DEVELOPER MANUAL

WP3.4 DATA ACCESS OPTIMIZATION

WP3.4

Document Filename:	**CG3.4-v1.0-CYF-DataAccDeveloperManual.doc**
Work package:	**WP 3.4-DataAccOpt**
Partner(s):	**CYF**
Lead Partner:	**CYF**
Config ID:	**CG3.4-v1.0-CYF-DataAccDeveloperManual**
Document classification:	**PUBLIC**

Abstract:

This document describes the internal architecture of UDAL software developed by WP3.4 within the CrossGrid project. The document is written at high technical level and requires advance knowledge about Linux operating systems as well as distributed data processing. It is addressed for developers who want to evolve provided code.

Document Log

Version	Date	Summary of changes	Author
1-0	20/11/2004	Inception of the document	Łukasz Dutka
	26/01/2005	Verified by the QE	Robert Pajak

CONTENTS

COPYRIGHT NOTICE 4

1. INTRODUCTION 5

1.1. ABBREVIATIONS AND ACRONYMS 5
1.2. REFERENCES AND SOURCE CODE 5

2. IMPLEMENTATION STRUCTURE 7

2.1. DIAGRAM - DEPENDENCIES BETWEEN ALL COMPONENTS 7
2.2. COMPONENT DESCROPTION 8
2.2.1. Component - Replica Manager (DataGrid / 3.4) 8
2.2.2. Component - GridFTP Plugin (3.4) 8
2.2.3. Component - Storage Element (3.4) 8
2.2.4. Component - Data Access Estimator (3.4) 8
2.2.5. Component - Component-Expert Subsystem (3.4) 8
2.2.6. Component - Storage Configuration (3.4) 8

3. DETAILED DESIGN 9

3.1. COMPONENT – DATA ACCESS ESTIMATOR (3.4) 9
3.1.1. Class Diagram – Main 9
3.1.2. Short Description of Classes 9
3.1.3. Sequence Diagram: Estimation of the data access factors 10
3.2. COMPONENT – STORAGE ELEMENT (3.4) 11
3.2.1. Class Diagram – Main 11
3.2.2. Short Description of Classes 12
3.3. COMPONENT – COMPONENT-EXPERT SUBSYSTEM (3.4) 13
3.3.1. Class Diagram – Main 13
3.3.2. Short Description of Classes 15
3.3.3. Sequence Diagram: CEXS Starting Sequence 18
3.3.4. Sequence Diagram: Virtualization of All Components 21
3.3.5. Sequence Diagram: New component registration 22
3.3.6. Sequence Diagram: Virtualization of One Component 23
3.3.7. Sequence Diagram: Unregistration of component 25
3.3.8. Sequence Diagram: Getting all component with the same type 26
3.3.9. Sequence Diagram: Best component selection process 26
3.3.10. Sequence Diagram: Rules interchanging 30
3.3.11. Sequence Diagram: Removing old RulesContainers 31
3.3.12. Sequence Diagram: Rules loading 33
3.3.13. Sequence Diagram: Deduction process 34
3.4. INTERNAL COMPONENT – TOOLSLIB (3.4) 36
3.4.1. Class Diagram – Main 36
3.4.2. Short Description of Classes 36

4. PRODUCT TESTING 39

5. CONTACT INFORMATION AND CREDITS 40

6. THE EDG LICENSE AGREEMENT 41

COPYRIGHT NOTICE

Copyright (c) 2004 by ACC Cyfronet AGH, Poland. All rights reserved.

Use of this product is subject to the terms and licenses stated in the EDG license agreement. Please refer to Chapter **Error! Reference source not found.** for details.

This research is partly funded by the European Commission IST-2001-32243 Project CrossGrid.

1. INTRODUCTION

Unified Data Access Layer (UDAL) provided by task 3.4 CrossGird project, provides a flexible architecture for storage nodes and storage centres. UDAL allows for a very flexible optimisation and control of the internal behaviour access to data stored in heterogeneous devices. It is fully adaptable for future purposes, but currently is used only for organizing the estimation of data access latency and bandwidth of internal storage nodes. However, it is important to notice that the provided solution could be also directly used for other purposes. The current version of UDAL is distributed together with a set of specialized components for data access cost estimation for data stored in secondary and tertiary storage.

1.1. ABBREVIATIONS AND ACRONYMS

Castor	HSM system developed at CERN
CrossGrid	The EU CrossGrid Project IST-2001-32243
CG	The EU CrossGrid Project IST-2001-32243
CEA	Component-Expert Architecture
CEComponent	A component designed for CEA
DataGrid	The EU DataGrid Project IST-2000-25182
DRM	Disk Resource Manager
EDG	European Data Grid Project - IST-2000-25182
GSI	Grid Security Infrastructure
HSM	Hierarchical Storage Management
HPSS	High Performance Storage System
HRM	HSM Resource Manager
MSS	Mass Storage System
RLS	Replica Location Service
SOAP	Simple Object Access Protocol
SRS	Software Requirements Specification
SDD	Software Design Description
TRM	Tape Resource Manager

UniTree Tape Archival/Management System, formerly – UniTree Central File Manager, at present – Legato DiskXtender

1.2. REFERENCES AND SOURCE CODE

The source code of this application and how to install it is fully described in the Installation Guide of the application. Another important document to read for more advanced users is Developers Guide.

All scientific as well as low-level technical aspects the described within this document modules are thoroughly described in the papers below.

1. Kitowski J., Dutka Ł., Słota R., Nikolow D., "Prototype Documentation – First Prototype" – D3.3 http://www.eu-crossgrid.org/Deliverables/M12pdf/CG3.4-D33-v1.1-CYF021-PrototypeDescriptionRpt.pdf.

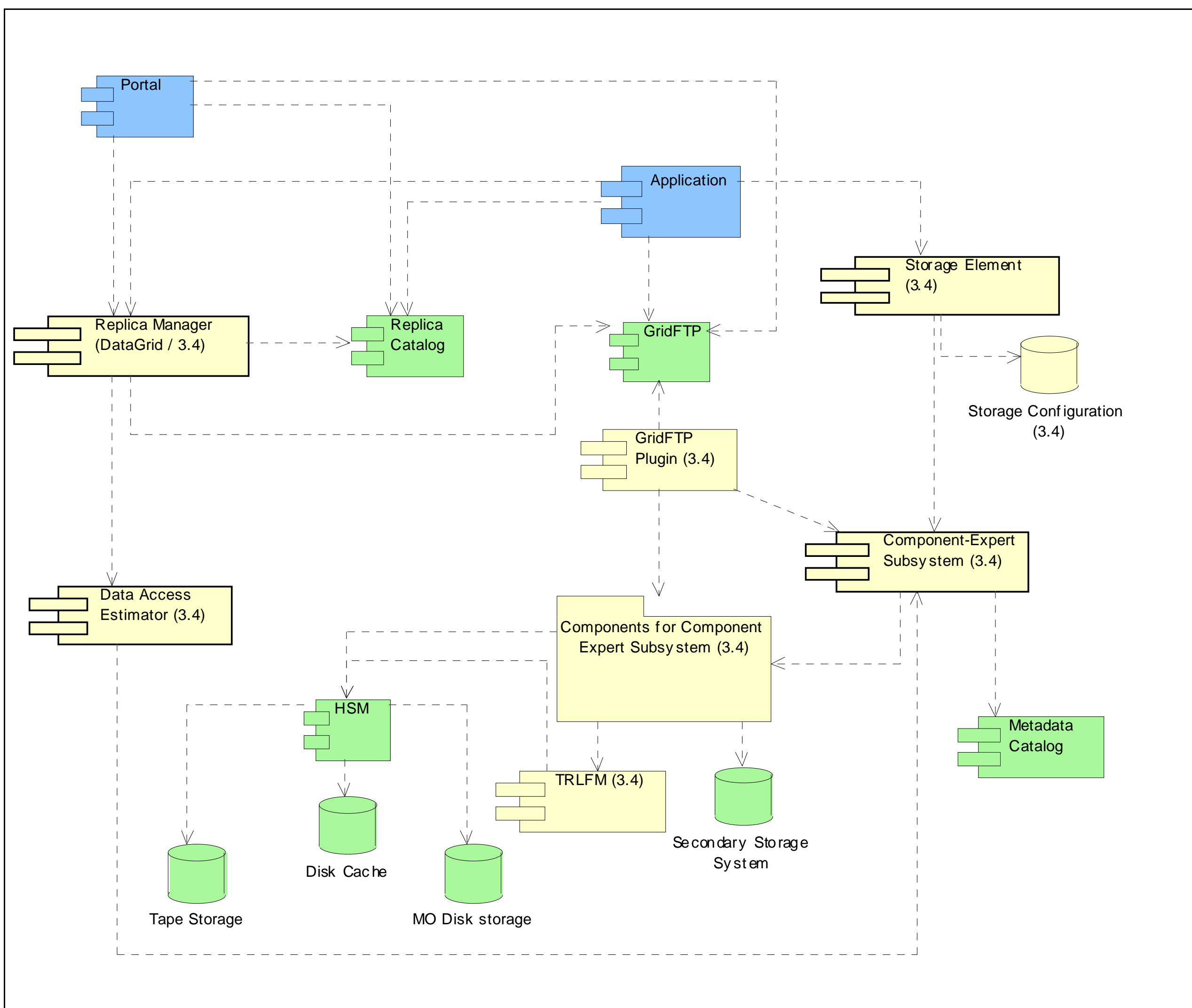
2. Kitowski J., Dutka Ł., Słota R., Nikolow D., "Task 3.4 SRS Optimization of Data Access" – D3.1 http://www.eu-crossgrid.org/Deliverables/M3pdf/CG-3.4-SRS-0012.pdf.
3. Kitowski J., Dutka Ł., Słota R., Nikolow D., "Task 3.4 SDD Software Design Description - Final" – D3.2
4. Dutka L., Slota R., Kitowski J., "Component-Expert Architecture as Flexible Environment for Selection of Data-handlers and Data-Access-Estimators in CrossGrid", in: Bubak, M., Noga, M., Turala, M. (Eds.), "Proceedings of Cracow'02 Grid Workshop", December 11-14, 2002, Cracow, Poland, ACC Cyfronet UMM, 2002, Kraków, pp. 201-208.
5. Nikolow D., Slota R., Kitowski J., "Data Access Time Estimation for HSM Systems in Grid Environment", in: Bubak, M., Noga, M., Turala, M. (Eds.), "Proceedings of Cracow'02 Grid Workshop", December 11-14, 2002, Cracow, Poland, ACC Cyfronet UMM, 2002, Kraków, pp. 209-216.
6. Dutka Ł. and J. Kitowski, "Application of Component-Expert Technology for Selection of Data-Handlers in CrossGrid", in: D. Kranzlmueller, P. Kacsuk, J. Dongarra, J. Volkert (Eds.), Proc. 9th European PVM/MPI Users' Group Meeting, Sept. 29 - Oct. 2, 2002, Linz, Austria, Lect.Notes on Comput.Sci., vol.2474, Springer, 2002, pp. 25-32.
7. Nikolow D., Slota R., Dziewierz M., Kitowski J., "Access Time Estimation for Tertiary Storage Systems", in: Monien, B., Feldman, R. (Eds.), Euro-Par 2002 Parallel Processing, 8th International Euro-Par Conference Paderborn, Germany, August 27-30, 2002 Proceedings , no. 2400, Lecture Notes in Computer Science, Springer, 2002, pp. 873-880.

2. IMPLEMENTATION STRUCTURE

This section provides a general idea of the system decomposition using UML component-diagrams. Each of these diagrams has its own subsection. In every section the diagrams are supported by short descriptions making their interpretation easier.

2.1. DIAGRAM - DEPENDENCIES BETWEEN ALL COMPONENTS

This diagram shows dependencies between all important UDAL components. The components, which are task 3.4 ones, are marked additionally by '(3.4)' and are filled by light yellow color. The doted directed line between two entities means that the first one depends on the second. This diagram does not provide information about communication between components but only shows dependencies between them.



2.2. COMPONENT DESCROPTION

Interfaces are assigned to components. Thus, this description is grouped by components. Each component has its own section in this chapter, and each assigned interface has its own subsection. This subsection specifies only the external interfaces.

2.2.1. Component - Replica Manager (DataGrid / 3.4)

Replica Manager is an advisor for selection of migration/replication policy in defined user states. Its main goal is to suggest whether and when a chosen data file should be replicated into the local environment of the remote user in order to shorten the waiting time for data availability. The EDG Replica Manager looks to be the best for the CrossGrid project, however some small extension of this component is planned. The abbreviation of this component name is REMG.

2.2.2. Component - GridFTP Plugin (3.4)

Plugins allow GridFTP to expand its ability to take control on any Grid command. This feature is used to connect GridFTP to Component-Expert Subsystem and to use the chosen components as local data serving modules. This component should be deployed on each Storage Element. The abbreviation of this component name is GPLG. This component is provided in beta version.

2.2.3. Component - Storage Element (3.4)

Storage Element is responsible for publishing its current configuration and answering general questions about the configuration, e.g., when the administrator of a storage element registers storage somewhere then he does not need to specify any value of attributes of his storage but he just enters the address of the registered storage and the rest of information will be obtained automatically. Additionally, Storage Element will be used as a temporary (transparent) tunneling platform for SOAP, which may be used for additional control of the data transfer when Component-Expert Subsystem is used. This component should be deployed on each Storage Element. The abbreviation of this component name is STEL.

2.2.4. Component - Data Access Estimator (3.4)

Data Access Estimator estimates data access cost for a given data storage system. It can return many measures of cost, e.g., latency, bandwidth, etc. This component should be located on each Storage Element. The abbreviation of this component name is DAES.

2.2.5. Component - Component-Expert Subsystem (3.4)

This component realizes the proposed Component-Expert (CE) strategy. It has many responsibilities. First of all, it registers new CE components, which can be chosen as the best matching CE component in some context. It also returns a handle to the best matching CE component in some context. This component should be located on each Storage Element. The abbreviation of this component name is CEXS.

2.2.6. Component - Storage Configuration (3.4)

Storage Configuration artifact represents the actual settings of Storage Element. It is used by Storage Element. This component should be deployed one each Storage Element.

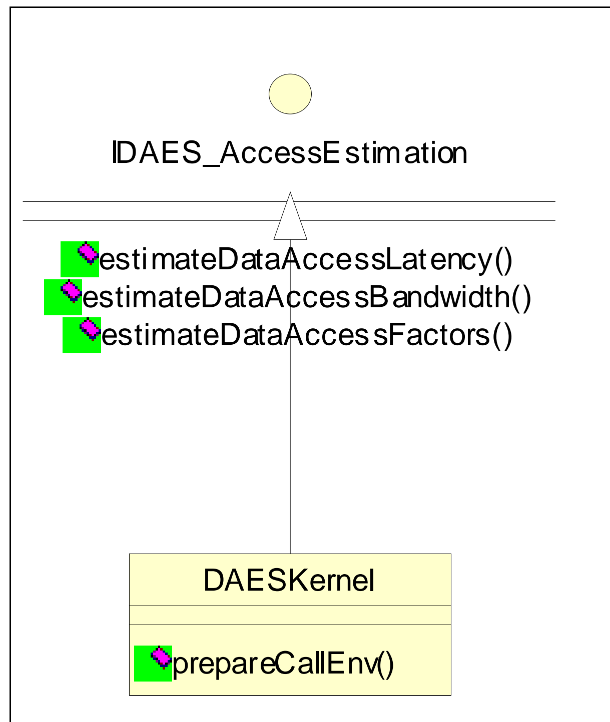
3. DETAILED DESIGN

This section describes the body of each component in separate subsection Each subsection presents a description of internal structure of respective component using UML class-diagrams and short comments on the diagrams. Additionally, these sections present detailed description of every class belonging to the respective component. To make functionality of some components more clear, in some cases subsections describing internal behavior of the components have been introduced.

3.1. COMPONENT – DATA ACCESS ESTIMATOR (3.4)

3.1.1. Class Diagram – Main

This diagram presents internal structure of the DAES component. Since this component is just an intermediate one then its internal structure is quite simple. It realizes one interface, which is accessible via the SOAP protocol and its main role is to receive messages and repacked them into other compatible with CEXS component and interprets results found by a component selected by CEXS.



3.1.2. Short Description of Classes

3.1.2.1. Class - DAESKernel

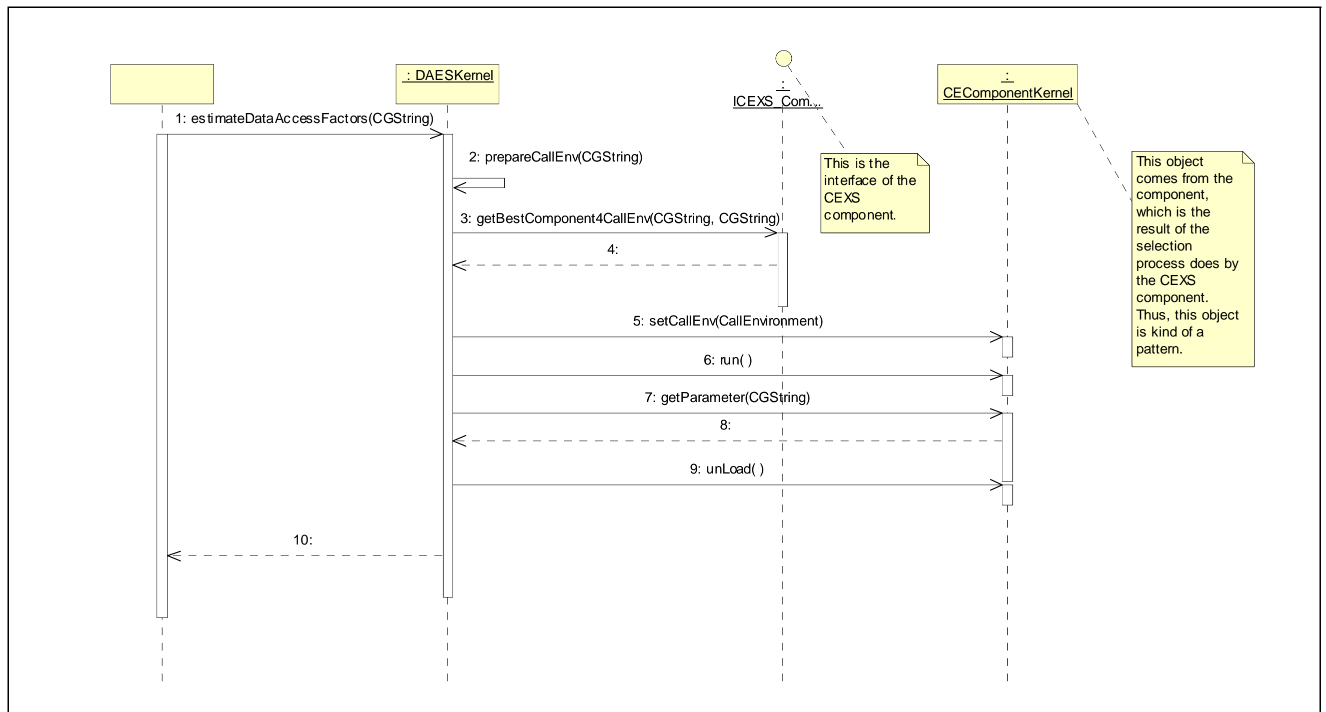
DAESKernel class is a runtime class of DAES component. It makes communication between clients and DAES. This class is an intermediary class, which processes requests, translates them into other requests to CEXS and processes them after selected by CEXS components. The real estimation is done by the component registered in the CEXS container. The estimation algorithms are depended on the type of device, which holds data.

3.1.2.2. Interface - IDAES_AccessEstimation

This interface is used to pass an inquiry about the cost of access to some file for a given Storage Element.

3.1.3. Sequence Diagram: Estimation of the data access factors

This diagram depicts in a simple way the sequences of the process estimation of the data-access factors - a data-latency or data-bandwidth.



Sequence description:

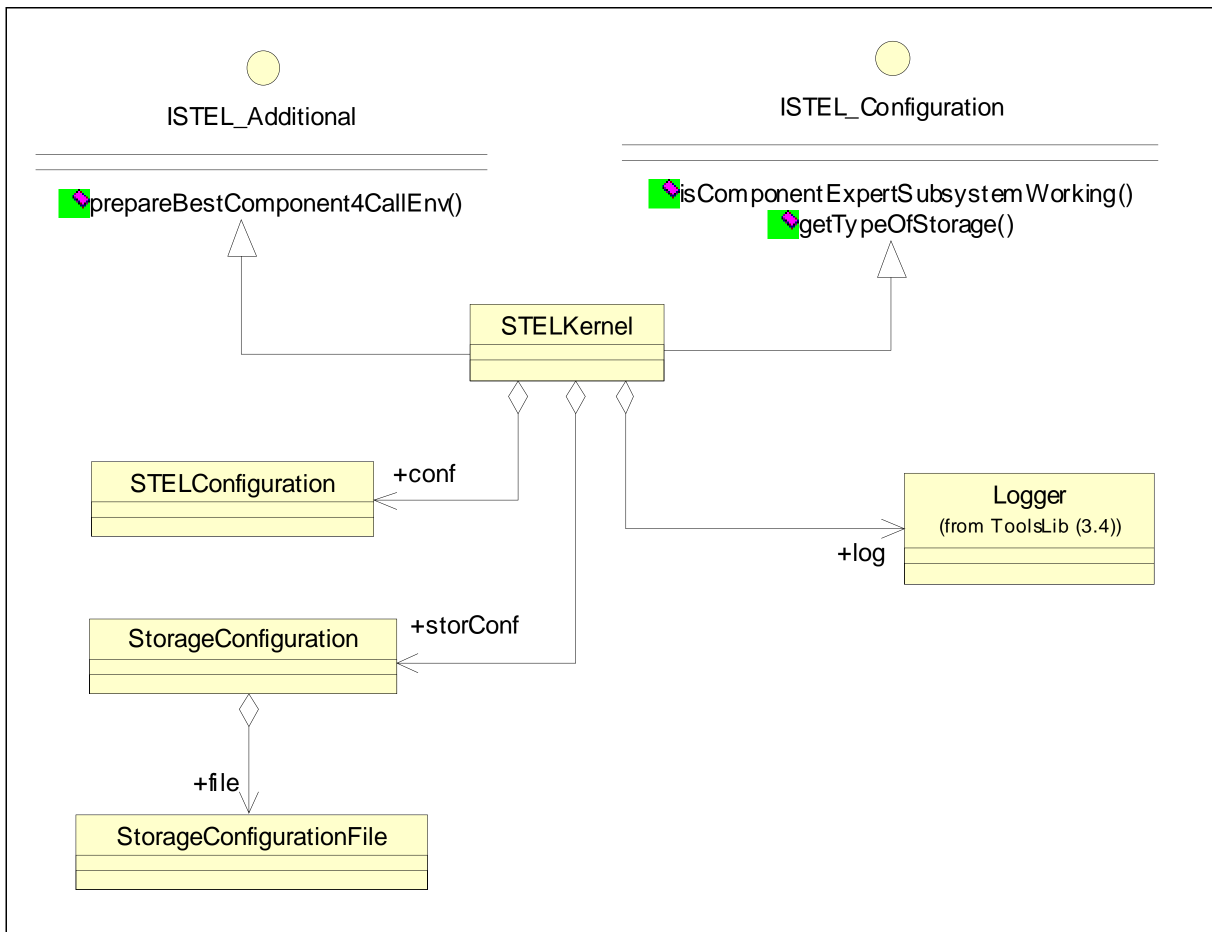
- Message no.: 1 – `estimateDataAccessFactors(CGString)`
  Documentation: DAESKernel obtains request from the network to estimate data-access factors (bandwidth or latency). This request can be sent by any component in the network, which has right to connect to it.
- Message no.: 2 – `prepareCallEnv(CGString)`
  Documentation: This message carries an unique data identifier. The goal of this message is to prepare the valid call-environment, which is next passed to CEXS.
- Message no.: 3 – `getBestComponent4CallEnv(CGString, CGString)`
  Documentation: This message is passed to the CEXS component. It passes the type of required component and call-environment, which are used in the deduction process. This message indicates the deduction process, which result is the best component given type matching to the passed call-environment.
- Message no.: 4
  Documentation: It returns a handle to the best component or error if no component match.

- Message no.: 5 – `setCallEnv(CallEnvironment)`
  Documentation: This is message passes the call-environment to the found component.
- Message no.: 6 – `run( )`
  Documentation: The component is activated.
- Message no.: 7 – `getParameter(CGString)`
  Documentation: The result of the component is just a simple set of numbers, so it can be obtained using getParameter method.
- Message no.: 8
  Documentation: This message passes the value of required parameter.
- Message no.: 9 – `unLoad( )`
  Documentation: The component is no longer needed.
- Message no.: 10
  Documentation: The estimated factors are returned.

3.2. COMPONENT – STORAGE ELEMENT (3.4)

3.2.1. Class Diagram – Main

This diagram shows the internal structure of the Storage Element component. The Storage Element is responsible for answering the questions about storage configuration and storage kind that keeps the respective data, if CEXS is installed and operational.



3.2.2. Short Description of Classes

3.2.2.1. Interface - ISTEL_Additional

This interface combines all additional operations, which are not matching other interfaces, e.g. temporary operations for a control of component-expert subsystem.

3.2.2.2. Interface - ISTEL_Configuration

This interface combines all operations related to obtaining information about current Storage Element configuration.

3.2.2.3. Class - STELKernel

This is the main class realizing interfaces and keeping a runtime code.

3.2.2.4. Class - STELConfiguration

The STEL component needs some configuration data, e.g., where logs should be stored or what detailed level of logging activity is, etc. These base configurations items are stored in a special configuration file, which name is known to STEL before (it is embedded in code or passed as a command line parameter). This class represents the object, which virtualizes that configuration file. It can read the configuration file and set respective fields, which next are used by the STELKernel class.

3.2.2.5. Class - StorageConfiguration

The StorageConfiguration class answers to the questions what type of device currently keeps the respective data. It has special objects StorageConfigurationFile, which can read the storage configuration file.

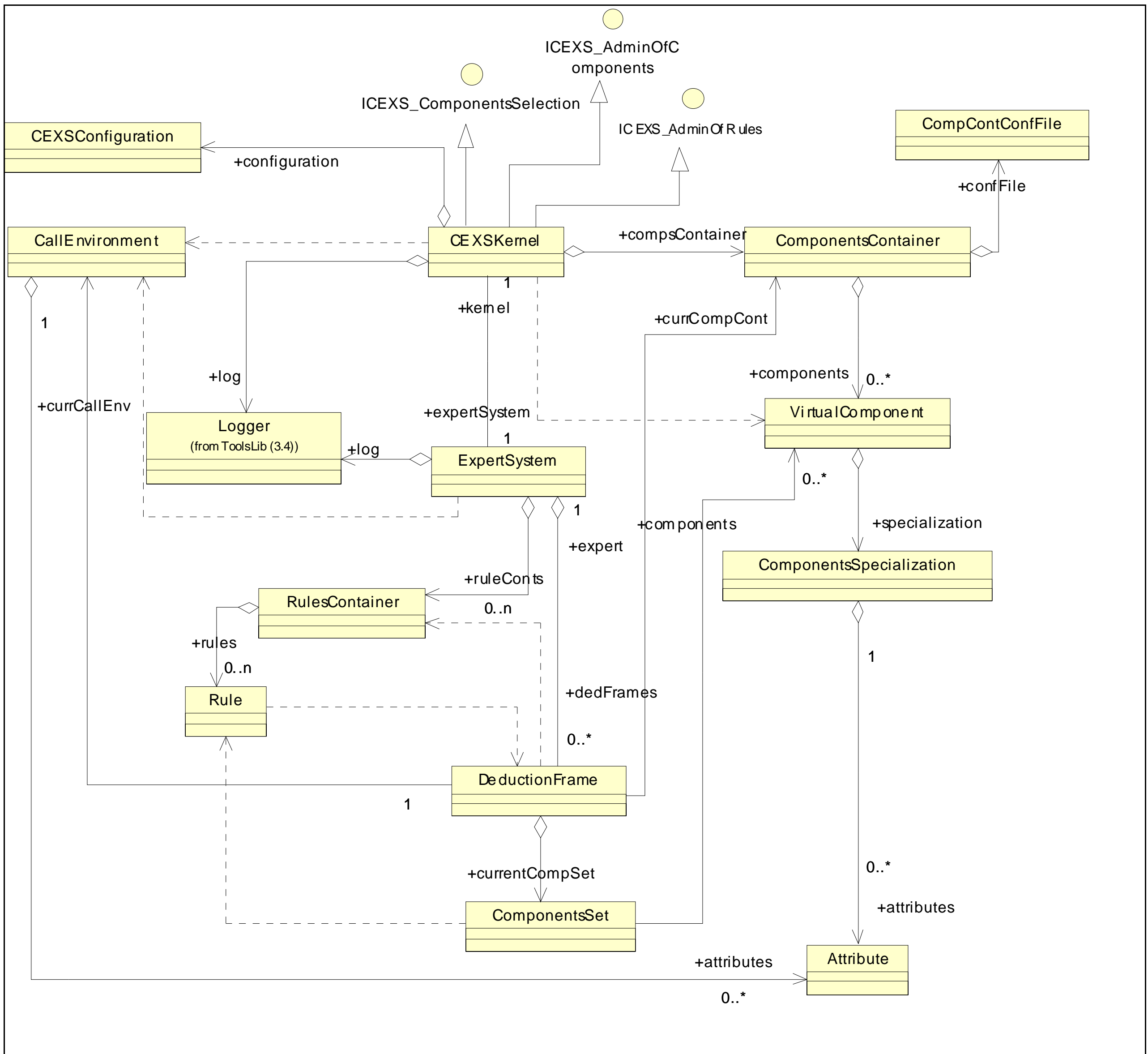
3.2.2.6. Class - StorageConfigurationFile

The StorageConfigurationFile class controls the configuration file. It can read data form file and interpret its.

3.3. COMPONENT – COMPONENT-EXPERT SUBSYSTEM (3.4)

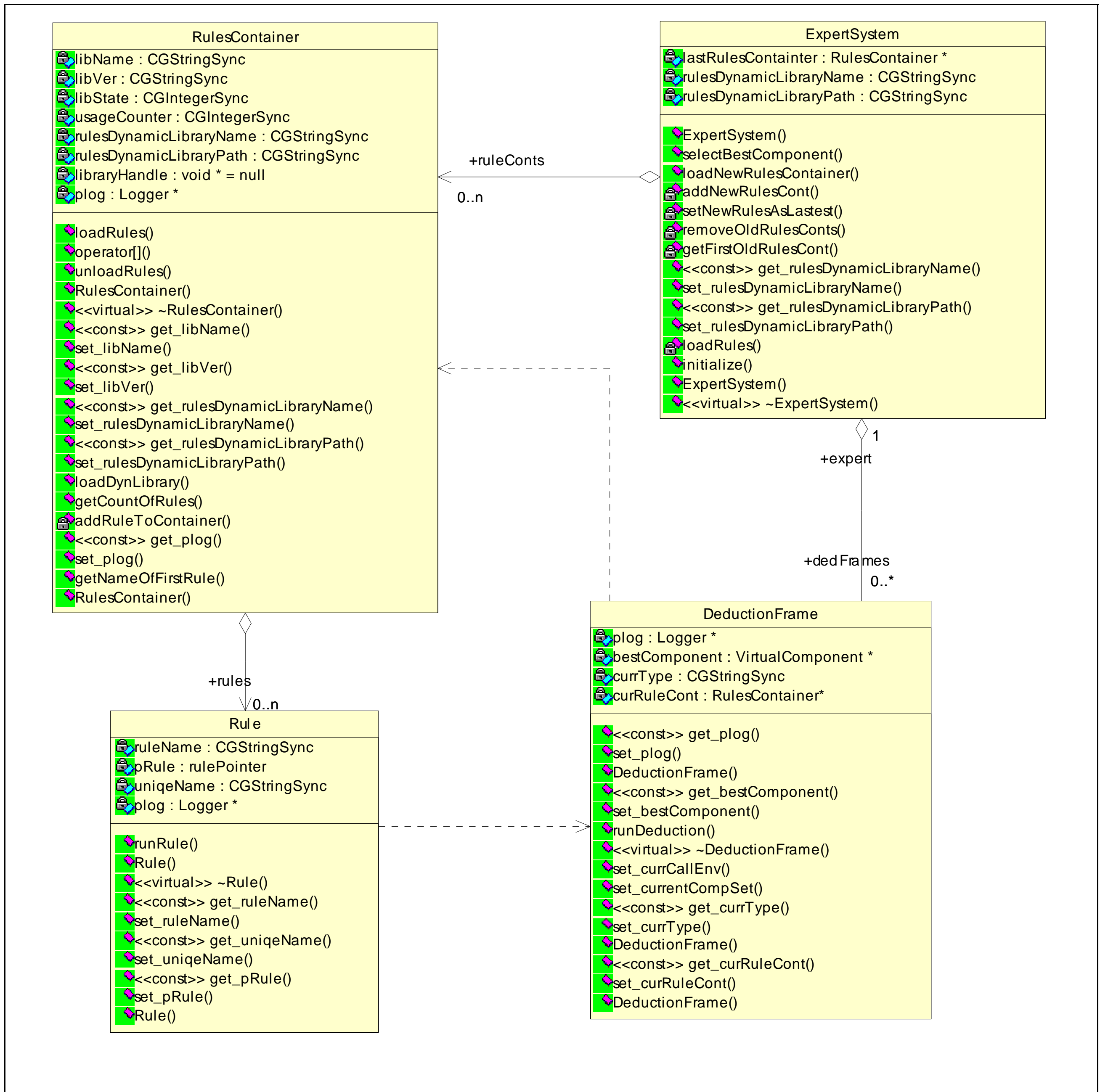
3.3.1. Class Diagram – Main

This diagram shows relationships between classes, which construct CEXS. The role of this diagram is to describe structure of CEXS body. The description of all classes and relationships between them is available below. To make this diagram more readable the classes are shown without operations and attributes. Important elements are interfaces, which are connected just to the CEXSKernel, thus all operation coming from peripheries go through this class.



Class Diagram – Rules and Expert System

This diagram shows relationships between classes connected with the representation of the rules used to make the expert system.



3.3.2. Short Description of Classes

3.3.2.1. Class - ComponentsContainer

Represents the container keeping all registered components. Once registered the component is available until it is unregistered. Each registration is kept in confFile:CompContConfFile.

3.3.2.2. Class - CallEnvironment

The class CallEnvironment represents the call-environment artifact from the Component-Expert Architecture. It consists of a list of attributes with their values.

3.3.2.3. Class - VirtualComponent

The VirtualComponent class is used to represent a physical instance of component. This is a structure of required information about components. It consists of the component type and the component specialization.

3.3.2.4. Class - ComponentsSpecialization

The ComponentSpecialization class represents specialization of components coming from the Component-Expert Architecture. It consists of a list of attributes.

3.3.2.5. Class - ExpertSystem

The ExpertSystem class represents the rule based expert system coming form the Component Expert Architecture. This class is an intermediary between the kernel and the deduction frame. The main objective of this class is preparation of the DeductionFrame.

3.3.2.6. Class - Attribute

The Attribute is an element of the component specialization, however this is also an element of the call-environment. It keeps the name of attribute and the value, which may be of different types.

3.3.2.7. Class - CEXSKernel

The CEXSKernel class is the class realizing interfaces; keeping all other classes together with the runtime code.

3.3.2.8. Class - ComponentsSet

To make the system more flexible, each call for the best component creates a deduction frame. It holds the internal variables, which must be independent during the deduction process. These frames keep the called call-environment, the type of searched component, a useful component set, which is used during deduction process etc.

3.3.2.9. Class - CEXSConfiguration

This class controls the configuration of the CEXS-component. It reads current state form the configuration file and serves values for a question.

3.3.2.10. Class - DeductionFrame

The DeductionFrame class keeps elements necessary for the deduction process. Parallelization of the deduction process of many queries is possible. The final structure of this class has not been set yet.

3.3.2.11. Class - CompContConfFile

This class manages the file, which keeps persistent information needed to start ComponentsContainer. It is a list of URLs (paths) to each component registered in

ComponentsContainer. This file will be updated during the system running after each new component registration.

3.3.2.12. Interface - ICEXS_AdminOfComponents

The administrator of the storage element uses this interface to register or to unregister components for Component-Expert (CE) Subsystem. All operations concerning administration of CE components go through this interface.

3.3.2.13. Interface - ICEXS_ComponentsSelection

Component Expert Subsystem selects the best matching CE component for a given call-environment. This interface is used to run such a process and to get handle to the best CE component.

3.3.2.14. Class - RulesContainer

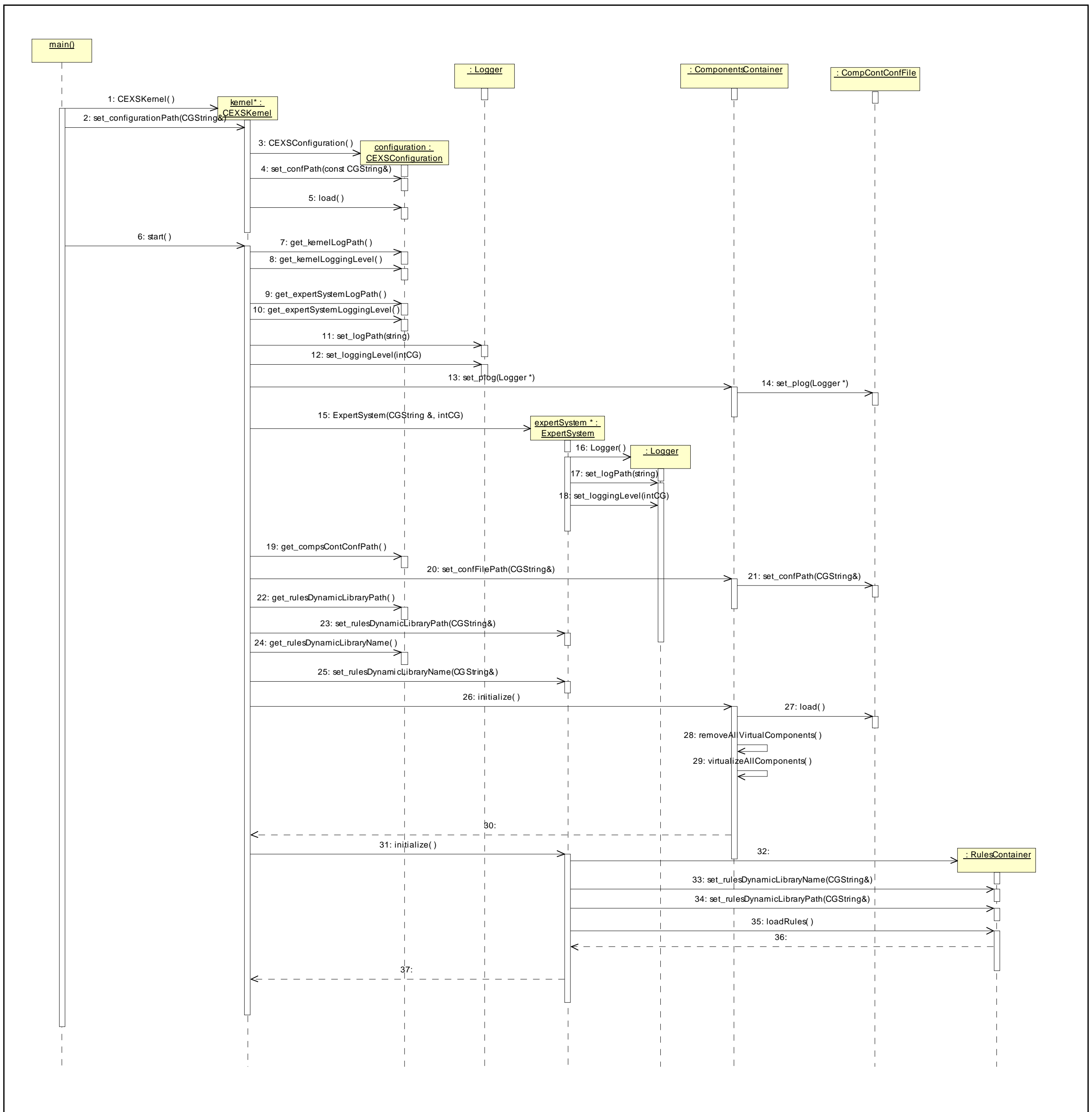
This class represents the dynamic-linking library contains rules. Each library is identified by the file name, e.g. librule.so.1

3.3.2.15. Class - Rule

This class represents one rule, which could be fired during the deduction process.

3.3.3. Sequence Diagram: CEXS Starting Sequence

This diagram presents the starting sequence of the whole component and shows a process of component preparation.



Sequence description:

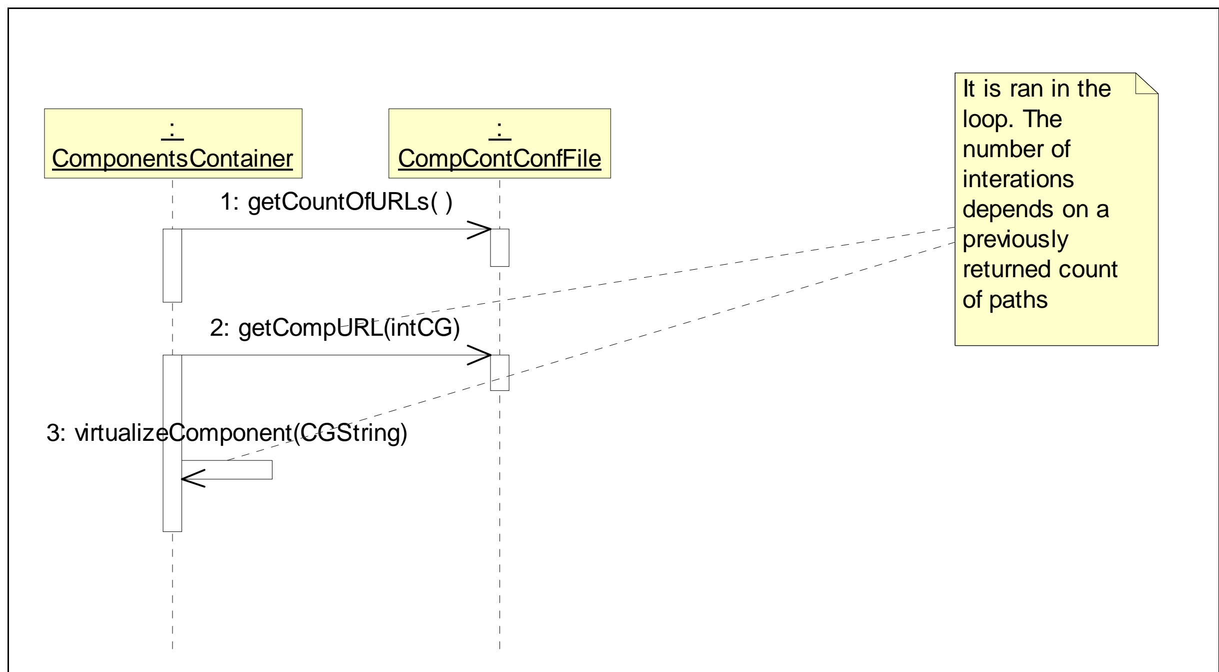
- Message no.: 1 – `CEXSKernel( )`
  Documentation: An instance of the CEXSKernel class is created in the main section of the component. Important fact is that calls to the CEXSKernel should not block it, thus CEXSKernel has to possess ability of operation with concurrent calls.
- Message no.: 2 – `set_configurationPath(CGString&)`
  Documentation: Kernel loads configuration from a file, thus it requires to work properly the path to the file keeping its configuration.
- Message no.: 3 – `CEXSConfiguration( )`
  Documentation: The creation of a configuration reader.
- Message no.: 4 – `set_confPath(const CGString&)`
  Documentation: Kernel passes the path to the configuration file.
- Message no.: 5 – `load( )`
  Documentation: Loading of data from the configuration file.
- Message no.: 6 – `start( )`
  Documentation: As a second stage of the preparation phase of the kernel it should be started. This phase creates instances of all needed objects.
- Message no.: 7 – `get_kernelLogPath( )`
  Documentation: It obtains the path to the file where log-items will be stored. This path is read before form the configuration file.
- Message no.: 8 – `get_kernelLoggingLevel( )`
  Documentation: Since Logger needs the logging level parameter, therefore it is obtained here.
- Message no.: 9 – `get_expertSystemLogPath( )`
  Documentation: It obtains the path to the log file for the ExpertSystem class, because it is different then log file for kernel.
- Message no.: 10 – `get_expertSystemLoggingLevel( )`
  Documentation: The log file for the ExpertSystem object works with differ logging level. Thus, it is obtained here.
- Message no.: 11 – `set_logPath(string)`
  Documentation: Setting the path to the log-file for the Logger of CEXSKernel.
- Message no.: 12 – `set_loggingLevel(intCG)`
  Documentation: Setting the logging level for the Logger of CEXSKernel.
- Message no.: 13 – `set_plog(Logger *)`
  Documentation: Since most of classes need a logger thus here is set a logger attached to the CEXSKernel object as a logger for the CallEnvironment object.
- Message no.: 14 – `set_plog(Logger *)`
  Documentation: ComponentsContainer passes the pointer to the logger to the CompContConfFile object.
- Message no.: 15 – `ExpertSystem(CGString &, intCG)`
  Documentation: The instance of the ExpertSystem class is created. Additionally, a path to the log-file and logging level of Logger is passed.

- Message no.: 16 – `Logger( )`
  Documentation: The creation of Logger attached to the ExpertSystem.
- Message no.: 17 – `set_logPath(string)`
  Documentation: The setting of a path to the log-file.
- Message no.: 18 – `set_loggingLevel(intCG)`
  Documentation: The setting of logging level.
- Message no.: 19 – `get_compsContConfPath( )`
  Documentation: An instance of ComponentContainer class keeps the configuration in a file, which is managed by the CompContConfFile class. This message obtains from the configuration the path to this file.
- Message no.: 20 – `set_confFilePath(CGString&)`
  Documentation: Previously obtained path is set here.
- Message no.: 21 – `set_confPath(CGString&)`
  Documentation: The path is passed to the configuration manager.
- Message no.: 22 – `get_rulesDynamicLibraryPath( )`
  Documentation: An instance of the ExpertSystem class loads the dynamic-linking library with rules. This library is stored in some directory. This message takes this information form the CEXSConfiguration where this path is kept.
- Message no.: 23 – `set_rulesDynamicLibraryPath(CGString&)`
  Documentation: Obtained as a result of the previous message path to the directory keeping the rules-libraries; it is passed now to ExpertSystem.
- Message no.: 24 – `get_rulesDynamicLibraryName( )`
  Documentation: The rules, as it was mentioned earlier, is stored in a dynamic-linking library. This library has a name, e.g., librules.so. However, each compilation of the rules creates the new version of the rule-library and some versions may be used concurrently for a while; this name should be a symbolic link to the newest library and the real name of library should contain the version indicator, e.g., librules.so.1.123.
- Message no.: 25 – `set_rulesDynamicLibraryName(CGString&)`
  Documentation: Name of the dynamic-linking library keeping the rules, obtained as a result of the previous message, is passed to the ExpertSystem now.
- Message no.: 26 – `initialize( )`
  Documentation: ComponentContainer must be initialized after startup.
- Message no.: 27 – `load( )`
  Documentation: The data loading.
- Message no.: 28 – `removeAllVirtualComponents( )`
  Documentation: All virtual components associated with ComponentContainer are removed now, to avoid duplication of virtual components.
- Message no.: 29 – `virtualizeAllComponents( )`
  Documentation: The message starts the virtualization of all components registered before. This process is described in another sequence diagram.
- Message no.: 30
  Documentation: An execution code is returned, as a result of the initialization.

- Message no.: 31 – `initialize( )`
  Documentation: Similarly to the ComponentsContainer the ExpertSystem must be initialized. The main goal of the initialization process is to load an appropriate dynamic-linking library with the rules, and to virtualize them.
- Message no.: 32
  Documentation: During the initialization process the instance of the RulesContainer class is created. This instance virtualizes the dynamic-linking library with the rules.
- Message no.: 33 – `set_rulesDynamicLibraryName(CGString&)`
  Documentation: The library file name is required to set the value of appropriate field in the instance of the ExpertSystem class.
- Message no.: 34 – `set_rulesDynamicLibraryPath(CGString&)`
  Documentation: The path to the library is required to set the value of appropriate field in the instance of the ExpertSystem class.
- Message no.: 35 – `loadRules( )`
  Documentation: During the initialization process the rules are loaded from the dynamic linking library. Details of this process are presents in the sequence diagram 'Rules loading'.
- Message no.: 36
  Documentation: The execution code of the rule loading is returned now.
- Message no.: 37
  Documentation: An execution code is returned, as a result of initialization.

3.3.4. Sequence Diagram: Virtualization of All Components

This diagram depicts the interactions during the virtualization sequence of components registered previously in ComponentsContainer.

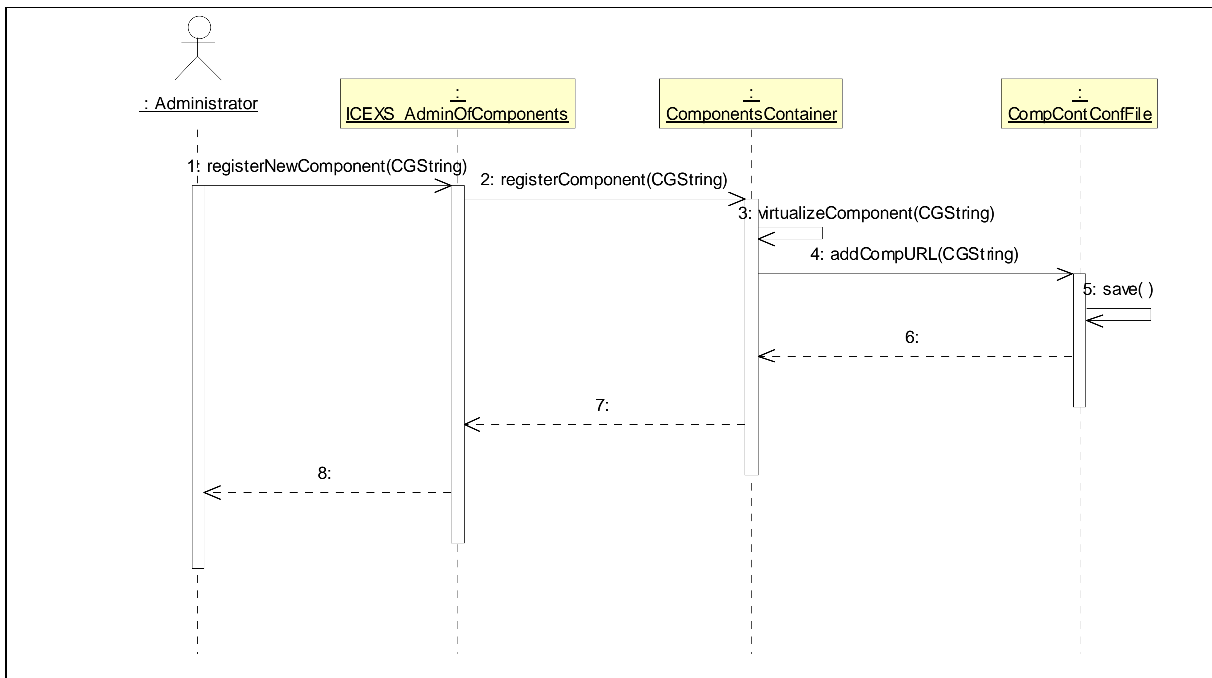


Sequence description:

- Message no.: 1 – `getCountOfURLs( )`
  Documentation: ComponentsContainer gets a number of URLs of previously registered components.
- Message no.: 2 – `getCompURL(intCG)`
  Documentation: For each number of URLs less then the previously obtained number of URLs (in a loop), ComponentContainer gets the current URL to the registered component.
- Message no.: 3 – `virtualizeComponent(CGString)`
  Documentation: Each obtained URL is used to virtualization of a registered component. The virtualization sequence is stated in other diagram.

3.3.5. Sequence Diagram: New component registration

This diagram shows the sequence of messages during a new component registration process. The internal sequence of the 'virtualizeComponent' message is depicted in 'Virtualization One Component' diagram.



Sequence description:

- Message no.: 1 – `registerNewComponent(CGString)`
  Documentation: An administrator or a programmer registers a new component. To do it, the string with a unique URL pointing the registering component should be passed.
- Message no.: 2 – `registerComponent(CGString)`
  Documentation: The URL is passed to ComponentsContainer.
- Message no.: 3 – `virtualizeComponent(CGString)`
  Documentation: ComponentsContainer calls its own special method for virtualization

of components and passes URL to it. The process of virtualization is shown in other diagram.

– Message no.: 4 – `addCompURL(CGString)`
Documentation: After successful virtualization, URL is added to CompContConfFile.

– Message no.: 5 – `save( )`
Documentation: A modification is saved.

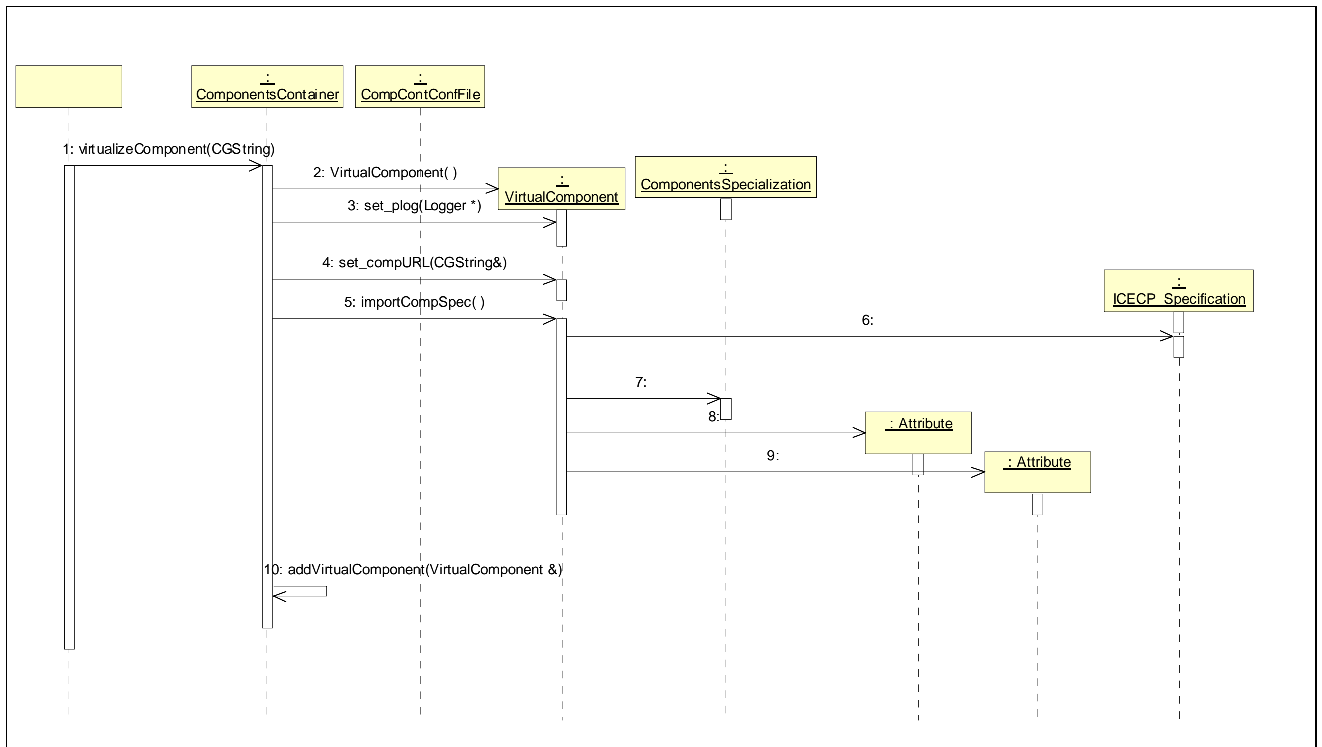
– Message no.: 6
Documentation: Acknowledge is returned.

– Message no.: 7
Documentation: Acknowledge is returned.

– Message no.: 8
Documentation: Acknowledge is returned.

3.3.6. Sequence Diagram: Virtualization of One Component

This diagram presents the process of virtualization of one component.



Sequence description:

– Message no.: 1 – `virtualizeComponent(CGString)`
Documentation: An URL to virtualized component is passed.

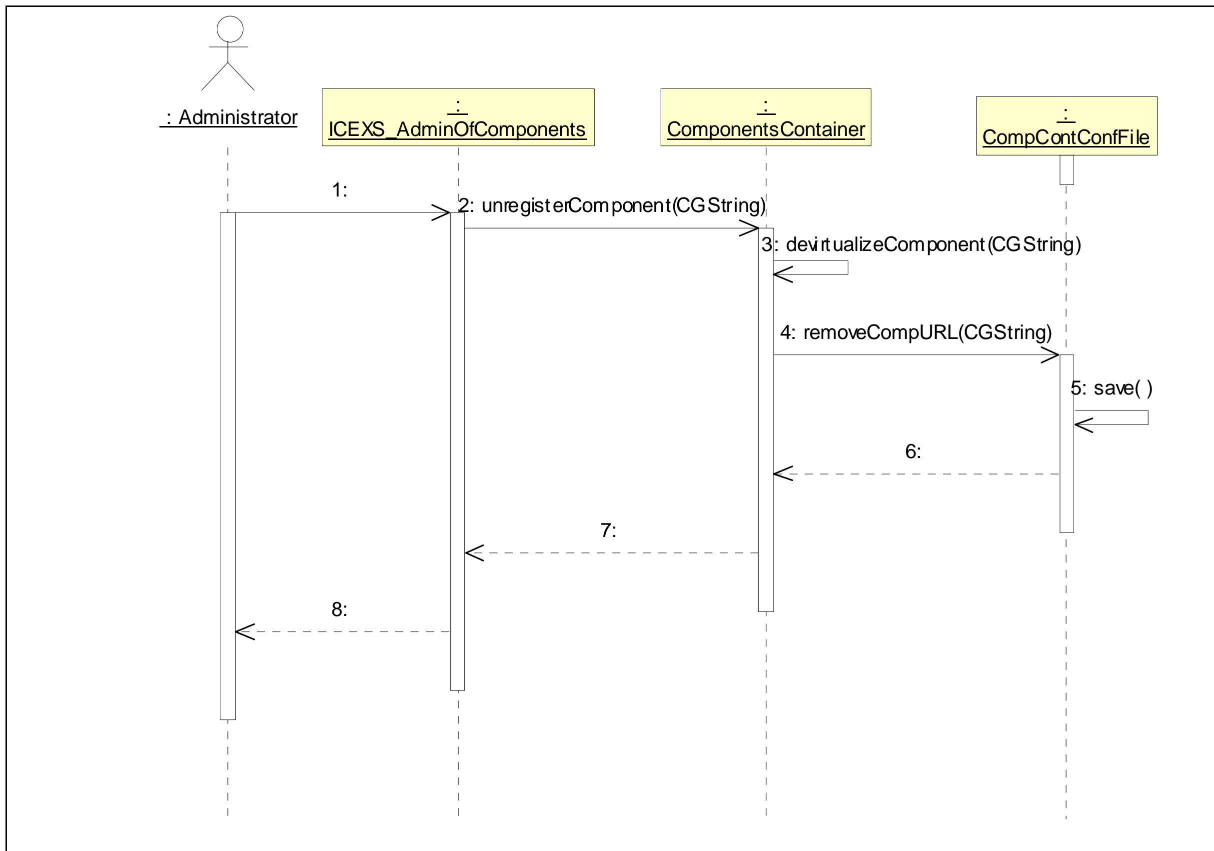
– Message no.: 2 – `VirtualComponent( )`
Documentation: A new instance of VirtualComponent is created.

– Message no.: 3 – `set_plog(Logger *)`
Documentation: A logger may be useful; therefore the pointer to the logger used by ComponentContainer is passed.

- Message no.: 4 – `set_compURL(CGString&)`
  Documentation: The URL to virtual instance of physical component is set.
- Message no.: 5 – `importCompSpec( )`
  Documentation: The process of import and registering the component is fired. This is quite a complicated part, since this importing process transcends the bounds of this component. During this process the type and the specialization of a registered component are gathered.
- Message no.: 6
  Documentation: This is a transcendental call, which must be described with more detail on this diagram, but the inter-component communication problem should be solved formerly. As a result of this message should be the type and specialization of component pointed by URL.
- Message no.: 7
  Documentation: The component specialization of the virtualized component is constructed using data obtained from previous step.
- Message no.: 8
  Documentation: The ComponentSpecialization consists of many attributes with their values and this list of attributes is constructed here.
- Message no.: 9
  Documentation: The ComponentSpecialization consists of many attributes with their values and this list of attributes is constructed here.
- Message no.: 10 – `addVirtualComponent(VirtualComponent &)`
  Documentation: It just adds the created virtual components to the set of components.

3.3.7. Sequence Diagram: Unregistration of component

This is the unregistration sequence of some component which identifier is specified as a parameter.



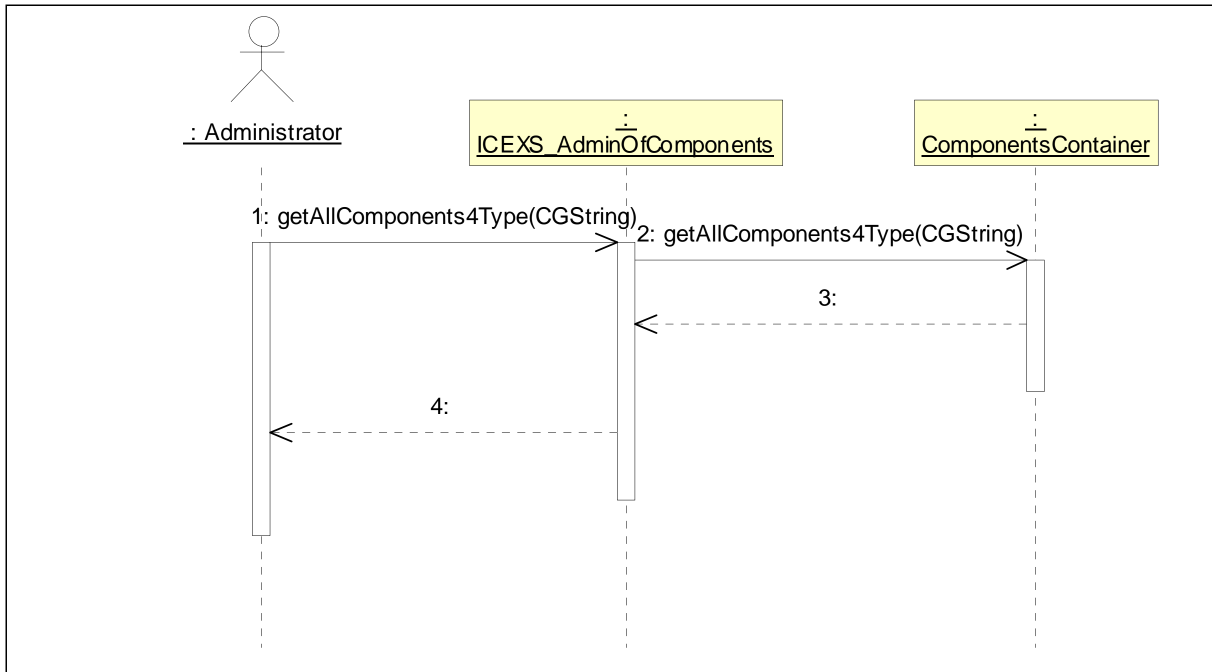
Sequence description:

- Message no.: 1
  Documentation: The URL to unregistering component is passed.
- Message no.: 2 – `unregisterComponent(CGString)`
  Documentation: The obtained URL is passed to ComponentContainer by an appropriate method calls.
- Message no.: 3 – `devirtualizeComponent(CGString)`
  Documentation: The internal method to devirtualization of components is called.
- Message no.: 4 – `removeCompURL(CGString)`
  Documentation: The URL to unregisteredComponent must be removed form CompContConfFile.
- Message no.: 5 – `save( )`
  Documentation: A modification of CompContConfFile must be stored.
- Message no.: 6
  Documentation: Acknowledge is returned.

- Message no.: 7
  Documentation: Acknowledge is returned.
- Message no.: 8
  Documentation: Acknowledge is returned.

3.3.8. Sequence Diagram: Getting all component with the same type

The process of getting of all components with some type is shown in this sequence diagram.

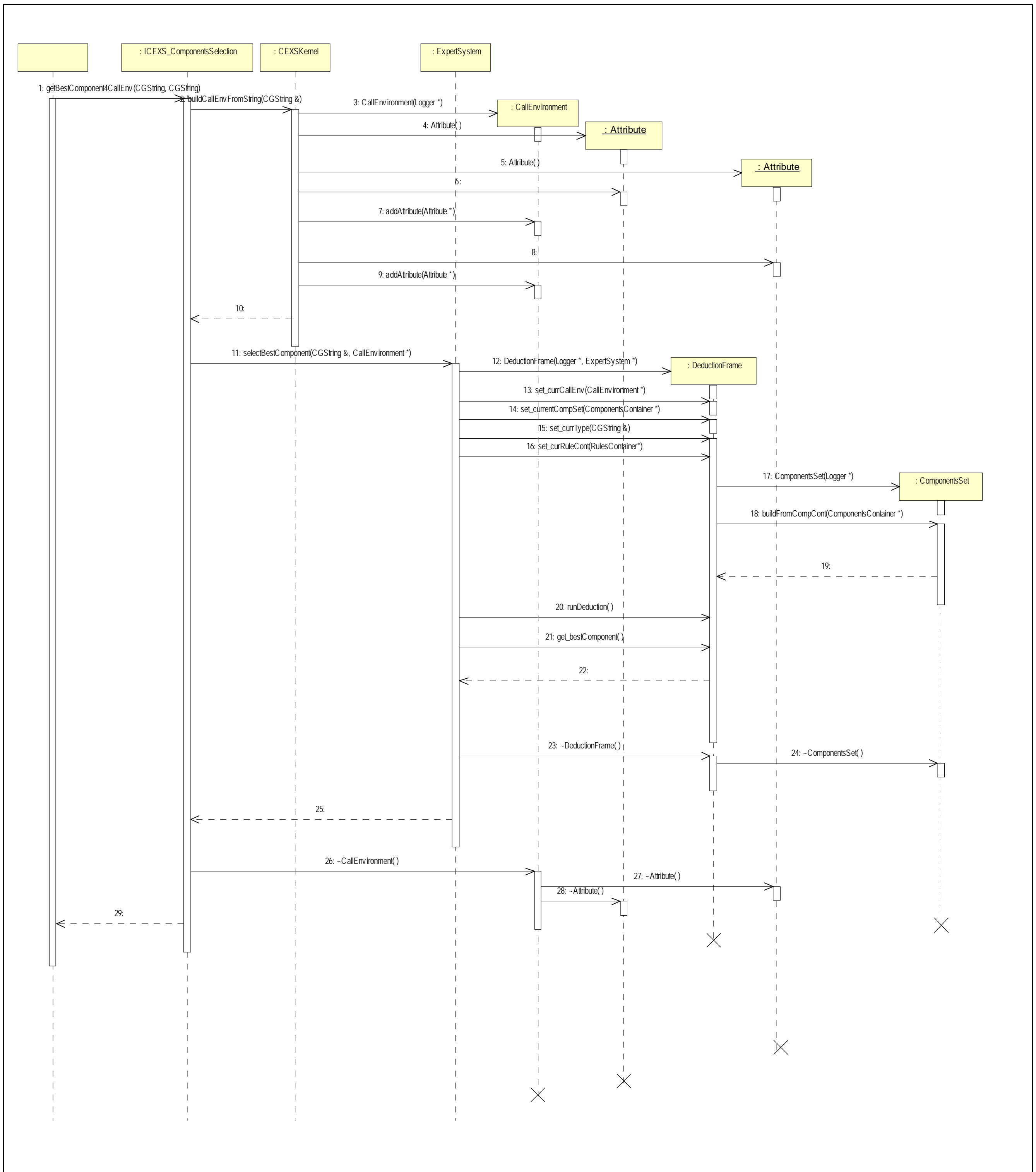


Sequence description:

- Message no.: 1 – `getAllComponents4Type(CGString)`
  Documentation: A type of searched components is passed by this message.
- Message no.: 2 – `getAllComponents4Type(CGString)`
  Documentation: Appropriate method of ComponentsContainer is called.
- Message no.: 3
  Documentation: A list of founded components is returned; it may be empty.
- Message no.: 4
  Documentation: The list is encoded to a transcendentally transport and is returned to a caller.

3.3.9. Sequence Diagram: Best component selection process

This diagram presents the best component selection process.


: ICEXS_ComponentsSelection
: CEXSKernel
: ExpertSystem
: CallEnvironment
: Attribute
: Attribute
: DeductionFrame
: ComponentsSet
1: getBestComponent4CallEnv(CGString, CGString)
2: buildCallEnvFromString(CGString &)
3: CallEnvironment(Logger *)
4: Attribute( )
5: Attribute( )
6:
7: addAttribute(Attribute *)
8:
9: addAttribute(Attribute *)
10:
11: selectBestComponent(CGString &, CallEnvironment *)
12: DeductionFrame(Logger *, ExpertSystem *)
13: set_currCallEnv(CallEnvironment *)
14: set_currentCompSet(ComponentsContainer *)
15: set_currType(CGString &)
16: set_curRuleCont(RulesContainer*)
17: ComponentsSet(Logger *)
18: buildFromCompCont(ComponentsContainer *)
19:
20: runDeduction( )
21: get_bestComponent( )
22:
23: ~DeductionFrame( )
24: ~ComponentsSet( )
25:
26: ~CallEnvironment( )
27: ~Attribute( )
28: ~Attribute( )
29:

Sequence description:

– Message no.: 1 – `getBestComponent4CallEnv(CGString, CGString)`
Documentation: Some client calls a method for selection of the best component. This call carries the type of component as a string parameter and also the call-environment encoded to a string. This call-environment has to be decoded and the instance of the CallEnvironment class should be created.

– Message no.: 2 – `buildCallEnvFromString(CGString &)`
Documentation: The Interface calls a special CEXS kernel method, which builds an instance of the CallEnvironment class from the string and passes the call-environment encoded to string.

– Message no.: 3 – `CallEnvironment(Logger *)`
Documentation: After call-environment parsing, the kernel creates the instance of the CallEnvironment class and sets the own Logger as a logger for this class instance.

– Message no.: 4 – `Attribute( )`
Documentation: After attributes parsing, the Attribute class instance is created.

– Message no.: 5 – `Attribute( )`
Documentation: After attributes parsing, the Attribute class instance is created.

– Message no.: 6
Documentation: Setting value of attribute is done here.

– Message no.: 7 – `addAttribute(Attribute *)`
Documentation: The just created attribute is added to the CallEnvironment.

– Message no.: 8
Documentation: Setting value of attribute is done here.

– Message no.: 9 – `addAttribute(Attribute *)`
Documentation: The just created attribute is added to the CallEnvironment.

– Message no.: 10
Documentation: The created call-environment is returned.

– Message no.: 11 – `selectBestComponent(CGString &, CallEnvironment *)`
Documentation: The type of searched component and the component-specialization is passed to the ExpertSystem.

– Message no.: 12 – `DeductionFrame(Logger *, ExpertSystem *)`
Documentation: An instance of DeductionFrame is created. ExpertSystem passes the pointer to its own Logger object.

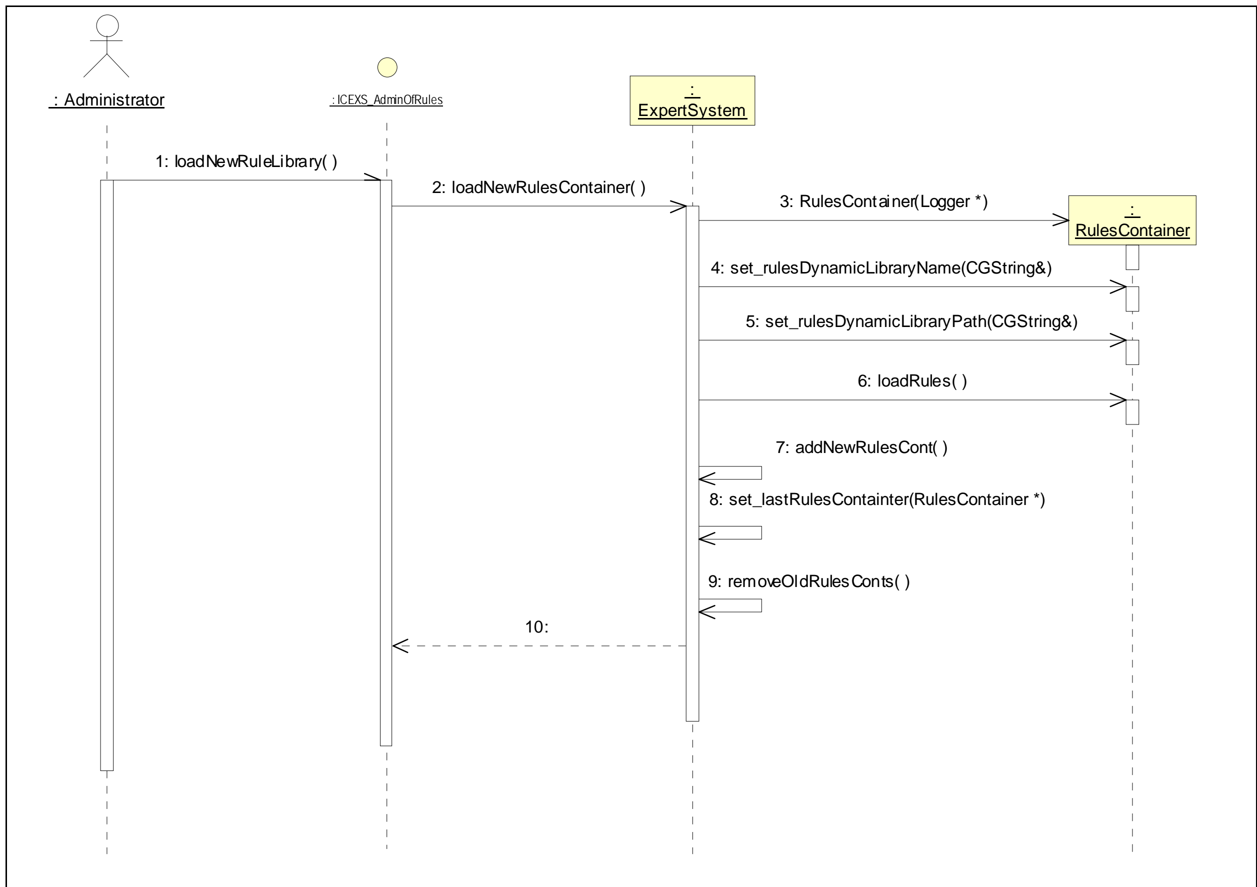
– Message no.: 13 – `set_currCallEnv(CallEnvironment *)`
Documentation: CallEnvironment is obtained by Expert System as a parameter of the selectBestComponent method and it is passed further.

– Message no.: 14 – `set_currentCompSet(ComponentsContainer *)`
Documentation: The ExpertSystem knows the instance of the ComponentsContainer which is set at the beginning. This message passes its pointer to the created DeductionFrame.

- Message no.: 15 – `set_currType(CGString &)`
  Documentation: The ExpertSystem obtained the type of searched component as a parameter of selectBestComponent method and it passes its further here.
- Message no.: 16 – `set_curRuleCont(RulesContainer*)`
  Documentation: The ExpertSystem knows the newest RulesContainer, because the pointer to this structure is stored in lastRulesContainter field. This field is set during the rules loading process. This pointer is now passed to the DeductionFrame.
- Message no.: 17 – `ComponentsSet(Logger *)`
  Documentation: The DeductionFrame creates the instance of the ComponentsSet class and it passes the pointer as the parameter to its own Logger object.
- Message no.: 18 – `buildFromCompCont(ComponentsContainer *)`
  Documentation: The state of the ComponentSet is built on the basis of current state of the ComponentsContainer, which is passed here is a parameter.
- Message no.: 19
  Documentation: An execution code is returned here.
- Message no.: 20 – `runDeduction( )`
  Documentation: The deduction process is fired. The details of the deduction process are presented in the sequence diagram 'Deduction process'.
- Message no.: 21 – `get_bestComponent( )`
  Documentation: The outcome of the deduction is stored in the bestComponent attribute and a current value of this attribute is taken here.
- Message no.: 22
  Documentation: The pointer to the best component is passed here.
- Message no.: 23 – `~DeductionFrame( )`
  Documentation: The instance of DeductionFrame is no longer necessary and it should be destroyed.
- Message no.: 24 – `~ComponentsSet( )`
  Documentation: The instance of the ComponentsSet is no longer necessary, thus it is removed.
- Message no.: 25
  Documentation: The found best component is returned here.
- Message no.: 26 – `~CallEnvironment( )`
  Documentation: The instance of the CallEnvironment is no longer necessary and it is destroyed.
- Message no.: 27 – `~Attribute( )`
  Documentation: The instance of all Attributes used for the construction of the CallEnvironment object are no longer necessary and are removed from the memory.
- Message no.: 28 – `~Attribute( )`
  Documentation: The instance of all Attributes used to the construct of the CallEnvironment object are no longer necessary and are removed from the memory.
- Message no.: 29
  Documentation: The handle to the best component is encoded and passed to the caller.

3.3.10. Sequence Diagram: Rules interchanging

The sequence of the rules interchanging process is presented below. This process is initialized by the administrator, which calls the loadNewRuleLibrary method of the ICEXS_AdminOfRules class.



Sequence description:

- Message no.: 1 – `loadNewRuleLibrary( )`
  Documentation: The administrator calls the message of the ICEXS_AdminOfRules interface.

- Message no.: 2 – `loadNewRulesContainer( )`
  Documentation: Implementation of the interface calls the appropriate ExpertSystem class method.

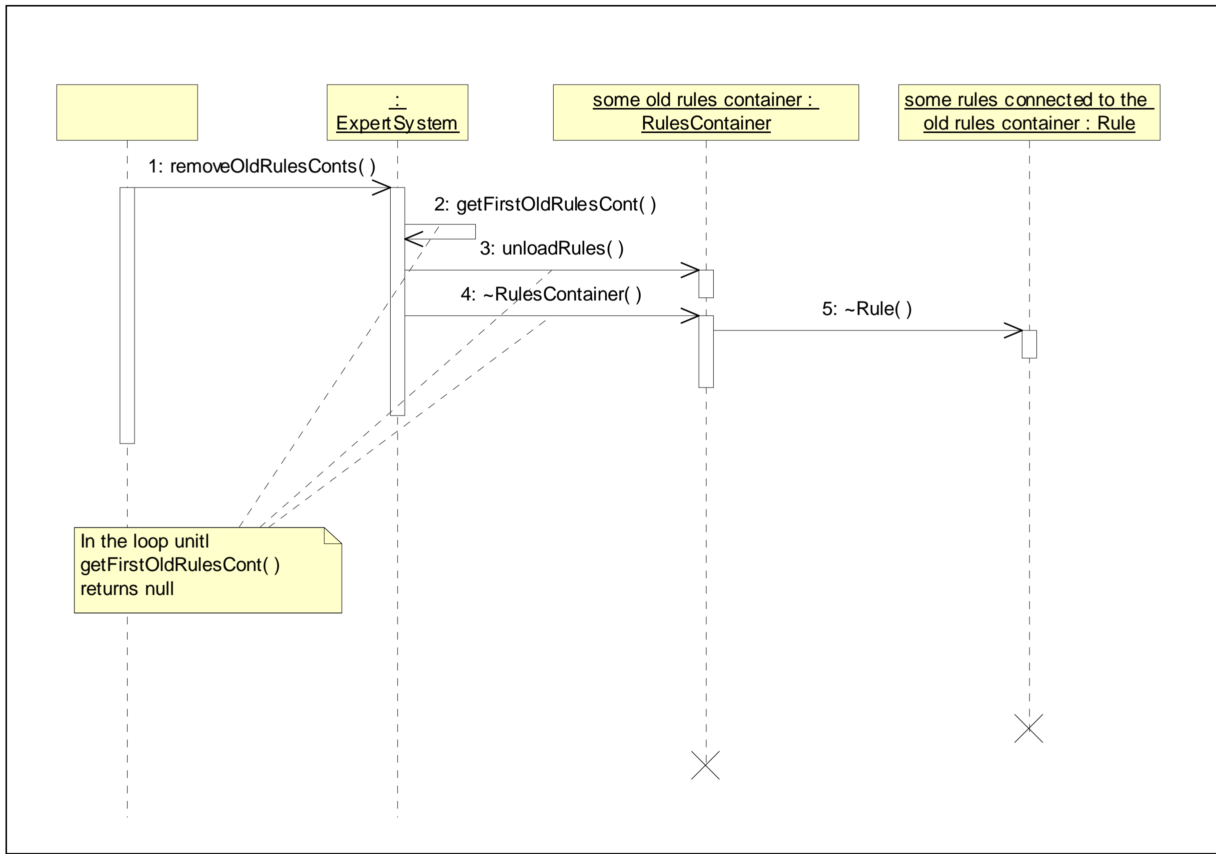
- Message no.: 3 – `RulesContainer(Logger *)`
  Documentation: This message calls the RulesContainer creation process and passes the pointer to the own ExpertSystem Logger.

- Message no.: 4 – `set_rulesDynamicLibraryName(CGString&)`
  Documentation: The ExpertSystem passes the name of the library, which is kept by the rulesDynamicLibraryName field.

- Message no.: 5 – `set_rulesDynamicLibraryPath(CGString&)`
  Documentation: The ExpertSystem passes the path to the dynamic-linking rules library, which is kept by the rulesDynamicLibraryPath field.
- Message no.: 6 – `loadRules( )`
  Documentation: After preparation of the RulesContainer object a linking process is fired. Details of this process are presented in the 'Rules loading' sequence diagram.
- Message no.: 7 – `addNewRulesCont( )`
  Documentation: When a new RuleContainer is created, it is added to the list of active RuleContainers (ruleConts).
- Message no.: 8 – `set_lastRulesContainter(RulesContainer *)`
  Documentation: The new RulesContainer is set as the last one to be used by the DeductionFrames during future deductions.
- Message no.: 9 – `removeOldRulesConts( )`
  Documentation: This message removes old RulesContainer. If some RulesContainer is in use, it waits until it is freed.
- Message no.: 10
  Documentation: An execution code is returned.

3.3.11. Sequence Diagram: Removing old RulesContainers

When a new RulesContainer is added then the old ones are removed. This diagram shows the removing process.

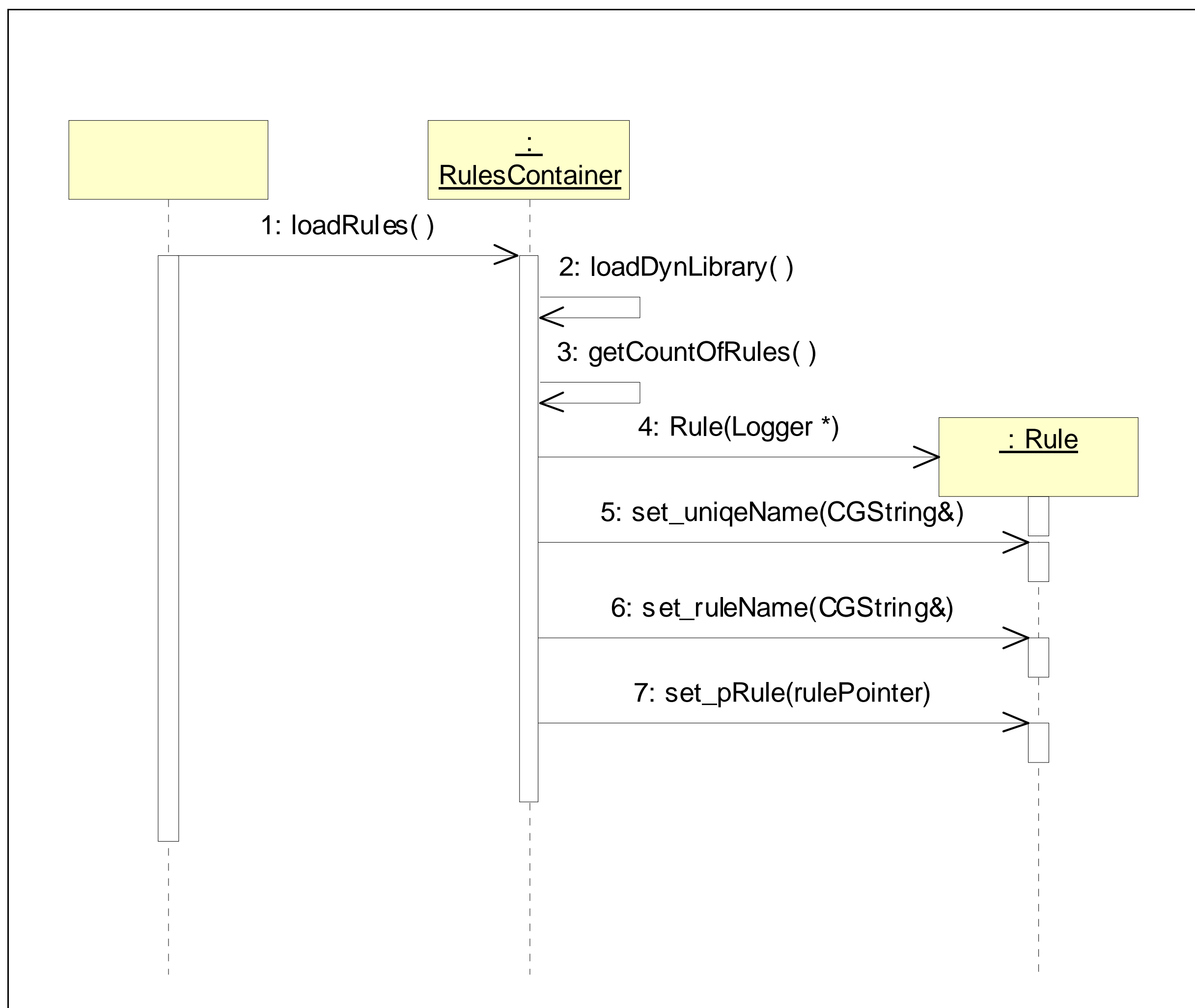


Sequence description:

- Message no.: 1 – `removeOldRulesConts( )`
  Documentation: Some class calls the removeOldRulesConts( ) method.
- Message no.: 2 – `getFirstOldRulesCont( )`
  Documentation: This operation is done in the loop until this method returns 'null'. It returns the first old RulesContainer, if it does not exist it returns 'null'.
- Message no.: 3 – `unloadRules( )`
  Documentation: This operation is done in the loop. It unloads rules container library form the memory. This operation waits until the RuleContainer is freed, if it is in use.
- Message no.: 4 – `~RulesContainer( )`
  Documentation: This instance of rules container is not needed yet, thus it is removed from the memory.
- Message no.: 5 – `~Rule( )`
  Documentation: Each instance of Rule class connected to the RulesContainer is removed.

3.3.12. Sequence Diagram: Rules loading

This diagram depicts the loading process of the rules used during deduction process. The rules are stored in the dynamic linking library, which is compiled using the g++ compiler. Each rule has own representation in memory - an instance of the Rule class. The messages 4 to 7, which are depicted in this diagram, are repeated in a loop - on loop iteration for one rule.



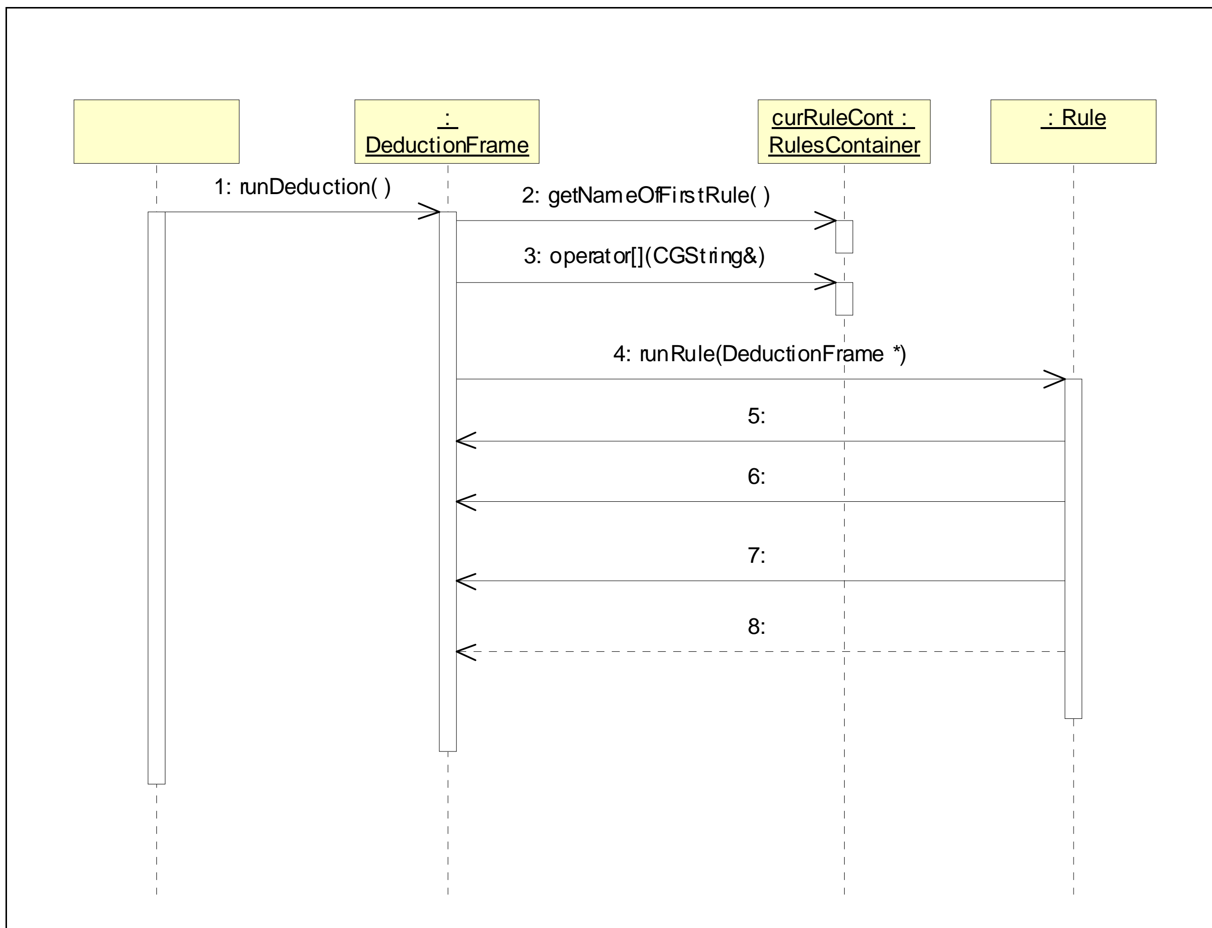
Sequence description:

- Message no.: 1 – `loadRules( )`
  Documentation: Something calls the loadRules method.
- Message no.: 2 – `loadDynLibrary( )`
  Documentation: Using the path and library name the dynamic-linking library is loaded and linked. All operational parameters are taken (e.g., the lib name, lib version, lib state, usageCounter, and library handle) and appropriate fields are set.

- Message no.: 3 – `getCountOfRules( )`
  Documentation: The number of rules stored in rules library is taken.
- Message no.: 4 – `Rule(Logger *)`
  Documentation: The new instance of the class Rule is created and the Logger assigned to RulesContainer is passed as the Logger for the Rule.
- Message no.: 5 – `set_uniqeName(CGString&)`
  Documentation: The unique name of the rule is read and set
- Message no.: 6 – `set_ruleName(CGString&)`
  Documentation: The userfreandly name of the rule is read and set.
- Message no.: 7 – `set_pRule(rulePointer)`
  Documentation: The pointer to the rule is read and set.

3.3.13. Sequence Diagram: Deduction process

The best component is selected during the deduction process. The deduction process is performed by the DeductionFrame, which keeps all data necessary for the deduction. The deduction rules manage this data and as the result of the deduction the pointer to the best component is set using the field 'bestComponent'.



Sequence description:

- Message no.: 1 – `runDeduction( )`
  Documentation: Usually the ExpertSystem instance class calls the runDeduction method. This message is sent only after appropriate DeductionFrame preparation.
- Message no.: 2 – `getNameOfFirstRule( )`
  Documentation: One of the first thing, which is done by the runDeduction method is to take name of the first rule from the curRuleCont (the current rule container). This name is necessary to obtain the first rule. This name may be stored in some local variable for example nextRuleName.
- Message no.: 3 – `operator[](CGString&)`
  Documentation: This is the loop beginning. This is run until nextRuleName is equal to the empty string. This operator is called with the nextRuleName, and as a result it returns the pointer to the nextRule.
- Message no.: 4 – `runRule(DeductionFrame *)`
  Documentation: This message is passed to the instance of the rule obtained by the operator [] form the curRuleCont. This message fires the rule and passes the pointer to the current deduction frame (i.e. the rule under study). This message is repeated in the loop.
- Message no.: 5
  Documentation: When the rule works then it modifies the instance of the deduction frame, which called the rule. Especially, it modifies currentCompSet, works with the currCompCont and with the currCallEnv. The final rule sets the value of the bestComponent attribute.
- Message no.: 6
  Documentation: Modification of some DeductionFrame attributes.
- Message no.: 7
  Documentation: Modification of some DeductionFrame attributes.
- Message no.: 8
  Documentation: As the result of the runDeduction method the name of the next rule is passed. If this is the last rule then it returns the empty string. The result is assigned to the nexRuleName local variable. This assignment is the last operation in the loop.

3.4. INTERNAL COMPONENT – TOOLSLIB (3.4)

3.4.1. Class Diagram – Main

This diagram depicts connections between classes that are contained in the TOLB library and relationships between them.

3.4.2. Short Description of Classes

3.4.2.1. Class - Logger

Logger is a universal class to logging support. It should be compiled to the library and should be reused by different executables.

3.4.2.2. Class - SemaphoreQueuing

This is the binary semaphore, which is used to make critical regions. It gives three operations: lock, unlock and islock.

3.4.2.3. Class - SemaphoreCounting

This is a special kind of the semaphore. This semaphore is used to protect some resources but it divides resource usage into two categories, the usage in reading mode and the usage in writing mode (the read process and write process). There is a possibility of concurrently usage of the protected resource in the reading mode by many readers. However, there is no way to share the protected resource between readers and writers and many writers. This class provides a counter of the readers using concurrently the protected resource. If this counter is equal to zero, then no readers can use the protected resource. In situation when no reader nor writer uses the resource then it could be used in the writing mode otherwise the writer process must wait. If any writer waits for the access to the resource then new readers must wait until the writer finishes the work.

It works in the following way: If there are only read processes in the critical section, they all read simultaneously. Once a writing process demands exclusive access to data the waitingWriters variable is incremented and from this moment no reading process can join the currently reading processes (they are queued on the readAllowed condition). The first of the waiting processes waits for all the currently reading processes to finish and then enters critical section and changes the data. When it finishes its work it allows all of the waiting read processes to access the data.

3.4.2.4. Class - CGFloat

The object-representation of the numeric-type 'floatCG'.

3.4.2.5. Class - CGInteger

The object-representation of the numeric-type 'integerCG'.

3.4.2.6. Class - CGLong

The object-representation of the numeric-type 'longCG'.

3.4.2.7. Class - CGObject

This is an abstract class representing any object.

3.4.2.8. Class - CGString

The CGString class represents string type. Actually, this class is built around string class coming from STL library. CGString has the filed 'value', which is the 'string' object and this filed keeps all necessary information.

3.4.2.9. Class - CGNumber

The abstract class representing numbers objects in general. All other number classes as CGLong, CGInteger etc. inherits form this class.

3.4.2.10. Class - CGLongSync

This is a synchronized version of CGLong class. In this class (using) semaphores all operations are suited to multithreading work.

3.4.2.11. Class - CGIntegerSync

The CGIntegerSync class is internally synchronized version of the CGInteger class. It can works in multithread environment and it automatically synchronizes an access to its protected resources. Additionally, this class contains special method (readLock, writeLock etc.), which allows block access to protected resources for longer controlled time, and that allows making some transactions spread over a few objects.

3.4.2.12. Class - CGFloatSync

The CGFloatSync class is internally synchronized version of the CGFloat class. It can works in multithread environment and it automatically synchronizes an access to its protected resources. Additionally, this class contains special method (readLock, writeLock etc.), which allows block access to protected resources for longer controlled time, and that allows making some transactions spread over a few objects.

3.4.2.13. Class - CGStringSync

The CGStringSync class is internally synchronized version of the CGString class. It can works in multithread environment and it automatically synchronizes an access to its protected resources. Additionally, this class contains special method (readLock, writeLock etc.), which allows block access to protected resources for longer controlled time, and that allows making some transactions spread over a few objects.

3.4.2.14. Class - ConditionVariable

This class is an implementation of standard concurrent primitive the Condition Variable.

4. PRODUCT TESTING

The program has been exhaustively tested in several platforms and in the CrossGrid Testbed. The output of the program was tested by experts.

Since the beginning of the project, several tests have been performed. As results of these tests, several improvements to the code have been done.

Finally, numerical results of these tests are available in different internal documents of the project and in a set of published papers.

5. CONTACT INFORMATION AND CREDITS

For more information ore problems, please contact

Lukasz Dutka Cyfronet UST-AGH
dutka@agh.edu.pl

6. THE EDG LICENSE AGREEMENT

Copyright (c) 2005 CrossGrid. All rights reserved.

This software includes voluntary contributions made to the CrossGrid Project. For more information on CrossGrid, please see http://www.eu-crossgrid.org.

Installation, use, reproduction, display, modification and redistribution of this software, with or without modification, in source and binary forms, are permitted. Any exercise of rights under this license by you or your sub-licensees is subject to the following conditions:

1. Redistributions of this software, with or without modification, must reproduce the above copyright notice and the above license statement as well as this list of conditions, in the software, the user documentation and any other materials provided with the software.

2. The user documentation, if any, included with a redistribution, must include the following notice:

"This product includes software developed by the CrossGrid Project (http://www.eu-crossgrid.org)."

Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the software itself.

3. The names "CrossGrid" and "CG" may not be used to endorse or promote software, or products derived therefrom, except with prior written permission by cgoffice@cyfronet.krakow.pl.

4. You are under no obligation to provide anyone with any bug fixes, patches, upgrades or other modifications, enhancements or derivatives of the features, functionality or performance of this software that you may develop. However, if you publish or distribute your modifications, enhancements or derivative works without contemporaneously requiring users to enter into a separate written license agreement, then you are deemed to have granted participants in the CrossGrid Project a worldwide, non-exclusive, royalty-free, perpetual license to install, use, reproduce, display, modify, redistribute and sub-license your modifications, enhancements or derivative works, whether in binary or source code form, under the license conditions stated in this list of conditions.

5. DISCLAIMER

THIS SOFTWARE IS PROVIDED BY THE CROSSGRID PROJECT AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE OR USE ARE DISCLAIMED. THE CROSSGRID PROJECT AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT.

6. LIMITATION OF LIABILITY

THE CROSSGRID PROJECT AND CONTRIBUTORS SHALL HAVE NO LIABILITY TO LICENSEE OR OTHER PERSONS FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA OR PROFITS, OR BUSINESS INTERRUPTION, HOWEVER CAUSED AND ON ANY THEORY OF CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.