



# PROTOTYPE DOCUMENTATION

## FIRST PROTOTYPE

### WP3.4

---

Document Filename: **CG3.4-D33-v1.1-CYF021-PrototypeDescription.doc**

Work package: **WP3.4**

Partner(s): **CYFRONET**

Lead Partner: **CYFRONET**

Config ID: **CG3.4-D33-v1.0-CYF021-PrototypeDescription**

Document classification: **PUBLIC**

---



Abstract: This document is intended to provide detail description of current status of Task 3.4 work. Amongst other things it presents architecture of the provided software as D3.3 deliverable.



**Delivery Slip**

	<b>Name</b>	<b>Partner</b>	<b>Date</b>	<b>Signature</b>
<b>From</b>	WP3, task 4	CYFRONET	19 Feb 2003	
<b>Verified by</b>				
<b>Approved by</b>				

**Document Log**

<b>Version</b>	<b>Date</b>	<b>Summary of changes</b>	<b>Author</b>
1.0	14/01/2002	Inception	Jacek Kitowski, Renata Słota, Darin Nikolow, Łukasz Dutka
1.1	19/02/2002	Corrections after review	Jacek Kitowski, Renata Słota, Darin Nikolow, Łukasz Dutka

---

## CONTENTS

<b>EXECUTIVE SUMMARY.....</b>	<b>6</b>
<b>1. INTRODUCTION.....</b>	<b>7</b>
1.1. PURPOSE .....	7
1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS .....	7
<b>2. REFERENCES.....</b>	<b>9</b>
2.1. BIBLIOGRAPHY .....	9
2.2. PRODUCTION DOCUMENTATION .....	10
2.3. SOURCE CODE.....	12
2.4. CONTACT INFORMATION.....	12
<b>3. IMPLEMENTATION STRUCTURE .....</b>	<b>13</b>
3.1. PROVIDED COMPONENTS .....	13
3.2. PROVIDED LIBRARIES .....	15
3.3. PROVIDED WEB ELEMENTS.....	16
3.4. PROVIDED CONFIGURATION FILES.....	17
3.5. RELATIONSHIP DIAGRAMS BETWEEN PROVIDED COMPONENTS.....	18
3.6. DEPLOYMENT DIAGRAM .....	23
3.7. DIFFERENCES BETWEEN IMPLEMENTED AND PRESENTED IN SDD (D3.2 DOCUMENT) MODEL .....	24
<b>4. PROTOTYPE FUNCTIONALITY .....</b>	<b>26</b>
4.1. FEATURES OF PROVIDED STEL COMPONENT.....	27
4.2. FEATURES OF PROVIDED CEXS COMPONENT .....	28
4.3. FEATURES OF PROVIDED DAES COMPONENT .....	29
4.4. FEATURES OF PROVIDED ‘ESTIMATE’ TYPE CECOMPONENTS .....	31
4.5. FEATURES OF DEMO CLIENT WEB PAGE.....	31
<b>5. USER MANUAL.....</b>	<b>33</b>
5.1. DEPENDENCIES .....	33
5.1.1. Storage Node .....	33
5.1.2. HSM UniTree Server .....	33

5.1.3. Web Server.....	33
5.1.4. Tester Desktop .....	33
5.2. INSTALLATION .....	34
5.2.1. Storage Node .....	34
5.2.2. HSM UniTree Server .....	34
5.2.3. Web Server.....	35
5.2.4. Tester Desktop .....	36
5.3. RUNNING .....	36
5.3.1. Storage Node .....	36
5.3.2. HSM UniTree Server .....	37
5.3.3. Web Server.....	37
5.3.4. Tester Desktop .....	37
5.4. INTERFACE DESCRIPTION .....	37
5.4.1. Component: Component-Expert Subsystem (3.4) – Interface: ICEXS_AdminOfComponents.....	37
5.4.2. Component: Component-Expert Subsystem (3.4) – Interface: ICEXS_AdminOfRules.....	39
5.4.3. Component: Component-Expert Subsystem (3.4) – Interface: ICEXS_ComponentsSelection .....	39
5.4.4. Component: Data Access Estimator (3.4) – Interface: IDAES_AccessEstimation .....	41
5.4.5. Component: Storage Element (3.4) – Interface: ISTEEL_Configuration .....	42
5.5. API.....	45
5.5.1. Class : CEXSClient.....	45
5.5.2. Class : DAESClient.....	49
5.5.3. Class : STELClient.....	51
<b>6. INTERNAL TESTS.....</b>	<b>56</b>
<b>7. ISSUES.....</b>	<b>58</b>

## EXECUTIVE SUMMARY

Please provide a brief description of the document contents.

This document contains a general description of the provided prototype as D3.3. Amongst other things, it focuses on functionality, which will be available in Month 12, and is discussed in detail in section 4. However this document contains other sections:

- References, which is intended for collecting references of all the relevant external documents
- Implementation Structure, which presents the general architecture of the provided solution. This section does not detail the internal architecture of components, but where necessary it provides appropriate references
- User Manual, which collects all information required to run the provided software
- Internal Tests, which provides information about production aspects
- Issues, which provides additional information such as known bugs etc.

The content of the document is at mid technical level, but the issues which might require additional, more detailed technical description are referred to external documents.

## 1. INTRODUCTION

### 1.1. PURPOSE

Please define the purpose of the prototype and summarize its intended functionality.

The activity of task 3.4 is mainly focussed on storage and related problems. The prototype developed by task 3.4 provides a flexible architecture for storage nodes and storage centres. This architecture allows for a very flexible optimization and control of the internal behaviour of storage centres. It is fully adaptable for future purposes, but is currently used only for organizing the estimation of data access latency and bandwidth of internal storage nodes. However, the provided solution could be directly used for other purposes. In the very near future it will be used for organizing data flow in storage nodes.

### 1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS

CEA	Component Expert Architecture
CEComponent	Components compatible with CEA, which have specialization and type. In the CrossGrid project CEcomponents are compatible with the CEXS module. This means they have interfaces which allow CEXS to read their specialization.
CEComponent Set	The set of registered components in CEXS
CEXS	Component-Expert Subsystem. The component which prototype will be provided as D3.3
CGI	Common Gateway Interface. The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.
CLLB	Common Client Library.
D3.1	Deliverable 3.1
D3.2	Deliverable 3.2
D3.3	Deliverable 3.3
DAES	Data Access Estimator. The component which prototype will be

	provided as D3.3
DCWP	Demo Client Web Page. A WWW service which allows an easy way to test provided components.
Designer	A person who designs Task 3.4's software model.
Developer	A person who codes designed classes.
EDG	European Data Grid
gSOAP	Tools for generation of the SOAP communication layer for C++ programs.
PID	Process Identifier
Reviewer	A person who reads the code and tries to find bugs by reading only.
SDD	Software Design Description. The document, D3.2.
SOAP	Simple Object Access Protocol
SOAPCG	The library collecting all gSOAP specific functions
SRS	Software Requirements Specification. The document D3.1.
STEL	Storage Element. One of the D3.3 prototype components.
Tester	A person who takes code prepared by a developer in order to find bugs. Testing the code often requires writing additional checking programs.
UEST	UniTree HSM Access Estimator

## 2. REFERENCES

### 2.1. BIBLIOGRAPHY

1. 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>
2. Kitowski J. , Dutka Ł., Słota R., Nikolow D., “Task 3.4 SDD Software Design Description - Final” – D3.2 [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/Docs/CG3.4-D3.2-v2.0-CYF021-OptDataAccDesgRpt.pdf](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/Docs/CG3.4-D3.2-v2.0-CYF021-OptDataAccDesgRpt.pdf)
3. “Development of Grid Environment for Interactive Applications – Annex 1 ” [http://www.cyf-kr.edu.pl/crossgrid/CrossGridAnnex1\\_v31.pdf](http://www.cyf-kr.edu.pl/crossgrid/CrossGridAnnex1_v31.pdf)
4. Dutka, Ł., and Kitowski, J., „Component-Expert Architecture as Flexible Environment for Selection of Data-handlers and Data-Access-Estimators in CrossGrid”, Cracow Grid Workshop. [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/Docs/CGW2002.pdf](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/Docs/CGW2002.pdf)
5. Dutka Ł and J. Kitowski, “Application of Component-Expert Technology for Selection of Data-Handlers in CrossGrid”, in: D. Kranzlmüller, 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.
6. Santiago Gonzalez de la Hoz et. al., “Test and Integration Process Description Deliverable D3.3” <http://www.eu-crossgrid.org/m6-drafts-login/M12deliverables-drafts/CG3.5-D3.3-v1.0-CSIC021-TestIntegrationPrototype.doc>
7. SOAP <http://www.w3.org/TR/soap>
8. UML <http://www.omg.org/uml/>
9. STL <http://www.stlport.org/>
10. POSIX <http://www.pasc.org/abstracts/posix.htm>
11. Clean Room Software Engineering <http://www.rspa.com/spi/cleanroom.html>

## 2.2. PRODUCTION DOCUMENTATION

This section contains the documentation resulting from the developing process, and often provides information at a higher technical level.

12. Dutka Ł., “CEXS Design Description SDD of « Component-Expert Subsystem (CEXS) » – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/CEXSdocs/CG-3.4-MDL-0001-CEXSDesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/CEXSdocs/CG-3.4-MDL-0001-CEXSDesignDescriptionRpt.doc)
13. Dutka Ł., “DAES Design Description SDD of « Component-Expert Subsystem (DAES) » – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/DAESdocs/CG-3.4-MDL-0001-DAESDesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/DAESdocs/CG-3.4-MDL-0001-DAESDesignDescriptionRpt.doc)
14. Dutka Ł., “STEL Design Description SDD of « Component-Expert Subsystem (STEL) » – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/STELdocs/CG-3.4-MDL-0001-STELDesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/STELdocs/CG-3.4-MDL-0001-STELDesignDescriptionRpt.doc)
15. Dutka Ł., “TOLB Design Description SDD of « Component-Expert Subsystem (TOLB) » – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/TOLBdocs/CG-3.4-MDL-0001-TOLBDesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/TOLBdocs/CG-3.4-MDL-0001-TOLBDesignDescriptionRpt.doc)
16. Dutka Ł., “CLLB Design Description SDD of « Component-Expert Subsystem (CLLB) » – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/CLLBdocs/CG-3.4-MDL-0001-CLLBDesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/CLLBdocs/CG-3.4-MDL-0001-CLLBDesignDescriptionRpt.doc)
17. Dutka Ł., “DCWP Design Description -- SDD of « Demo Client Web Page (DCWP) »” – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/DCWPdocs/CG-3.4-DCWPDesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/DCWPdocs/CG-3.4-DCWPDesignDescriptionRpt.doc)
18. Dutka Ł., “CEXSClient.CGI Design Description -- SDD of « Demo Client Web Page (CEXSClient.CGI) »” – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/DCWPdocs/CG-3.4-CEXSClient.cgi-DesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/DCWPdocs/CG-3.4-CEXSClient.cgi-DesignDescriptionRpt.doc)
19. Dutka Ł., “DAESClient.CGI Design Description -- SDD of « Demo Client Web Page (DAESClient.CGI) »” – [http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/DCWPdocs/CG-3.4-DAESClient.cgi-DesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/DCWPdocs/CG-3.4-DAESClient.cgi-DesignDescriptionRpt.doc)

- 
20. Dutka Ł., “STELClient.CGI Design Description -- SDD of « Demo Client Web Page (STELClient.CGI) »” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/DCWPDOcs/CG-3.4-STELClient.cgi-DesignDescriptionRpt.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/DCWPDOcs/CG-3.4-STELClient.cgi-DesignDescriptionRpt.doc)
  21. Dutka Ł., “Expert System Specification – Specificaiton of Attributes and Rules for Task 3.4” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/CEXSDOcs/CG-3.4-ExpertSystemSpec.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/CEXSDOcs/CG-3.4-ExpertSystemSpec.doc)
  22. Buczak W., Dutka Ł., “CETM Specialization” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/CETMDOcs/CETM\\_Specialization.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/CETMDOcs/CETM_Specialization.doc)
  23. Buczak W., Dutka Ł., “Communication with CEComponent” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/CETMDOcs/CETM\\_Communication.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/CETMDOcs/CETM_Communication.doc)
  24. Nowakowski P., Dutka Ł., “CEXS Configuration File -- Short Description” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/CEXSDOcs/CEXSConfigurationFile.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/CEXSDOcs/CEXSConfigurationFile.doc)
  25. Buczak W., Dutka Ł., “Components Container Configuration File -- Short Description” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/CEXSDOcs/CompContConfigurationFile.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/CEXSDOcs/CompContConfigurationFile.doc)
  26. Nowakowski P., Dutka Ł., “STEL Configuration File -- Short Description” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/STELDOcs/STELConfigurationFile.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/STELDOcs/STELConfigurationFile.doc)
  27. Karbowski J., Dutka Ł., “Storage Configuration File -- Short Description” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/STELDOcs/StorageConfigurationFile.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/STELDOcs/StorageConfigurationFile.doc)
  28. Nowakowski P., Dutka Ł., “DAES Configuration File -- Short Description” – [http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/DAESDOcs/DAESConfigurationFile.doc](http://gridportal.fzk.de/cgi-bin/viewcvcs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/DAESDOcs/DAESConfigurationFile.doc)
  29. Dutka Ł., “Code Review Guide”, <http://gridportal.fzk.de/cgi->

[bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3\\_4-expert/doc/Docs/Code%20review%20guide.doc](bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_4-expert/doc/Docs/Code%20review%20guide.doc)

### 2.3. SOURCE CODE

The latest version of the Task 3.4 source code is located in Karlsruhe CVS and is obtainable from the following web page: <http://gridportal.fzk.de/projects/cg-wp3-4/>.

The whole code is divided onto independent modules (components), and each module has its own technical documentation specifying in detail its internal architecture.

Please provide references to properly annotated source code and any external documentation components, especially detailed class descriptions as generated by JavaDoc (or other automated documentation tools in accordance with the CrossGrid Standard Operating Procedures document **CG5.2-D5.2.3-v1.0-CYF020-StandardOperatingProcedures**). Provide a codename for the prototype so that it can be easily referred to.

### 2.4. CONTACT INFORMATION

The code provided by Task 3.4 is prepared by Cyfronet. The task leader is Jacek Kitowski [kito@agh.edu.pl](mailto:kito@agh.edu.pl).

### 3. IMPLEMENTATION STRUCTURE

[OPEN1]

[REPEAT2]

#### 3.1. PROVIDED COMPONENTS

This section shows short description of all provided by Task 3.4 elements, which are components. The presented items below are ordered in the alphabetical order.

- [REPEAT3]**Component:** “CEComponent - Type = Estimate” is ANSI C++ project. This is the template of an 'Estimate' type CEComponent, which is compatible with Component Expert Architecture (for more information about Component Expert Architecture see SRS [1] section 2.1). 'Estimate' type components are intended to estimate data access factors (like data access latency or access bandwidth) to particular data objects (i.e. files, data bases etc.). This type of components estimates data access latency or access bandwidth to particular data object. Estimation of data access factors to data stored on TSS requires a sophisticated algorithm; its description was done in SRS document section 2.1.3. 'DataID' attribute passed to the component by a call-environment using ICECP\_ControlParameters interface identifies data objects. This component is based on the 'CEComponent - Template' and it is the template of all 'Estimate' type CEcomponents, which will be used in CrossGrid. Additionally, it introduces ICEES\_Estimate interface, which provides a special operation for data access estimation.
  
- [REPEAT4]**Component:** “CEXSClientTest” is ANSI C++ project. This is a sample C++ client for CEXS component. It tests all operations offered by interfaces of CEXS. This component has been prepared as a helpful tool for Testbed.
  
- [REPEAT5]**Component:** “Component-Expert Subsystem (3.4)” is ANSI C++ project. This component realizes the proposed Component-Expert Architecture (CEA). It has many responsibilities like: registering and managing a set of CEComponents; providing a deduction facilities, which can be modified if necessary and finally handling requests of the best component selection for a particular context passed as call-environment. This component

should be located on each Storage Element. The provided version can use STEL component to improve deduction quality by obtaining current storage node configuration. The abbreviation of its name is CEXS.

- [REPEAT6]**Component:** “DAESClientTest” is ANSI C++ project. This is a sample C++ client for DEAS component. It tests all operations offered by interfaces of DAES. This component has been prepared as a helpful tool for Testbed. It is similar to CEXSClientTest.
  
- [REPEAT7]**Component:** “Data Access Estimator (3.4)” is ANSI C++ project. Data Access Estimator estimates data access cost for a given data object. It can return internal latency and bandwidth. This component should be located on each Storage Element. It is CEA oriented, thus estimation process is move out to 'Estimate' type CEComponents and the decision which one component is used for a particular estimation is taken by CEXS. In consequence DAES requires access to CEXS component, which typically is installed on the same node. The abbreviation of this component name is DAES.
  
- [REPEAT8]**Component:** “STELClientTest” is ANSI C++ project. This is a sample C++ client for STEL component. It tests all operations offered by interfaces of STEL. This component has been prepared as a helpful tool for Testbed. It is similar to STELClientTest.
  
- [REPEAT9]**Component:** “Storage Element (3.4)” is ANSI C++ project. The Storage Element component is responsible for managing and answering question about current configuration of the storage node on which it works. This component should be deployed on each storage node. In the first prototype, it is used especially by CEXS, however it can be used freely by other components, which can communicate using SOAP protocol.. The abbreviation of this component name is STEL.
  
- [REPEAT10]**Component:** “UniTree HSM Access Estimator (3.4)” is ANSI C++ project. This component is deployed on a particular UniTree HSM controller and estimate data access factors for a particular file. It is used by a specific 'Estimate' type CEComponent, which is

called by DAES (when CEXS decides it is appropriate in a particular context). This CEComponent communicates with "UniTree HSM Access Estimator".

### 3.2. PROVIDED LIBRARIES

This section shows short description of all provided by Task 3.4 elements, which are libraries used by other components. The presented items below are ordered in the alphabetical order.

- [REPEAT11]**Component:** “Common Client Library (3.4)” is realized as ANSI C++ project. Common Client Library (CLLB) contains classes, which allow in easy way to communicate with DAES, STEL and CEXS. In reality, it covers low level gSOAP communication function and provides clients classes for CEXS, STEL and DAES which can be remote.
- [REPEAT12]**Component:** “RulesSet (3.4)” is realized as ANSI C++ project. This component is a dynamic linking library, which contains a set of rules used by expert system in the deduction process. Each rule is a special function containing internal rule logic. The provided set of rules could be freely modified for specific purposes. In the current state there are rules using STEL to improve the quality of component selection by getting external information about current configuration of storage node state related to a particular data object. This component is used by CEXS.
- [REPEAT13]**Component:** “SOAPCG” is realized as ANSI C++ project. This component is a static linking library and is mostly generated by gSOAP tool. It contains gSOAP communication level functions.
- [REPEAT14]**Component:** “ToolsLib (3.4)” is realized as ANSI C++ project. The ToolsLib is a library of helpful tools, as numeric classes, synchronization primitives etc., which should be used by all task 3.4 components. This is the statically linked library, which in addition defines elementary platform independent types as integerCG, boolCG, longCG, floatCG. One of the most interesting solutions provided by this library is the set of the thread safe classes, e.g.,

CGIntegerSync, CGStringSync, etc. These classes provide bunch of base methods and operators, which are thread safe and make programming for the multithread environment as easy as possible. This library was introduced to simplify the programming process and make code more transparent. The abbreviation of this component name is TOLB.

### 3.3. PROVIDED WEB ELEMENTS

This section shows short description of all provided by Task 3.4 elements, which are web pages etc. The presented items below are ordered in the alphabetical order.

- [REPEAT15] **Component:** “CEXSClient.cgi (3.4)” is realized as Web Modeler project. CGI client which communicates with a particular CEXS. It calls required operations and generates result web page containing received results from the called CEXS component. It uses HTML output templates in order to make its internal code independent from a graphical design of whole service. It is used by DCWP as a hyphen between WWW service and CEXS. For a technical details see "DCWP Design Description" document.
- [REPEAT16] **Component:** “DAESClient.cgi (3.4)” is realized as Web Modeler project. This is very similar to CEXSClient.cgi but it connects with DAES. It is used by DCWP as a hyphen between WWW service and DAES. For a technical details see "DCWP Design Description" document.
- [REPEAT17] **Component:** “Demo Client Web Page (3.4)” is realized as Web Modeler project. This is WWW service, which allows testing components CEXS, STEL, DAES and UEST independently. It can keep a communication configuration for multiple nodes and in easy way switch between nodes.
- [REPEAT18] **Component:** “HSMClient.cgi (3.4)” is realized as Web Modeler project. This component is very similar to CEXSClient.cgi but it connects with UEST. It is used by DCWP as a hyphen between WWW service and UEST. For a technical details see "DCWP Design

Description" document.

- [REPEAT19]**Component:** “STELClient.cgi (3.4)” is realized as Web Modeler project. This is very similar to CEXSClient.cgi but it connects with STEL. It is used by DCWP as a hyphen between WWW service and STEL. For a technical details see "DCWP Design Description" document.

### 3.4. PROVIDED CONFIGURATION FILES

This section shows short description of all provided by Task 3.4 elements, which are configuration files required by other components. The presented items below are ordered in the alphabetical order.

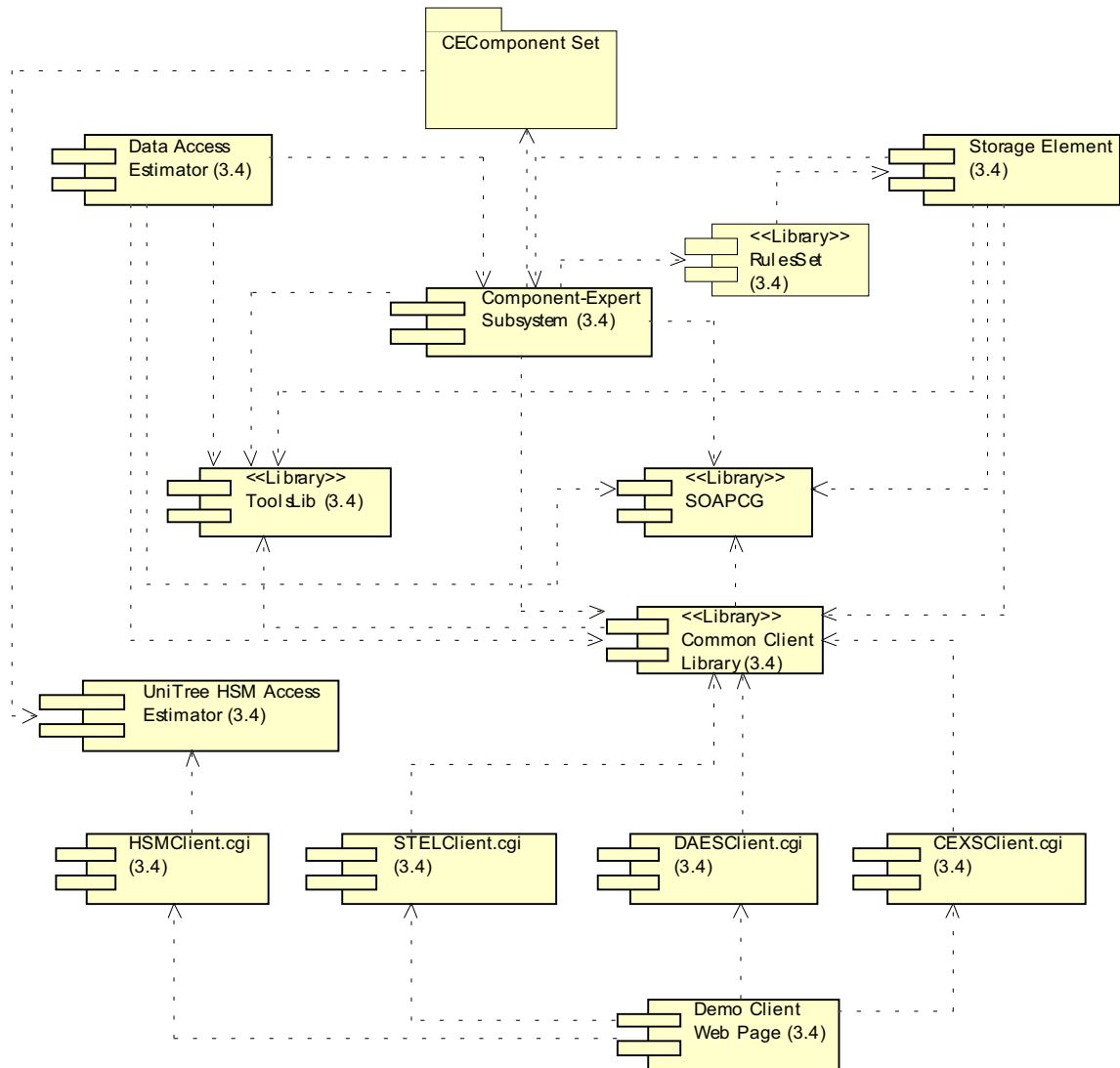
- [REPEAT20]**Component:** “CEXS Configuration File” is realized as ANSI C++ project. 'CEXS Configuration' is a file keeping all important for running of CEXS parameters which should be changed by the system administrator. It keeps: logging level; path to important files; number of TCP ports etc. This file is loaded by CEXS during its starting process. It is modified only manually. For more technical details see document: "CEXS Configuration File -- Short Description".
- [REPEAT21]**Component:** “Components Container Configuration File” is realized as ANSI C++ project. 'Components Container Configuration' is a file keeping list of currently registered and managed by CEXS CEComponents. In fact is keeps list of URLs pointing at particular CEComponents. During CEXS activity the state of this file can be changed. This file is loaded ones during CEXS starting. CEXS modifies this file automatically (after calling register/unregisterComponent operation). However this file can be modified manually by system administrator but only when CEXS is not working. For more technical detail see: "Components Container Configuration File -- Short Description"
- [REPEAT22]**Component:** “DAES Configuration File” is realized as ANSI C++ project. This configuration file is very similar to 'CEXS Configuration File' but it is intended for DAES

component. It specifies parameters all parameters important for running of DAES. For more technical details see document: "DAES Configuration File -- Short Description".

- [REPEAT23] **Component:** “STEL Configuration File” is realized as ANSI C++ project. This configuration file is very similar to 'CEXS Configuration File' but it is intended for STEL component. It specifies parameters all parameters important for running of STEL. For more technical details see document: "STEL Configuration File -- Short Description".
  
- [REPEAT24] **Component:** “Storage Element Configuration” is realized as ANSI C++ project. This configuration file keeps all information about the current storage configuration. The main goal of this file is to provide all information about a storage device (storage type, vendor connection to storage type, physical system path etc.) which keeps particular data object. Obviously, it is not necessary to put each data objects independently, but a usage of regular expressions which describe a group of objects is allowed. This file is used by STEL and is loaded during its starting process. It is intended to manual modifications. For more technical details see document: "Storage Configuration File -- Short Description".

### 3.5. RELATIONSHIP DIAGRAMS BETWEEN PROVIDED COMPONENTS

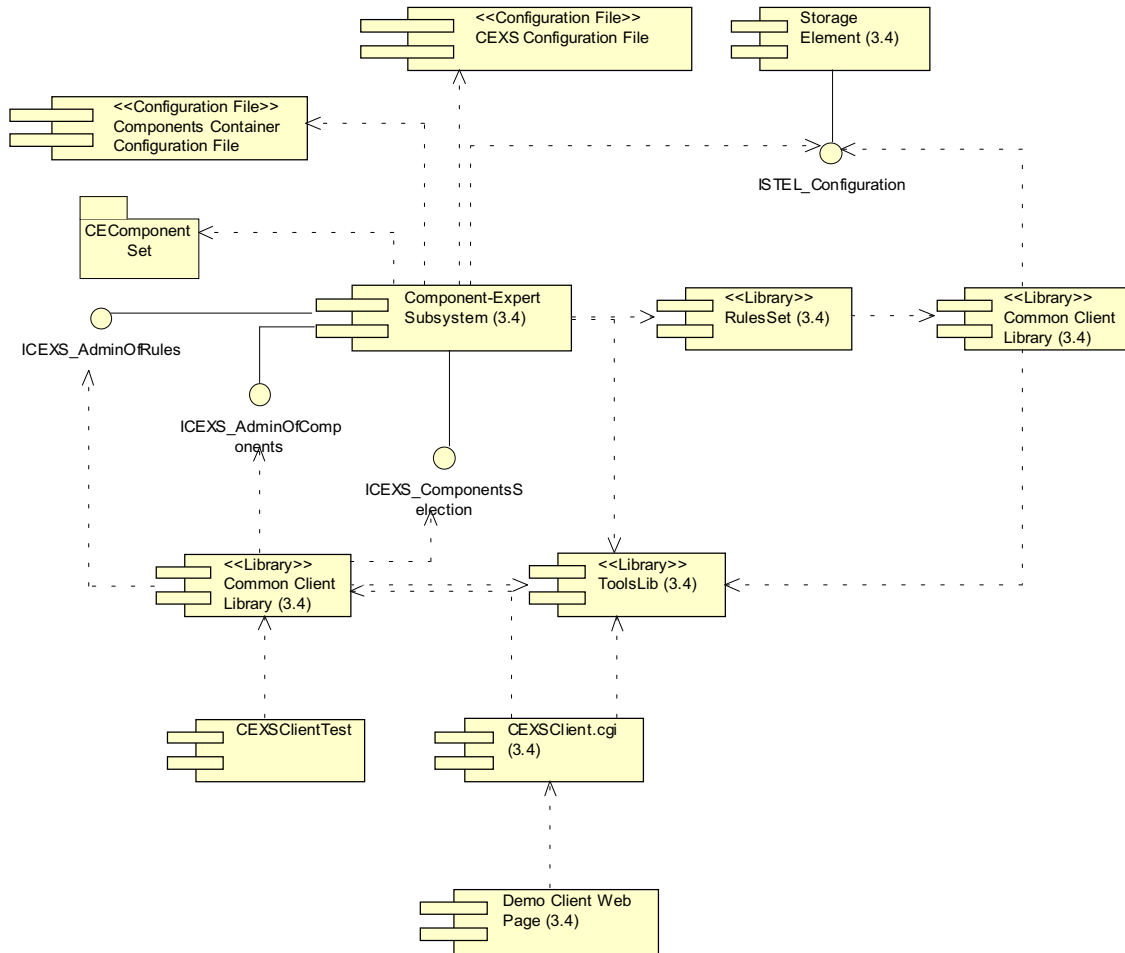
[REPEAT25]



**Fig. 1 D3.3 Components Dependencies**

This diagram depicts dependencies between the all components, which are provided as prototype for D3.3.

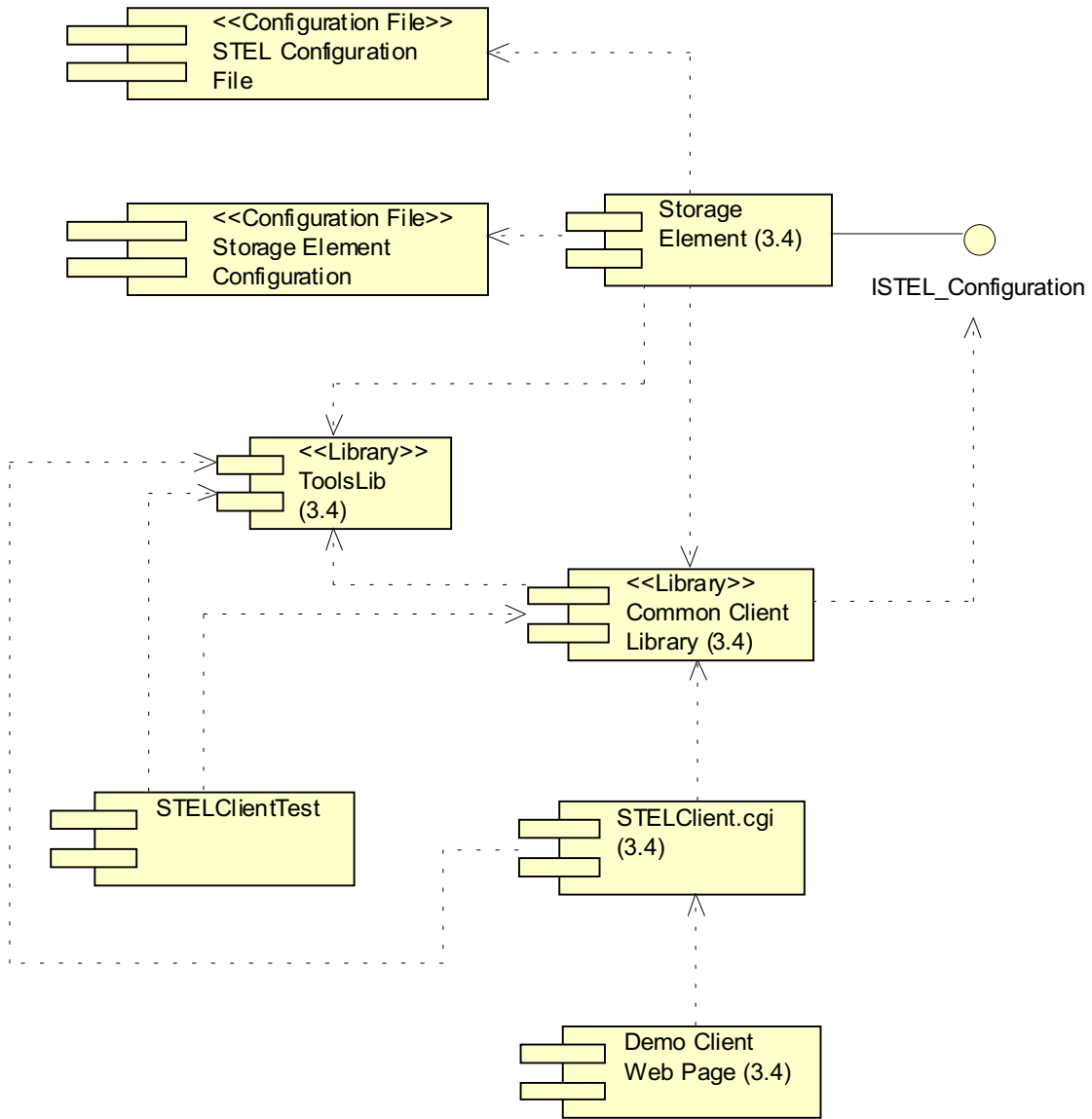
[REPEAT26]



**Fig. 2 D3.3 Environment of CEXS**

This diagram presents intercomponent communication model which is drawn from CEXS point of view. Additionally, it presents dependencies between CEXS and required configuration files.

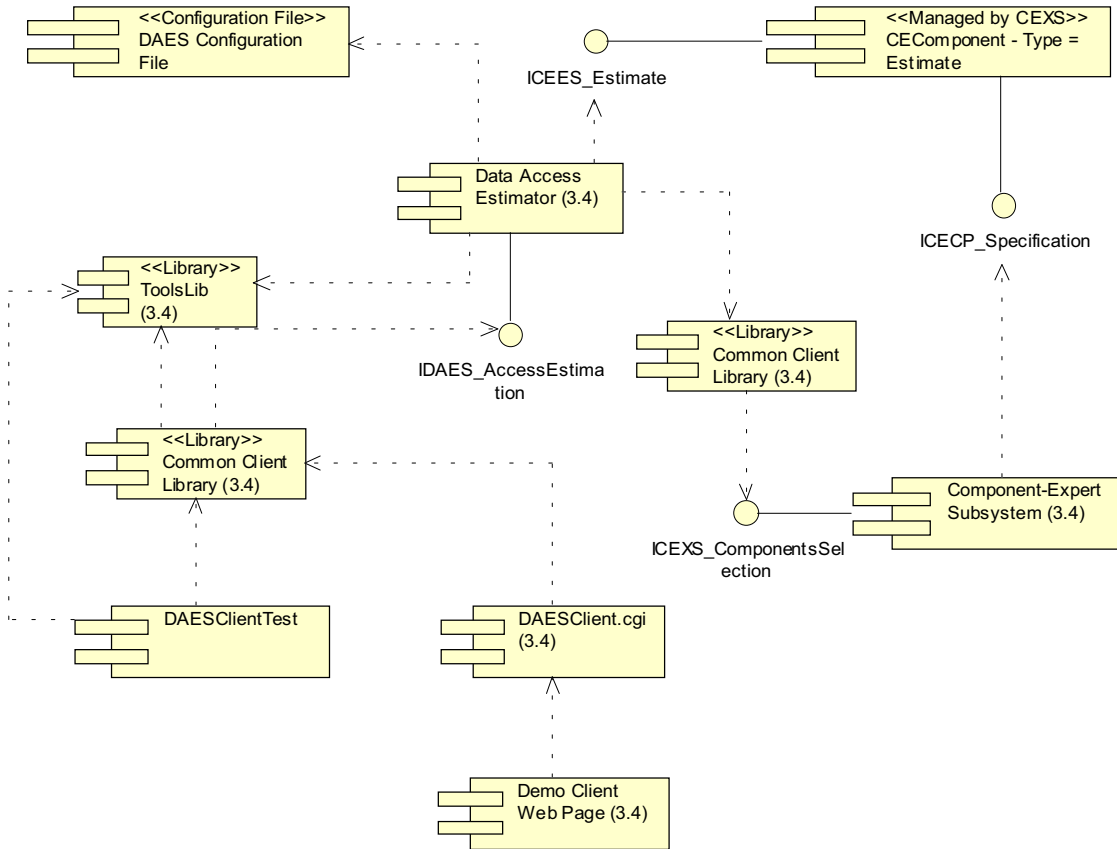
**[REPEAT27]**



**Fig. 3 D3.3 Environment of STEL**

This diagram presents intercomponent communication model which is drawn from STEL point of view. Additionally, it presents dependencies between STEL and required configuration files.

[REPEAT28]



**Fig. 4 D3.3 Environment of DAES**

This diagram presents intercomponent communication model which is drawn from DAES point of view. Additionally, it presents dependencies between DAES and required configuration files.



- Component Expert Subsystem,

which are presented in Fig. 5 as the processes below ‘Storage Nodes’. Additionally, Task 3.4 provides related to Data Access Estimator set of CEComponents for estimations, and especially a CEComponent for estimation of data access factors for UniTree storages, which exploits another provided as D3.3 component ‘UniTree HSM Access Estimator’ which is marked in Fig. 5 as the process below the ‘HSM Server UniTree’ node.

Task 3.4’s nodes are marked in Fig. 5 as dark boxes (green). Mainly there are ‘Storage node’s but there are also other not mentioned yet: ‘Web Server’ and ‘Tester Desktop’. These nodes were introduced only for testing purposes. Task 3.4’s code cannot be at the moment fully integrated with the rest of grid due to reasons, which are explained in section 3.7.

### **3.7. DIFFERENCES BETWEEN IMPLEMENTED AND PRESENTED IN SDD (D3.2 DOCUMENT) MODEL**

The presented component model at the begging of section 3 is almost identical as one proposed in the SDD document.

Basing on your Design Document, please state which modules have been implemented as part of the prototype. Provide class diagrams where appropriate. Compare the implementation model against the design model. Point out any discrepancies and explain their rationality. However, there are little modification – extends.

First of all, SOAP communication functions have been moved from the TOLB library to two separate components Common Client Library and SOAPCG. The main reason of such modification was dictated by problems with generated code by gSOAP tool (gSAOP is used for generation of SOAP communication layer).

Additionally, communication classes covering low level SOAP functions generated by gSOAP and gathered in SOAPCG, brought together in Common Client Library (CLLB). In the previous mode; these classes were in the TOLB library.

During implementation arose extra requirements for available attributes of configuration of the storage node. Currently following attributes are available:

- Type of a storage device
- Vendor of a storage device:

- Connection type to a storage device
- Physical path pointing at a storage device.

In consequence new operations have been added to ISTELE\_Configuration interface of STELE .

The SDD document states internal class structure of each component. Currently implemented modules are very similar to the presented in the previous document model. However, during implementation some of the assumed structure of STELE and DAES had to be extended and additional features have been added. The technical detail and a detail description of internal structure of each component is too large and goes beyond this document but it is provided in external documents [12-20].

In summary none of proposed in SDD features have been removed but current changes have introduced new features of developed functionality.

## 4. PROTOTYPE FUNCTIONALITY

Please describe the functionality of your prototype in terms of sequence diagrams and/or use cases. Explain which aspects of overall functionality have been implemented. Compare the results with the provisions of the technical annex as applied to your software module(s).

The main goals of Task 3.4 stated in Annex 1 (p.8, 16, 65, 69-70) [3] are following:

- a) build supporting subsystem that will estimate time needed to make file available,
- b) extend the data management system under development of DataGrid,
- c) build middleware for faster access to large tape resident files, using sub-filing,
- d) build heuristic expert system that will act as advisor for selection of migration/replication policies in user defined circumstances.

However, as the result of the state of the art and requirements analysis, point d) has been removed due to duplication of work done by DataGrid project. This modification was accepted by reviewers during verification of the documents SRS [1] and SDD [2]. Instead of Task 3.4 starts extended cooperation with DataGrid project concerning migration and replication policies. Additionally, Task 3.4 has enhanced functionality of point a), and introduces subsystem for managing of internal activity of storage nodes and organized according Component-Expert Architecture [1-5]. In consequence, it will provide scalable, flexible and easy configurable environment for very heterogeneous part of grid. (The details are stated in SRS and SDD.)

The main activity during implementation of the first prototype has been aimed at the realization of the extended point a), thus to build scalable and flexible internal architecture of storage nodes (usually for storage centers, see Fig. 5, dark (yellow) plains) and at a preparations of estimation modules for different storage types, which are compatible with CEA. As the result of the work the following component are provided:

- Storage Element (STEL) – more detailed description in subsection 4.1.
- Component-Expert Subsystem (CEXS) – more detailed description in subsection 4.2.
- Data Access Estimator (DAES) – more detailed description in subsection 4.3.
- Sample set of CEComponents for estimation, compatible with CEXS and working with DAES – more details in subsection 4.4.
- Estimation module working at UniTree server: UEST.

- Communication and tool libraries for above components: Common Client Library (CLLB); Tools Library (TOLB) and SOAPCG Library.

Any code to realizing Point b) couldn't be done due to ongoing drastic changes in EDG Replica Manager. However Task 3.4 has done many things, which are necessary for future integration.

Point c) requires working CEA and due to large scale of complexity components realizing this architecture (CEA) and very short implementation time for the first prototype (only 3 months) only base module and estimation part of CEA have been accomplished. This middleware is still under construction. And will be available soon.

Additionally, Task 3.4 provided additional Web Page Service to easy test the provided software. More detailed description of functionality is stated in section 4.5.

#### 4.1. FEATURES OF PROVIDED STEL COMPONENT

STEL is mainly intended for managing of current configuration of the storage node. It interprets storage configuration file [27] and answers on the following questions using SOAP protocol:

- What is a type of the storage, which keeps the data pointed by a physical data name passed as a parameter,
- What is a storage vendor, which provides hardware storing the data pointed by a physical data name passed as a parameter,
- What is a type of connection between 'Storage Node' and the device which keeps the data pointed by a physical data name passed as a parameter,
- What is the internal system path for passed as a parameter a physical data name (e.g. what is the internal system path for pfn://UniTree/file.dat, and the answer could be ftp://hsm.host/dir/file.dat or /var/crossgrid/dir/file.dat etc.),

Additionally, running parameters of this component are fully configurable via configuration file (STELConfigurationFile [26]). The administrator can configure following features:

- The detail level of internal logging activity form very detail (LOG\_DEBUG) to almost none (LOG\_NONE)
- The path to the StorageConfigurationFile
- The path to the log file

- TCP Port used for listening of SOAP calls
- Threading: multi or single thread
- The path to a file where PID of the main process is stored (it is useful for external scripts)

Additional information about internal architecture of this component as: class diagrams, sequence diagrams and other are described in detail in document STEL Design Description [14].

## 4.2. FEATURES OF PROVIDED CEXS COMPONENT

CEXS is intended for managing of a set of CEComponents and the selection of the best component for a particular call-environment (conext) using a rule-based deduction process [1-4, 21]. The functionality of the provided prototype version of this component realizes all assumed in SDD features and in the summary it looks as following:

- Returning a handle to a registered CEComponent, which has required type, is registered in CEXS (actually in the Component Set) and is the best for a passed call-environment.
  - o During deduction currently registered components are used
  - o To take a decision a set of rules is used, which are loaded as external dynamic linking library and could be reloaded in the fly
  - o The passed in the request call-environment could be extended by rules (if it is necessary).
  - o The provided prototype set of rules can extend call-environments by information returned by STEL. The extended information is stored as additional attributes in the processed call-environment.
  - o As the result the last state of the processed call-environment is also returned. This avoids obtaining the same information from STEL again by a client of CEXS, which request is processed.
- Registering new CEComponents in the fly. Unregistering CEComponent in the fly. Saving registered CEComponents to the Components Container Configuration File [25]
- Returning handles to all registered CEComponents with a particular type. Reloading the current rules library in the fly. (New rules are used for next deductions)
- The set of rules is provided as external and easy to modification dynamic linking library.

- All activity is logged into files: one log file for general activity and the second for expert system activity.

Additionally, running parameters of this component are fully configurable via configuration file (CEXSConfigurationFile [24]). The operator can configure following features:

- The detail level of internal logging activity form very detail (LOG\_DEBUG) to almost none (LOG\_NONE) for general logging and for logging of expert system activity separately.
- The path to the log files
- The path to the dynamic linking library containing rules
- The path to the Components Container Configuration file
- TCP Port used for listening for SOAP calls
- TCP Port used for SOAP communication with STEL
- Threading: multi or single thread
- The path to a file where PID of the main process is stored (it is useful for external scripts)

Additional information about internal architecture of this component as: class diagrams, sequence diagrams and other are described in detail in document “CEXS Design Description” [12].

### 4.3. FEATURES OF PROVIDED DAES COMPONENT

DAES is intended for handling estimations requests. It is CEA oriented, thus estimations are moved to external CEComponents with type ‘Estimate’ (this value is fixed). It answers the following questions:

- What is the internal latency of the access to a data object identified by a passed physical file name
- What is the internal bandwidth of the access to a data object identified by a passed physical file name
- What are both factors (internal bandwidth and latency) of the access to a data object identified by a passed physical file name

To realize above features DAES does the following steps:

- Get the passed physical file name and use it for the preparation of the call-environment. The obtained name is encoded as the value of ‘DataID’ attribute (the name of this attribute is fixed and documented in the specification of Attributes [21]) in the prepared call-environment

- DAES asks CEXS what is the best ‘Estimate’ type CEComponent for the just prepared call-environment
- CEXS makes the deduction using currently registered rules. This deduction works selects amongst all ‘Estimate’ type CEComponents and to improve the result, rules try to extend the passed call-environment by asking STEL for the following values:
  - What is the type of the storage, which keeps the data pointed by ‘DataID’ attribute in the processed call-environment? The returned value is stored in ‘StorageType’ attribute in the processed call-environment.
  - What is the storage vendor of hardware currently storing the data pointed by ‘DataID’ attribute in the processed call-environment? The returned value is stored in ‘StorageVendor’ attribute in the processed call-environment.
  - What is the type of connection between ‘Storage Node’ and the device which keeps the data pointed by ‘DataID’ attribute in the processed call-environment? The returned value is stored in ‘ConnectionType’ attribute in the processed call-environment.
  - What is the internal system path for the data pointed by ‘DataID’ attribute in the processed call-environment? The returned value is stored in ‘DataPhysicalID’ attribute in the processed call-environment.
- As the result of the deduction are returned to DAES the handle to the best CEComponent and the last state of the processed call-environment.
- DAES using the just obtained handle connects to a particular ‘Estimate’ type CEComponent [23]. It calls appropriate operation of ‘ICEES\_Estimate’ interface and as the argument of this operation it passes the just obtained call-environment to CEComponent.
- The CEComponent processes obtained call-environment, and especially takes out the ‘DataPhysicalID’ attribute, which contains a system path to the estimated data object. The internal estimation process strongly depends on a particular estimating CEComponents (see section 4.4). The computed results are returned to DAES.
- DAES preprocess the obtained values and returns the answer the operation caller.

Additionally, running parameters of DAES are fully configurable via configuration file (DAESConfigurationFile [28]). The operator can configure following features:

- The detail level of internal logging activity form very detail (LOG\_DEBUG) to almost none (LOG\_NONE) for general logging
- The path to a log file
- TCP Port used for listening for SOAP calls
- The name or address interface for listening (it could be automatically detected also)
- TCP Port used by CEXS
- Threading: multi or single thread
- The path to a file where PID of the main process is stored (it is useful for external scripts)

Additional information about internal architecture of DAES as: class diagrams, sequence diagrams and other are detailed described in document DAES Design Description [13].

#### **4.4. FEATURES OF PROVIDED ‘ESTIMATE’ TYPE CECOMPONENTS**

Provided sample ‘Estimate’ type CECOMPONENTS are compatible with CEXS, and are capable to return own specialization, type and estimated access factors for a particular data. The data is identified by attributes of the passed call-environment which is a required for the estimation.

As the prototype couple estimation components are provided. There is one for estimation data access factors of files managed by UniTree HSM. That CECOMPONENT in reality connects to the UEST component working directly at a particular UniTree node. This is implied from the fact that to estimate access factor for UniTree located files the direct access the UniTree node is required. Communication, between these components is done via SOAP.

#### **4.5. FEATURES OF DEMO CLIENT WEB PAGE**

This component is intended only for providing an easy way to test the delivered components. It allows the following operations:

- Registering Storage Nodes and UniTree HSM Servers. During this registration process an address and appropriate ports are taken and stored.
- Modification currently stored connection parameters of already registered nodes.
- Removing unnecessary nodes.
- Testing each operation provided by the components: STEL, CEXS, DAES and UniTree HSM

Access Estimator, which are running at a selected by the tester server.

- Sample values for each form are shown to make testing easier.

## 5. USER MANUAL

Because the provided software is divided into groups depending on a type of node then all information concerning one particular type of node are collected in one subsection below.

This section is intended to explain how to install and use the prototype. Please provide specific and understandable information, using checklists and screenshots where appropriate.

### 5.1. DEPENDENCIES

#### 5.1.1. Storage Node

At the current state Task 3.4's components do not require any specific hardware and software modules more than are provided with Linux Red Hat 6.2. These components are daemons and they listen for connections on some specific TCP ports (the numbers of these ports are freely configurable in appropriate configuration files), thus the chosen ports have to be available, and obtainable by clients. Sometimes, it is required a little modification of a configuration of a firewall.

#### 5.1.2. HSM UniTree Server

For the HSM server the Legato DiskXtender UL ver.2.4 (former UniTree) is required. Additionally the Jetty web server (<http://jetty.mortbay.org>) and Apache-SOAP (<http://xml.apache.org/soap/>) have to be installed and running in order to have SOAP access to the access time estimation component.

#### 5.1.3. Web Server

To start own web service Apache HTTP server is required, with working PHP and execution of CGI have to be allowed. However, working installation is available at Task 3.4 Homepage (<http://gridportal.fzk.de/websites/crossgrid/cg-wp3-4/>), and it can be used freely for testing components working on any nodes, which are accessible over TCP/IP from gridportal.fzk.de host. If tested nodes are behind firewall then probably own installation will be required.

#### 5.1.4. Tester Desktop

The web pages generated by DCWP are delivered to the tester desktop. This service has been tested for compliance with Internet Explorer 5.5 and Netscape Navigator 4.79 for Linux. However, it should work properly for other browsers supporting HTML 3.2.

Please list the hardware and external software required to install and run the prototype (including

libraries and other CrossGrid modules). Be specific when it comes to version and revision numbers.

## 5.2. INSTALLATION

Please explain how to install the prototype (in steps, if possible).

### 5.2.1. Storage Node

To install software on a 'Storage Node' the source code of the Task 3.4's modules have to be downloaded. There are two possibilities: checkout whole Task 3.4' CVS repository, which is more preferable or download prepared installation pack, which is available at the page <http://gridportal.fzk.de/websites/crossgrid/cg-wp3-4/>.

To checkout whole CVS repository the following commands should be executed:

```
export CVSROOT=:ext:anoncvs@gridportal.fzk.de:/cvs/crossgrid
cvs -z3 co crossgrid/wp3/wp3_4-expert
```

After the code is taken the following activities have to be done:

1. Run `configureall` script. This script walks thru all sub directories and prepares appropriate make files. This step is usually run just ones.
2. Run `compileall`. This script walks thru all sub directories and compiles the code. This script should be run all times when something has been changed in the code.

As the result of these two steps above all links to binaries are gathered in the subdirectory 'bin' and sample configuration files are gathered in the subdirectory 'etc'.

### 5.2.2. HSM UniTree Server

To install the access time estimation component on the HSM server the CVS repository has to be checked-out (description how to do it is stated above). To do it the following commands should be executed:

After this the code has to be compiled:

1. `cd eta`
2. `gmake`

and the estimation service deployed:

1. `cd soapeta`
2. `java org.apache.soap.server.ServiceManagerClient http://localhost:8080/soap/servlet/rpcrouter  
deploy DeploymentDescriptor.xml`

### 5.2.3. Web Server

To install software on a 'Web Server Node' the prepared modules have to be taken. The prepared and compiled package of required modules is available at the page <http://gridportal.fzk.de/websites/crossgrid/cg-wp3-4/> (Demo Client Web Page link). This package contains PHP files and three subdirectories:

- `cgi-bin`, which contains compiled CGI binaries for Intel Linux
- `Images`, which contains all images required to present web pages properly
- `Tables`, which contains all configuration of this web service items.

After the modules are taken, the following steps should be done:

1. Copy all PHP files to the root directory of Apache Web server
2. Copy subdirectories `Images`, `Tables` and `cgi-bin` to the same directory like previously copied PHP
3. Change write access for the just copied file `Tables/Sites` and allow writing to this file for the Apache process
4. Copy subdirectory `cgi-bin` to the same directory like `Images` and `Tables` in the step just before.
5. Allow to execute CGIs located in the just created directory (created by copying process from in 4) `cgi-bin`.
6. Homepage for this service is `HomePage.php` file, so it should be used as the beginning of a usage of DCWP.

In the case when a recompilation of CGIs required, step 4 in the list of steps above should be replaced by the following steps:

1. Checkout the source from task 3.4's CVS repository. Using following commands  
`export CVSROOT=:ext:anoncvs@gridportal.fzk.de:/cvs/crossgrid  
cvs -z3 co crossgrid/wp3/wp3_4-expert`

2. Run `configureall` script. This script walks thru all sub directories and prepares appropriate make files. It is usually run just ones.
3. Run `compileall`. This script walks thru all sub directories and compiles code. This script should be run all times when something has been changed in the code.
4. Create `cgi-bin` directory in the same directory where PHP files have been copied.
5. Copy to this directory following binary files:
  - `./crossgrid/wp3/wp3_4-expert/src/DCWP/CGIS/CEXSClient/CEXSClientCgi` as `CEXSClient.cgi`
  - `./crossgrid/wp3/wp3_4-expert/src/DCWP/CGIS/STELClient/STELClientCgi` as `STELClient.cgi`
  - `./crossgrid/wp3/wp3_4-expert/src/DCWP/CGIS/DAESClient/DAESClientCgi` as `DAESClient.cgi`

#### 5.2.4. Tester Desktop

No specific preparations are required.

### 5.3. RUNNING

Please explain how to access the prototype and make use of its functionality as described in section 4.

#### 5.3.1. Storage Node

At storage nodes as it is presented in Fig. 5 have to be installed CEXS, STEL and DAES components. However, they these components require special configuration files. The example files are gathered in `etc` directory and executables are gathered in `bin` directory. This is, amongst other things, the result of installation activities presented in section 5.2.1.

The `bin` directory contains special script `aserver`, which is helpful during starting, restarting and stopping of these demons (components). To avoid problems, with different local directory structure `aserver` script has been written using so called ‘relative paths’. In consequence it can be run anywhere but it has to be fired directly from `bin` directory. In other words, the current working directory in starting moment has to point at `bin`

To start the daemons the operator runs `aserver start`.

To restart the daemons the operator runs `aserver restart`.

To stop the daemons the operator runs `aserver stop`.

### 5.3.2. HSM UniTree Server

At the HSM server the jetty web server should be up and running.

### 5.3.3. Web Server

At the web server node, which in fact is considered only for testing, DCWP component has to be working. If the previous described in section 5.2.3 steps have been done properly then the final step is to check if the Apache server is started if not then to start it.

### 5.3.4. Tester Desktop

Nothing special has to be done, just connection the appropriate Web Server node using a browser to should be. (One Web Server Node is working and it available for CrossGrid users at the address: <http://gridportal.fzk.de/websites/crossgrid/cg-wp3-4/> (Demo Client Web Page link).

## 5.4. INTERFACE DESCRIPTION

The presented in SDD document description of interfaces for provided as D3.3 has been a little extended during the implementation. The current state of these interfaces is presented below.

[REPEAT29]

[REPEAT30]

[REPEAT31]

### 5.4.1. Component: **Component-Expert Subsystem (3.4) – 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.

[REPEAT32]

*Operation Name:* `getAllComponents4Type`

*Return type:* `vector<CGString>`

*Documentation:* It returns the list of all UID of components, which have type as specified in parameter list for this operation.

[REPEAT33]

*Parameter Name:* compType

*Parameter Type:* CGString

*Documentation:* The type of searched components.

[REPEAT34]

*Operation Name:* registerNewComponent

*Return type:* intCG

*Documentation:* This operation registers a new CE component and makes its available for the selection process. Used by a developer or a system administrator. It returns an execution code.

[REPEAT35]

*Parameter Name:* compURL

*Parameter Type:* CGString

*Documentation:* The URL to a registered component. This URL must be unique and it should allow running component in order to do an acquisition of a component-header.

[REPEAT36]

*Operation Name:* unregisterComponent

*Return type:* intCG

*Documentation:* This operation unregisters a CE component and makes its unavailable for the selection process. Used by a developer or a system administrator. It returns an execution code.

[REPEAT37]

*Parameter Name:* compURL

*Parameter Type:* CGString

*Documentation:* The URL to a registered component. This URL must be unique.

[REPEAT38]

**5.4.2. Component:    Component-Expert    Subsystem    (3.4)    –    Interface:**  
**ICEXS\_AdminOfRules**

[REPEAT39]

*Operation Name:* loadNewRuleLibrary

*Return type:* intCG

*Documentation:* This method loads new rule library, keeping the old one until it is in use.

[REPEAT40]

**5.4.3. Component:    Component-Expert    Subsystem    (3.4)    –    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.

[REPEAT41]

*Operation Name:* getBestComponent4CallEnv

*Return type:* GetBestResult\_struct

*Documentation:* This operation starts the best component selection process, which is made by Component-Expert Subsystem. It takes two parameters 'componentType' and 'callEnvironment'. The first one represents the type of the searched component. The second one represents an encoded into the structure form call-environment. The role of the call-environment is to describe the context for the selection process. As a result, this operation returns the structure (GetBestResult\_struct) which is consisted of two fields. The first 'bestURL' is URL pointed at the best component which is encoded into 'char\*'. The second field 'callEnv' is 'CallEnvironment\_struct' which represents the last state of the call-environment used for the selection. (It is important, because the call-environment can change during the deduction process). In the case when there isn't any component matching to the call-environment and type then GetBestResult\_struct in the field bestURL has the empty string. However, callEnv also has to represent the last state of the call-environment.

[REPEAT42]

*Parameter Name:* componentType

*Parameter Type:* CGString

*Documentation:* The type of component.

[REPEAT43]

*Parameter Name:* callEnvironment

*Parameter Type:* CallEnvironment\_struct\*

*Documentation:* The pointer to CallEnvironment\_structure keeping the encoded call-environment.

[REPEAT44]

[REPEAT45]

#### 5.4.4. Component: Data Access Estimator (3.4) – Interface: IDAES\_AccessEstimation

This interface is used to pass an inquiry about the cost of access to some file for a given Storage Element. This is sequence diagram of estimateDataAccessFactors operation realized by IDAES\_AccessEstimation interface.

[REPEAT46]

*Operation Name:* estimateDataAccessBandwidth

*Return type:* longCG

*Documentation:* It returns an estimated bandwidth to physical data access.

[REPEAT47]

*Parameter Name:* pfn

*Parameter Type:* CGString

*Documentation:* This is a physical file name.

[REPEAT48]

*Operation Name:* estimateDataAccessFactors

*Return type:* vector<longCG>

*Documentation:* It estimates both factors the latency and the bandwidth for the data access. It returns two factors the latency and the bandwidth.

[REPEAT49]

*Parameter Name:* pfn

*Parameter Type:* CGString

*Documentation:* This is a physical file name.

[REPEAT50]

*Operation Name:* estimateDataAccessLatency

*Return type:* longCG

*Documentation:* It estimates the latency for the data access.

[REPEAT51]

*Parameter Name:* pfn

*Parameter Type:* CGString

*Documentation:* This is a physical file name.

[REPEAT52]

[REPEAT53]

#### 5.4.5. Component: Storage Element (3.4) – Interface: ISTEEL\_Configuration

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

[REPEAT54]

*Operation Name:* convertIntoPhysicalName

*Return type:* CGString

*Documentation:* This operation converts URL pointing at the data stored on the storage element node. As the result it returns the physical path to the object pointed by 'dataURL' in the string form. If there is no such object then it returns the empty string.

[REPEAT55]

*Parameter Name:* dataURL

*Parameter Type:* CGString

*Documentation:* A particular URL pointing at the data object. This URL is encoded into the string form.

[REPEAT56]

*Operation Name:* getConnectionToStorageType

*Return type:* CGString

*Documentation:* This operation returns a way of connection to the device, which keeps data identified by dataID parameter.

[REPEAT57]

*Parameter Name:* dataID

*Parameter Type:* CGString

*Documentation:* It carries data id identifier.

[REPEAT58]

*Operation Name:* getStorageVendor

*Return type:* CGString

*Documentation:* This operation returns vendor parameters for the device, which keeps data identified by dataID parameter. The vendor parameter is understood generally. It could be just a vendor name, but it could contain additional parameters like a version or something else.

[REPEAT59]

*Parameter Name:* dataID

*Parameter Type:* CGString

*Documentation:* It carries data id identifier.

[REPEAT60]

*Operation Name:* getTypeOfStorage

*Return type:* CGString

*Documentation:* This operation returns a type of storage, which keeps the data identified by dataID parameter.

[REPEAT61]

*Parameter Name:* dataID

*Parameter Type:* CGString

*Documentation:* It carries a physical data name.

[REPEAT62]

*Operation Name:* isCEXSWorking

*Return type:* boolCG

*Documentation:* Returns true if on storage-element node the component-expert subsystem is running.

## 5.5. API

If your prototype provides an API, please describe its current state and list its elements.

Task 3.4 is not going to provide public available API because the communication with main components is organized using SOAP protocol and users can generate own client dispatchers for a required programming language. However, for a testing purposes internal library CLLB for communication with the main components (CEXS, STEL, DAES) can be used. This library works only for C++ programming language.

The sections below describe in brief all available methods provided by classes from CLLB, which could be used during tests of the components.

[REPEAT63]

[REPEAT64]

### 5.5.1. Class : CEXSClient

This class allows in the easy way to communicate with CEXS component. In reality, it conceals SOAP protocol low level operations and gives the programmer just clear methods for the instant usage. The most important fields are 'CEXSPort' and 'CEXSInterface' which store the data required to build the valid URL pointing CEXS component. Additionally, this class can log events but only when 'plog' filed is set to a valid pointer to an instance of Logger.

#### 5.5.1.1. Methods

[REPEAT65]

*Method name:* ~CEXSClient

*Documentation:* The default destructor.

[REPEAT66]

*Method name:* CEXSClient

*Documentation:* The default constructor. It doesn't require any parameters.

[REPEAT67]

*Method name:* CEXSClient

[REPEAT68]

*Parameter name:* right

*Parameter type:* const CEXSClient&

[REPEAT69]

*Method name:* get\_CEXSIPInterface

*Return class:* CGStringSync const&

*Documentation:* This method returns current value of CEXSIPInterface field.

[REPEAT70]

*Method name:* get\_CEXSPort

*Return class:* CGIntegerSync const&

*Documentation:* This method returns current value of CEXSPort field.

[REPEAT71]

*Method name:* get\_plog

*Return class:* Logger \*

*Documentation:* It returns pointer to currently used Logger.

[REPEAT72]

*Method name:* getAllComponents4Type

*Return class:* vector<CGString>

*Documentation:* This method calls getAllComponents4Type operation of CEXS component, and passes a required type of searched components. As the result the list of handles to the appropriate CEComponents is returned.

[REPEAT73]

*Parameter name:* compType

*Parameter type:* CGString

*Documentation:* The type of searched components.

[REPEAT74]

*Method name:* loadNewRuleLibrary

*Return class:* intCG

*Documentation:* This method calls loadNewRuleLibrary of CEXS component pointed by the fields: CEXSIPInterface and CEXSPort.

[REPEAT75]

*Method name:* registerNewComponent

*Return class:* intCG

*Documentation:* This method calls registerNewComponent operation of CEXS component, and passes URL to the registered CEComponent.

[REPEAT76]

*Parameter name:* compURL

*Parameter type:* CGString

*Documentation:* The URL to a registered component.

[REPEAT77]

*Method name:* set\_CEXSIPInterface

*Return class:* void

*Documentation:* This method sets the value of CEXSIPInterface filed on the value passed as left argument This filed determines address of CEXS component.

[REPEAT78]

*Parameter name:* left

*Parameter type:* CGStringSync

[REPEAT79]

*Method name:* set\_CEXSPort

*Return class:* void

*Documentation:* This method sets the value of CEXSPort filed on the value passed as left argument. This filed determines TCP port on which CEXS is listening for connection.

[REPEAT80]

*Parameter name:* left

*Parameter type:* CGIntegerSync

[REPEAT81]

*Method name:* set\_plog

*Return class:* void

*Documentation:* This method sets the pointer to Logger, which should be used.

[REPEAT82]

*Parameter name:* left

*Parameter type:* Logger \*

[REPEAT83]

*Method name:* unregisterComponent

*Return class:* intCG

*Documentation:* This method calls unregisterComponent operation of CEXS component, and passes URL to the unregistered CComponent. It returns an execution code.

[REPEAT84]

*Parameter name:* compURL

*Parameter type:* CGString

*Documentation:* The URL to a registered component. This URL must be unique.

[REPEAT85]

### 5.5.2. Class : DAESClient

This is the client class for the DAES components. This class covers the SOAP communication methods and gives the access to the operations served by DAES component via simple call of an appropriate method.

#### 5.5.2.1. Methods

[REPEAT86]

*Method name:* ~DAESClient

*Documentation:* Default destructor.

[REPEAT87]

*Method name:* estimateDataAccessBandwidth

*Return class:* CGLong

*Documentation:* This method calls estimateDataAccessBandwidth operation of DAES component, and passes a physical file name to the data for estimation is expected. As the result bandwidth in kilobits per second is returned.

[REPEAT88]

*Parameter name:* pfn

*Parameter type:* CGString

*Documentation:* This is a physical file name.

[REPEAT89]

*Method name:* estimateDataAccessFactors

*Return class:* vector<CGLong>

*Documentation:* This method calls estimateDataAccessFactors operation of DAES component, and passes a physical file name to the data for estimation is expected. As the result is returned two elements vector consisted of latency in milliseconds and bandwidth in kilobits per second.

[REPEAT90]

*Parameter name:* pfn

*Parameter type:* CGString

*Documentation:* This is a physical file name.

[REPEAT91]

*Method name:* estimateDataAccessLatency

*Return class:* CGLong

*Documentation:* This method calls estimateDataAccessLatency operation of DAES component, and passes a physical file name to the data for estimation is expected. As the result latency in milliseconds is returned.

[REPEAT92]

*Parameter name:* pfn

*Parameter type:* CGString

*Documentation:* This is a physical file name.

[REPEAT93]

*Method name:* get\_DAESIPInterface

*Return class:* CGStringSync const&

*Documentation:* This methods returns a current value of DAESIPInterface field.

[REPEAT94]

*Method name:* get\_DAESPort

*Return class:* CGIntegerSync const&

*Documentation:* This methods returns a current value of DAESPort field.

[REPEAT95]

*Method name:* set\_DAESIPInterface

*Return class:* void

*Documentation:* This method sets the value of DAESIPInterface filed on the value passed as left argument This filed determines address of DAES component.

[REPEAT96]

*Parameter name:* left

*Parameter type:* CGStringSync

[REPEAT97]

*Method name:* set\_DAESPort

*Return class:* void

*Documentation:* This method sets the value of DAESPort filed on the value passed as left argument. This filed determines TCP port on which DAES is listening for connection.

[REPEAT98]

*Parameter name:* left

*Parameter type:* CGIntegerSync

[REPEAT99]

### **5.5.3. Class : STELClient**

This class allows in the easy way to communicate with the STEL component. In reality, it conceals SOAP protocol low level operations and gives the programmer just clear methods for the instant usage. The most important fields are 'STELPort' and 'STELIPInterface' which store the data required to build the valid URL pointing STEL component. Additionally, this class can log events but only when 'plog' filed is set to a valid pointer to an instance of Logger.

#### **5.5.3.1. Methods**

[REPEAT100]

*Method name:* ~STELClient

*Documentation:* Default destructor.

[REPEAT101]

*Method name:* convertIntoPhysicalName

*Return class:* CGString

*Documentation:* This method calls `convertIntoPhysicalName` operation of STEL component, and passes a physical data name to the data for which the conversion is expected. As the result the system physical path the data pointed by pfn is returned of storage type is returned.

[REPEAT102]

*Parameter name:* pfn

*Parameter type:* CGString

[REPEAT103]

*Method name:* `get_plog`

*Return class:* `Logger * const&`

*Documentation:* It returns a pointer to the currently used Logger.

[REPEAT104]

*Method name:* `get_STELIPInterface`

*Return class:* `CGStringSync const&`

*Documentation:* This methods returns a current value of STELIPInterface field.

[REPEAT105]

*Method name:* `get_STELPort`

*Return class:* `CGIntegerSync const&`

*Documentation:* This methods returns a current value of STELPort field.

[REPEAT106]

*Method name:* `getConnectionToStorageType`

*Return class:* CGString

*Documentation:* This method calls `getConnectionToStorageType` operation of STEL component, and passes a physical data name of the data object for which a type of connection between storage node and storage device is expected. As the result the name of connection type is returned.

[REPEAT107]

*Parameter name:* DataID

*Parameter type:* CGString

[REPEAT108]

*Method name:* getStorageVendor

*Return class:* CGString

*Documentation:* This method calls getStorageVendor operation of STEL component, and passes a physical data name of the data object for which the vendor's name of storage device is expected. As the result the name of vendor is returned.

[REPEAT109]

*Parameter name:* DataID

*Parameter type:* CGString

[REPEAT110]

*Method name:* getTypeOfStorage

*Return class:* CGString

*Documentation:* This method calls getTypeOfStorage operation of STEL component, and passes a physical data name of the data object for which a type of storage is expected. As the result the name of storage type is returned.

[REPEAT111]

*Parameter name:* dataID

*Parameter type:* CGString

*Documentation:* It carries data id identifier.

[REPEAT112]

*Method name:* isCEXSWorking

*Return class:* boolCG

*Documentation:* This method calls isCEXSWorking operation of STEL component. As the

result true or false is returned.

[REPEAT113]

*Method name:* set\_plog

*Return class:* void

*Documentation:* Sets pointer to Logger, which should be used.

[REPEAT114]

*Parameter name:* left

*Parameter type:* Logger \*

[REPEAT115]

*Method name:* set\_STELIPInterface

*Return class:* void

*Documentation:* This method sets the value of STELIPInterface filed on the value passed as left argument This filed determines address of STEL component.

[REPEAT116]

*Parameter name:* left

*Parameter type:* CGStringSync

[REPEAT117]

*Method name:* set\_STELPort

*Return class:* void

*Documentation:* This method sets the value of STELPort filed on the value passed as left argument. This filed determines TCP port on which STEL is listening for connection.

[REPEAT118]

*Parameter name:* left

*Parameter type:* CGIntegerSync

[REPEAT119]

*Method name:* STELClient

*Documentation:* Default constructor.

[REPEAT120]

*Method name:* STELClient

[REPEAT121]

*Parameter name:* right

*Parameter type:* const STELClient&

## 6. INTERNAL TESTS

The production of the code has been led according to guidelines specified in Clean Room Software Engineering [11]. However, due to financial limitation some simplification has been done, but the general idea: “to avoid inserting bugs instead of tracking them” has been preserved. To come closer this unequalled target, it had been introduced multilevel reviews of the code and intensive testing of each class separately.

The production process could be presented (in simplification) as the following steps:

1. Designer designs detailed model of each component which usually consists of classes, and produces technical production documentation. This model describes static and dynamic aspects of a particular component.
2. Project Manager assigns each class to a developer and determines time of the task accomplishing. This assignment determines that the chosen developer stays a main maintainer of the particular class and he is responsible for it. Project Manager additionally determines who will be responsible for the first review, productions review, intensive testing and the final review.
3. Developer builds the first implementation of the assigned class, and he puts the result into CVS repository. Additionally, he notifies the chosen reviewer for the first review.
4. Reviewer makes the first review and marks suspected parts of the code. To review a special marking language [29] is used. Finally the code is accepted or rejected by the reviewer, and the remarked code is put into CVS repository.
5. In the case of rejection of the code the developer (author) takes a chance to repair defects. All contentious issues are verified with the designer. After repairing the new version of the code is put into CVS repository.
6. The next reviewer reviews the code. Similarly to the first one, it remarks the code if any error is found. As the result he accepts or rejects the code. If the code is rejected then back to step 5.
7. In the same time the developer (author) builds own test of the class program. This is a simple test of correctness of the built class. If any error is found then back to step 5.
8. The code is passed to the selected Tester, who is responsible for intensive testing. During this test all bugs are collected and put into bug tracking system works together with portal. If any bugs are founded the tested code is marked as rejected by the tester. If code is rejected then

back to step 5.

9. The code is passed to the final review, which should accomplish only by acceptance of the code. Otherwise the Project Manager should be notified and he takes a decision what to do next with the particular class.

The presented steps above were practically and consequently used during production of the provided prototype. During production all built classes were tested separately. The kind of these tests and how deep they reached strongly depended on specific class. However, usually whitebox and blackbox techniques were harnessed. Additional information related to production and integration can be found in the document [6] “Test and Integration Process Description Deliverable D3.3”. Please explain what kinds of internal testing procedures have been applied to the prototype and in what manner was the software tested (comparison of actual output with required output, i.e. “black box” testing, confirmation of expected behavior of individual components, i.e. “white box” testing, other methods).

## 7. ISSUES

Please list all issues which affect the functioning of the prototype, including any known bugs.

The provided prototype cannot tackle properly with all exception and extraordinary situations. Better exception handling is planned in next versions.

The intensive tests showed that used 'string' class from STL [9] library has memory leaks, due to the extensive usage of this class in whole code and server characteristic of this software (no memory leaks are expected) this situation is unacceptable in production version of this software. Therefore, the own string class has been written but the old one hasn't been replaced yet. However, the issue of problems memory usage is seriously taken into account and will be systematically monitored in the future.

Another issue is the usage of unstable (DEVEL) version of the gSOAP tool. The stable version does not provide required functionality and the unstable version had to be used. This reflects in stability of the SOAPCG and CLLB libraries and in fact in stability of whole prototype. There are situations, in which provided by gSOAP code could be unstable. However, there is assured that all of these problems will be removed in the next stable version.