is name of class without extension .class
  - **Returns** - class file in serializable representation.
  - **Exceptions**
    - `sofa.node.repository.NoSuchComponentException` -
    - `sofa.node.repository.ComponentCorruptedException` -

- *getComponentDeclaration*
  `public SerializedData` **getComponentDeclaration(**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Usage**
    - Specialized version of getResource() method.

- *getDeploymentDescriptor*
  `public SerializedData` **getDeploymentDescriptor(**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Usage**
    - Specialized version of getResource() method.

- *getJarWithAllClasses*
  `public SerializedData` **getJarWithAllClasses(**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Returns** - JARfile containing all .class files in serializable representation.

- *getMainClass*
  `public SerializedData` **getMainClass(** `sofa.node.repository.ComponentDescriptor` **cd)**

    * Specialized version of getResource() method.

- *getManagerClass*
  `public SerializedData` **getManagerClass(**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Usage**
    * Specialized version of getResource() method.

- *getResource*
  `public Resource` **getResource(** `java.lang.String` **resourceName,**
  `sofa.node.repository.ComponentDescriptor` **cd)**
  - **Parameters**
    * `resourceName` - is name defined in XML description of this component. Common names for 'BuilderClass', etc. are defined in ResourceName (in 13.1.6, page 87)
  - **Returns** - demanded resource.
  - **Exceptions**
    * `sofa.node.repository.NoSuchComponentException` -
    * `sofa.node.repository.ComponentCorruptedException` -

13.2 Classes

13.2.1 Class **ComponentAbstractor**

Object containing not-fully-qualified information about component and that is why this object may be used for querying purposes only. Because functionality of this object can influence working of TR it's marked as final to not allow anybody to overwrite it.

Declaration

public final class ComponentAbstractor
extends java.lang.Object
implements java.io.Serializable

Serializable Fields

- private String provider
  - Provider name
- private String fullName
  - Full name including levels separated by '/'
- private String version
  - String representation of version informations

CONSTRUCTORS

- *ComponentAbstractor*
  `public` **ComponentAbstractor**`( )`
  - **Usage**
    - Creates empty ComponentAbstractor.

- *ComponentAbstractor*
  `public` **ComponentAbstractor**`( sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    - Creates ComponentAbstractor and inicializes local variables with values taken from ComponentDescriptor.

- *ComponentAbstractor*
  `public` **ComponentAbstractor**`( java.lang.String` **provider**`, java.lang.String` **fullName**`, java.lang.String` **version** `)`
  - **Usage**
    - Creates ComponentAbstractor and inicializes local variables with given values.

METHODS

- *getFullName*
  `public String` **getFullName**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - full name.
  - **Exceptions**
    - `sofa.node.repository.IncorrectUseException` - if full name was not set yet.

- *getFullName*
  `public String` **getFullName**`( char` **separator** `)`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - full name with given separator.
  - **Exceptions**
    - `sofa.node.repository.IncorrectUseException` - if full name was not set yet.

- *getNames*
  `public String` **getNames**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - Array of level names. Main name of component is at the end.

        * `sofa.node.repository.IncorrectUseException` - if full name was not set yet.

- *getProvider*
  `public String` **getProvider**`( )`
  - **Usage**
    * Authorized access to private variable.
  - **Returns** - name of provider.
  - **Exceptions**
    * `sofa.node.repository.IncorrectUseException` - if provider was not set yet.

- *getVersion*
  `public String` **getVersion**`( )`
  - **Usage**
    * Authorized access to private variable.
  - **Returns** - version.
  - **Exceptions**
    * `sofa.node.repository.IncorrectUseException` - if version was not set yet.

- *setFullName*
  `public void` **setFullName**`( java.lang.String` **fullName** `)`
  - **Usage**
    * This method sets full name to given value.

- *setProvider*
  `public void` **setProvider**`( java.lang.String` **provider** `)`
  - **Usage**
    * This method sets provider name to given value.

- *setVersion*
  `public void` **setVersion**`( java.lang.String` **version** `)`
  - **Usage**
    * This method sets version to given value.

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    * Return String representation of this object

13.2.2 Class **ComponentAlreadyPresentException**

Thrown when application tries to insert component into TR but the same component (same provider, same name, same version, same revision) is already present in TR.

DECLARATION

public class ComponentAlreadyPresentException
extends sofa.node.repository.TRException

CONSTRUCTORS

- *ComponentAlreadyPresentException*
  `public` **ComponentAlreadyPresentException**`( java.lang.String` **fqvName** `)`

13.2.3 CLASS ComponentCorruptedException

Thrown when TR is to use Component which is incomplete or some of its parts is not in proper state.

DECLARATION

public class ComponentCorruptedException
extends sofa.node.repository.TRException

CONSTRUCTORS

- *ComponentCorruptedException*
  `public` **ComponentCorruptedException**`( java.lang.String` **description**`, java.lang.String` **fqvName** `)`

13.2.4 CLASS ComponentDescriptor

Object containing fully-qualified information about component. Using of this class is the only way to access components in TR. Because functionality of this object can influence working of TR it's marked as final to not allow anybody to overwrite it.

DECLARATION

public final class ComponentDescriptor
extends java.lang.Object
implements java.io.Serializable

SERIALIZABLE FIELDS

- private String provider

- private String names
  - Level names which are separated by '/' in sofaname
- private VersionData version
  - Object containing version informations

CONSTRUCTORS

- *ComponentDescriptor*
  `public` **ComponentDescriptor**`( sofa.node.repository.ComponentAbstractor` **ca** `)`
  - **Usage**
    - Create object from ComponentAbstractor which has all informations.

- *ComponentDescriptor*
  `public` **ComponentDescriptor**`( java.lang.String` **fqvName** `)`
  - **Usage**
    - Creates object from fqvName.

METHODS

- *equals*
  `public boolean` **equals**`( java.lang.Object` **obj** `)`
  - **Usage**
    - Test wheter this ComponentDescriptor describes the same component-variant as given ComponentDescriptor

---

- *getFullName*
  `public String` **getFullName**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - full name of component including levels separated by '/'

- *getFullName*
  `public String` **getFullName**`( java.lang.String` **separator** `)`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - full name of component including levels separated by 'separator'

- *getName*
  `public String` **getName**`( )`
  - **Usage**
    - Authorized access to private variable.

- *getNames*
  `public String` **getNames**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - Array of level names. Main name of component is at the end.

- *getProvider*
  `public String` **getProvider**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - name of provider.

- *getVersion*
  `public VersionAccess` **getVersion**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - Object containing version informations.

- *main*
  `public static void` **main**`( java.lang.String [] args )`
  - **Usage**
    - Basic test of functionality of this class

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    - Return String representation of this object

13.2.5 Class ComponentInUseException

Thrown when TR is to 'action' component which is currently in use.

Declaration

public class ComponentInUseException
extends sofa.node.repository.TRException

Constructors

- *ComponentInUseException*
  `public` **ComponentInUseException**`( java.lang.String action,`

13.2.6 Class **ComponentStatus**

This method is structure containing ComponentDescriptor and status of component it describes.

Declaration

public class ComponentStatus
extends java.lang.Object
implements java.io.Serializable

Serializable Fields

- private ComponentDescriptor cd
  - fqvName of component
- private int status
  - status of component
- private String comment
  - comment

Fields

- public static final int UNDEFINED
  -
- public static final int OK
  -
- public static final int ALREADY_PRESENT
  -
- public static final int CORRUPTED
  -
- public static final int NO_SUCH_COMPONENT
  -

CONSTRUCTORS

- *ComponentStatus*
  `public` **ComponentStatus**`( sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    - Creates ComponentStatus for given component with UNDEFINED status.

- *ComponentStatus*
  `public` **ComponentStatus**`( sofa.node.repository.ComponentDescriptor` **cd**`, int` **status** `)`
  - **Usage**
    - Creates ComponentStatus for given component with given status.

- *ComponentStatus*
  `public` **ComponentStatus**`( sofa.node.repository.ComponentDescriptor` **cd**`, int` **status**`, java.lang.String` **comment** `)`
  - **Usage**
    - Creates ComponentStatus for given component with given status and comment.

METHODS

- *getComment*
  `public String` **getComment**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - comment of status of component.

- *getName*
  `public ComponentDescriptor` **getName**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - fqvName of component.

- *getStatus*
  `public int` **getStatus**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - status of component.

- *setComment*
  `public void` **setComment**`( java.lang.String` **comment** `)`
  - **Usage**
    - Authorized access to private variable. Comment of status of component is changed

- *setStatus*
  `public void` **setStatus**`( int` **status** `)`
  - **Usage**
    - Authorized access to private variable. Status of component is changed to given new status.

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    - Return String representation of Object

13.2.7 Class DistributionPackageCorruptedException

Thrown when TR is to use DistributionPackage which is incomplete or some of its parts are not in proper state.

Declaration

public class DistributionPackageCorruptedException
extends sofa.node.repository.TRException

Constructors

- *DistributionPackageCorruptedException*
  `public` **DistributionPackageCorruptedException**`( java.lang.String` **description** `)`

13.2.8 Class IncorrectUseException

Thrown when application tries to ComponentAbstractor incorrectly. For example using as parametr for method getAllComponents(provider) and provider of type ComponentAbstractor does not contain provider information.

Declaration

public class IncorrectUseException
extends sofa.node.repository.TRException

Constructors

- *IncorrectUseException*

13.2.9 Class **InternalException**

Cast when TR run into state which was unexpected during coding.

Declaration

public class InternalException
extends java.lang.RuntimeException

Constructors

- *InternalException*
  `public` **InternalException**`( java.lang.String` **description** `)`

13.2.10 Class **NoSuchComponentException**

Thrown when application tries to access component specified by fully-qualified name but there is no such component in TR.

Declaration

public class NoSuchComponentException
extends sofa.node.repository.TRException

Constructors

- *NoSuchComponentException*
  `public` **NoSuchComponentException**`( java.lang.String` **fqvName** `)`

13.2.11 Class **NotValidNameException**

Thrown when application tries to inicialize ComponentDescriptor with fqvName which is not valid SOFA element name.

Declaration

public class NotValidNameException
extends sofa.node.repository.TRException

CONSTRUCTORS

- *NotValidNameException*
  `public` **NotValidNameException**`( java.lang.String` **fqvName** `)`

13.2.12 CLASS Resource

Class encapsulating resource. Mime-type is the only one which may be null. Value is java.lang.Object and it depends on application how the value will be handled.

DECLARATION

public class Resource
extends java.lang.Object
implements ResourceType, ResourceName, java.io.Serializable

SERIALIZABLE FIELDS

- private String name
  - Name of resource
- private String type
  - Type of resource
- private String mime
  - Mime-type of resource
- private Object value
  - Value of resource

CONSTRUCTORS

- *Resource*
  `public` **Resource**`( )`
  - **Usage**
    - Creates new empty Resource. Resource must be then initialized by method setXXX before being used.

- *Resource*
  `public` **Resource**`( sofa.node.repository.Resource` **resource** `)`
  - **Usage**
    - Creates new resource from another resource.

METHODS

- *getMime*
  `public String` **getMime**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - mime-type of resource.

- *getName*
  `public String` **getName**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - name of resource.

- *getType*
  `public String` **getType**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - type of resource.

- *getValue*
  `public Object` **getValue**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - value of resource.

- *isType*
  `public boolean` **isType**`( java.lang.String` **type** `)`
  - **Usage**
    - Test whether resource is of given type; type is not case-sensitive

---

- *setMime*
  `public String` **setMime**`( java.lang.String` **mime** `)`
  - **Usage**
    - Sets resources mime-type. If the mime-type was previously set mime-type will be not changed. It may be set to null.
  - **Returns** - new mime-type of resource.

- *setName*
  `public String` **setName**`( java.lang.String` **name** `)`
  - **Usage**
    - Sets resources name. If the name was previously set name will be not changed.
  - **Returns** - new name of resource.

        * `NullPointerException` - when parametr 'name' is null.

- *setType*
  `public String` **setType**`( java.lang.String` **type** `)`
  - **Usage**
    * Sets resources type. If the type was previously set type will be not changed.
  - **Returns** - new type of resource.
  - **Exceptions**
    * `NullPointerException` - when parametr 'type' is null.

- *setValue*
  `public Object` **setValue**`( java.lang.Object` **value** `)`
  - **Usage**
    * Sets resources value. If the value was previously set value will be not changed.
  - **Returns** - New value of resource.
  - **Exceptions**
    * `NullPointerException` - when parametr 'value' is null.

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    * Return String representation of object.

13.2.13 Class **SerializedData**

This class is used as return type and input parameter of methods because no standard Stream implements interface Serializable which is necessary for RMI.

Declaration

public class SerializedData
extends java.lang.Object
implements java.io.Serializable

Serializable Fields

- private byte data
  - array of bytes; stored information

CONSTRUCTORS

- *SerializedData*
  `public` **SerializedData**`( byte [] byteArray )`
  - **Usage**
    - This constructor simply store datas in byteArray in local variable.

- *SerializedData*
  `public` **SerializedData**`( java.io.InputStream` **inputStream** `)`
  - **Usage**
    - This constructor reads at once datas from inputStream. Amount of datas to read is retreived by method InputStream.available().

- *SerializedData*
  `public` **SerializedData**`( java.io.InputStream` **inputStream**`, int` **length** `)`
  - **Usage**
    - This constructor reads datas from inputStream. Amount of datas to read is specifiead in parameter 'length'.

METHODS

- *getByteArray*
  `public byte` **getByteArray**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - shallow copy of local array of bytes.

- *getByteArrayInputStream*
  `public ByteArrayInputStream` **getByteArrayInputStream**`( )`
  - **Usage**
    - Authorized access to private variable.
  - **Returns** - ByteArrayInputStream with buffer set to local array of bytes.

- *toString*
  `public String` **toString**`( )`
  - **Usage**
    - Return String representation of datas.

- *writeToFile*
  `public void` **writeToFile**`( java.lang.String` **pathname** `)`
  - **Usage**
    - Writes stream contents in file.

13.2.14 Class **TemplateRepository**

Main class of package.

Declaration

public class TemplateRepository
extends java.rmi.server.UnicastRemoteObject
implements Node2TR

Serializable Fields

- private RepositoryHandler handler
  –
- private InPart in
  –
- private OutPart out
  –
- private MadePart made
  –
- private RunPart run
  –
- private QueryPart query
  –

Fields

- public static PrintStream log
  –

Constructors

- *TemplateRepository*
  `public` **TemplateRepository**`( java.lang.String` **repositoryLocation** `)`
- *TemplateRepository*
  `public` **TemplateRepository**`( java.lang.String` **repositoryLocation**`, int` **bufferSize** `)`

METHODS

- *finalize*
  `public void` **finalize**`( )`
- *getAllComponents*
  `public ComponentAbstractor` **getAllComponents**`( )`
- *getAllComponents*
  `public ComponentAbstractor` **getAllComponents**`(`
  `sofa.node.repository.ComponentAbstractor` **provider** `)`
- *getAllProviders*
  `public ComponentAbstractor` **getAllProviders**`( )`
- *getAllVersions*
  `public ComponentAbstractor` **getAllVersions**`( )`
- *getAllVersions*
  `public ComponentAbstractor` **getAllVersions**`(`
  `sofa.node.repository.ComponentAbstractor` **fqName** `)`
- *getBuilderClass*
  `public SerializedData` **getBuilderClass**`(`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`
- *getClass*
  `public SerializedData` **getClass**`( java.lang.String` **className**`,`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`
- *getComponent*
  `public void` **getComponent**`( java.lang.String` **directoryPath**`,`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`
- *getComponentDeclaration*
  `public SerializedData` **getComponentDeclaration**`(`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`
- *getDeploymentDescriptor*
  `public SerializedData` **getDeploymentDescriptor**`(`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`
- *getJarWithAllClasses*
  `public SerializedData` **getJarWithAllClasses**`(`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`
- *getMainClass*
  `public SerializedData` **getMainClass**`( sofa.node.repository.ComponentDescriptor`
  cd `)`
- *getManagerClass*
  `public SerializedData` **getManagerClass**`(`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`

- *getNewestVersion*
  `public ComponentAbstractor` **getNewestVersion**`(`
  `sofa.node.repository.ComponentAbstractor` **fqName** `)`

- *getPackage*
  `public SerializedData` **getPackage**`( sofa.node.repository.ComponentDescriptor`
  cd `)`

- *getPackage*
  `public SerializedData` **getPackage**`( sofa.node.repository.ComponentDescriptor`
  `[]` **cds** `)`

- *getResource*
  `public Resource` **getResource**`( java.lang.String` **resourceName**`,`
  `sofa.node.repository.ComponentDescriptor` **cd** `)`

- *insertComponent*
  `public ComponentStatus` **insertComponent**`( sofa.node.repository.SerializedData`
  distributionPackage`, sofa.node.repository.ComponentDescriptor` **cd** `)`

- *insertComponent*
  `public ComponentStatus` **insertComponent**`( java.lang.String` **directoryPath** `)`

- *insertComponent*
  `public ComponentStatus` **insertComponent**`( java.lang.String`
  distributionPackagePathname`, sofa.node.repository.ComponentDescriptor` **cd** `)`

- *insertPackage*
  `public ComponentStatus` **insertPackage**`( sofa.node.repository.SerializedData`
  distributionPackage `)`

- *insertPackage*
  `public ComponentStatus` **insertPackage**`( java.lang.String`
  distributionPackagePathname `)`

- *isAnyComponentAvailable*
  `public boolean` **isAnyComponentAvailable**`(`
  `sofa.node.repository.ComponentAbstractor []` **versions** `)`

- *main*
  `public static void` **main**`( java.lang.String [] args )`

- *removeComponent*
  `public void` **removeComponent**`( sofa.node.repository.ComponentDescriptor` **cd**
  `)`

- *updateComponent*
  `public ComponentStatus` **updateComponent**`( java.lang.String` **directoryPath** `)`

- *usage*
  `public static void` **usage**`( )`

13.2.15 Class **TRException**

General exception for Template Repository usage. This exception is an ancestor for all exceptions in this package. Parametr 'description' shall contain explanation of specific exception. If any user of this package dont want to catch specific TR exception he can simply catch this exception and display explanation.

Declaration

public class TRException
extends java.lang.Exception

Constructors

- *TRException*
  `public` **TRException**`( java.lang.String` **description** `)`

Chapter 14

Package sofa.debug

Package Contents *Page*

Auxiliary debugging interface. For internal purpose only.

This package implementes a debugging interface that is used to catch internal debug messages of the SOFA framework. There is also a user interface that display caught messages for the operator. Here you can see how components are loaded, executed and supervise their communication.

14.1 Interfaces

14.1.1 Interface **ISOFADebug**

Auxiliary debugging interface.

Declaration

public interface ISOFADebug

Methods

- *debug*
  `public void` **debug**`( java.lang.Object` **sender**`, java.lang.String` **debugText** `)`

  - **Usage**
    - Sends specified string to a debug subsytem. The string can be futher processed by the subsystem.
  - **Parameters**
    - `sender` - reference to the object that creates the message
    - `debugText` - string with debugging information. Format as you wish.

14.2 Classes

14.2.1 Class **DebuggerFrame**

Creates a window that is used by the test application.

Declaration

public class DebuggerFrame **extends** javax.swing.JFrame **implements** ISOFADebug

Serializable Fields

- private Globals fglobals

  –

Constructors

- *DebuggerFrame*
  `public` **DebuggerFrame**`( sofa.application.Globals` **gl** `)`

Methods

- *debug*
  `public void` **debug**`( java.lang.Object` **sender**`, java.lang.String` **debugText** `)`

Chapter 15

Package sofa.repository

Package Contents *Page*

Classes

OBSOLETE! First implementation of Template Repository. A rather simple version that allow only a basic functionality. See package sofa.node.repository for new implementation based on RMI and with better versioning support.

15.1 Classes

15.1.1 Class **ComponentDescriptor**

OBSOLETE. This class defines a standard way how to describe components by their producers, names and versions. Detailed description is not available.

Declaration

public class ComponentDescriptor
extends java.lang.Object

Constructors

- *ComponentDescriptor*
  `public` **ComponentDescriptor**`( java.lang.String` **strComponentID**`, java.lang.String` **producer**`, java.lang.String` **componentName**`, java.lang.String` **version** `)`

Methods

- *getComponentID*
  `public SOFAComponentID` **getComponentID**`( )`
- *getComponentName*
  `public String` **getComponentName**`( )`
- *getProducer*
  `public String` **getProducer**`( )`
- *getVersion*
  `public String` **getVersion**`( )`
- *toString*
  `public String` **toString**`( )`

15.1.2 Class **ComponentDescriptorList**

OBSOLETE. Auxiliary list of component descriptors. Detailed description is not available.

Declaration

public class ComponentDescriptorList
extends java.util.Vector

CONSTRUCTORS

- *ComponentDescriptorList*
  `public` **ComponentDescriptorList**`( )`

METHODS

- *addDescriptor*
  `public boolean` **addDescriptor**`( sofa.repository.ComponentDescriptor` **desc** `)`
- *getDescriptor*
  `public ComponentDescriptor` **getDescriptor**`( int` **index** `)`

15.1.3 CLASS TemplateRepository

OBSOLETE. Template Repository base class. This implementation stores components in a directory tree on a local filesystem. Component vendors, names and versions are mapped directly to directory/file names. No detailed description is available.

DECLARATION

public class TemplateRepository
extends java.lang.Object
implements sofa.interfaces.ITR2RP, sofa.interfaces.ITR2Sh

CONSTRUCTORS

- *TemplateRepository*
  `public` **TemplateRepository**`( )`

METHODS

- *getComponentList*
  `public ComponentDescriptorList` **getComponentList**`( )`
  - **Usage**
    - Gets a list of stored components.
- *getStreamWithBinaryImageOfComponent*
  `public InputStream` **getStreamWithBinaryImageOfComponent**`(`
  `sofa.common.SOFAComponentID` **compID** `)`

15.1.4 CLASS TemplateRepositoryFrame

OBSOLETE. Administrator's GUI to the Template Repository with very limited functionality. No detailed

DECLARATION

public class TemplateRepositoryFrame
extends javax.swing.JFrame

SERIALIZABLE FIELDS

- private Globals fglobals
  –

CONSTRUCTORS

- *TemplateRepositoryFrame*
  `public` **TemplateRepositoryFrame**`( sofa.application.Globals` **gl** `)`

METHODS

- *displayAvailableComponents*
  `public void` **displayAvailableComponents**`( )`
- *processWindowEvent*
  `protected void` **processWindowEvent**`( java.awt.event.WindowEvent` **e** `)`

15.1.5 CLASS **TRList**

OBSOLETE. Auxiliary class that creates a list of component descriptiors for all components in the Template Repository.

DECLARATION

public class TRList
extends java.lang.Object
implements sofa.interfaces.ITR2Sh

CONSTRUCTORS

- *TRList*
  `public` **TRList**`( )`

METHODS

- *getComponentList*
  `public ComponentDescriptorList` **getComponentList**`( )`

Chapter 16

Package sofa.runpart

Package Contents *Page*

Classes

Performs SOFA Run Part functionality. This package contains all classes of the Run Part that are not used by other packages. The package contain also a graphical user interface.

The runpart is technically a top-level Component Manager. Thus there is not much functionality in this package - the fundamental parts are located in Component Manager and Root Object classes.

16.1 Classes

16.1.1 Class **RunPart**

Provides fundamental functionality of the Run Part. Manages a list of registered Component Manangers. Loads and updates applications or components.

Declaration

public class RunPart
extends java.lang.Object
implements sofa.interfaces.ICM2RP, sofa.interfaces.IRP2Sh, sofa.interfaces.IRP2CB, sofa.interfaces.ICM2CM

Fields

- public final String BASEDIR
  –

Constructors

- *RunPart*
  `public` **RunPart**`( )`
  - **Usage**
    * is default constructor - no action.

Methods

- *getCMList*
  `public SOFAComponentManagerList` **getCMList**`( )`
  - **Usage**
    * Reurns the list of subordinate components. Again, the Run Part is technically a top-level compoennt manager.

- *getInfo*
  `public String` **getInfo**`( )`
  - **Usage**
    * not used. Must be implemented, comes from interface ICM2CM.

- *getRPRef*
  `public IRP2CB` **getRPRef**`( )`

  * Returns reference to the RunPart. Technically, the Run Part equals to a top-level component manager. In this case it returns self.

- *loadApplication*
  `public ICManager` **loadApplication**`( sofa.node.repository.Node2TR` **TRref,** `sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    * loads an application from template repository. This method finds out a name of corresponding component manager, gets a .JAR file with binary image of specified component, and instattiates component manager using a new class loader. Then it sets appropriate CMID and calls init() method of the component manager.
  - **Parameters**
    * `TRref` - a reference to the 'new' Template Repository (connected via RMI)
    * `cd` - specification of the "aplication" component to load

---

- *makeComponent*
  `public ICManager` **makeComponent**`( sofa.interfaces.ICManager` **parent,** `java.lang.String` **CMID,** `java.lang.String` **producer,** `java.lang.String` **componentName,** `java.lang.String` **version** `)`
  - **Usage**
    * instantiates a component from Template Repository. This method is called indirectly by Component Builders when they need to instatntiate a subcomponent. A CB provides description of the subcomponent (that is hard-wired into the builder by CDL compoler) in three strings that are processed into a component descriptor. Technically, this method is similar to loadApplication() method
  - **Parameters**
    * `parent` - reference to parent component manager, i.e. the component that call this method
    * `CMID` - unique identification of the new component. Used for registration of its component manager.
    * `producer` - used to build-up a component descriptor
    * `complonentName` - used to buid-up a component descriptor
    * `version` - used to build-up a component descriptor
  - **See Also**
    * `sofa.vers.sofa.vers`

- *registerComponentManager*
  `public void` **registerComponentManager**`( sofa.common.SOFACMID` **cmID,** `sofa.interfaces.ICManager` **refCM** `)`
  - **Usage**
    * registers a Component Manager (and thus a component) into a flat list of subordinate components.
  - **Parameters**
    * `cmID` - unique identification of the subordinate component
    * `refCM` - reference to the component manager of the sub-ordinate component

---

- *run*

  - **Usage**
    * no action for now.

- *unRegisterCM*
  `public void` **unRegisterCM**`( sofa.interfaces.ICManager` **cm** `)`
  - **Usage**
    * removes a Component from a flat list of subordinate components.
  - **Parameters**
    * `cm` - reference to the component manager of the component

---

- *updateComponent*
  `public ICBuilder` **updateComponent**`( sofa.interfaces.ICManager` **myCM**`, sofa.node.repository.ComponentDescriptor` **cd** `)`
  - **Usage**
    * replaces a component in Run Part by a newer version. UNDER DEVELOPMENT!!
  - **Parameters**
    * `myCM` - reference to the component manager of the component beeing upgraded. Don't forget that the permanent part is untouched - except references to CB and implementation objects.
    * `cd` - specification of the new version of the component

16.1.2 Class **RunPartFrame**

Creates user window into the Run Part. Provides some useful runtime information, mainly for SOFAnode admin.

Declaration

public class RunPartFrame
extends javax.swing.JFrame

Serializable Fields

- private Globals fglobals
  –

Constructors

- *RunPartFrame*
  `public` **RunPartFrame**`( sofa.application.Globals` **gl** `)`

METHODS

- *processWindowEvent*
  `protected void` **processWindowEvent**`( java.awt.event.WindowEvent  e )`