Web-SynDic

Web System for Demonstrating the Syntactic Algorithms for Solving Linear Equations in Nonnegative Integers (Nonnegative Linear Diophantine Equations)

Design Specification

Department of Computer Science, Petrozavodsk State University, Russia

15th November 2004

Contents

1 General Description

This section provides an overview of the entire design document.

2 System Architecture

In this section the architecture of the Web-SynDic system is designed. Several models are developed as design for the following key views: decomposition of Web-SynDic into subsystems, subsystem interfaces, and composite behavior of the subsystems.

Section 2.1 “Static Structure Model” describes the mentioned decomposition into subsystems; key static relations between the subsystems are presented. Section 2.2 “Subsystem Interface” states major dependences between the subsystems; high-level interface functions are listed for each subsystem. In section 4 “Behavioral Model” the key dynamic issues of the Web-system are designed—event-based communications between the subsystems with an appropriate time order.

The design of each subsystem must be strictly based on this architecture.

2.1 Static Structure Model

The Web-SynDic system is divided into three principal subsystems: “Server” (data processing and coordination), “Client Part” (a point for user access and data visualization), and “Data Store” (storage of user profiles and user activity information). Internal high-level structure of the subsystems and their communications are described in the next three subsections (2.1.1, 2.1.2, 2.1.3).

Each subsystem has its interface functions. They provide methods to access the data and to interact with the subsystem. In this section the high-level interface functions are presented only; for the detailed interface design see section 2.2.

2.1.1 Server

Subsystem “Server” can be divided into five subsystems, see Figure 1.

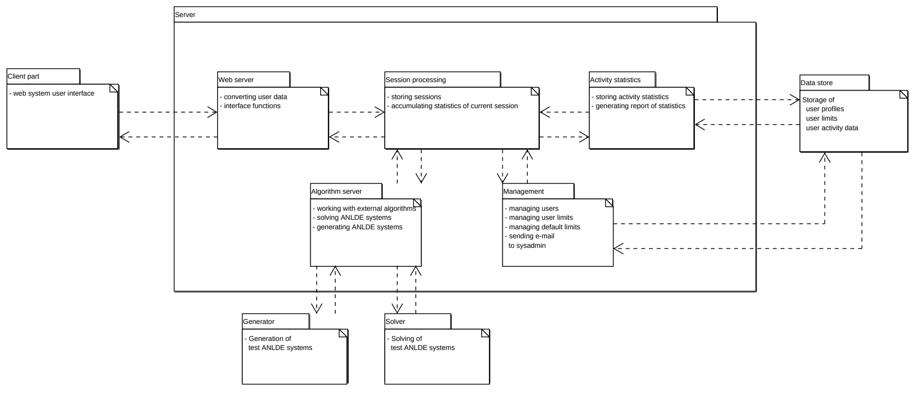


Figure 1: Detailed architecture of the server subsystem

Subsystem “Web server” is an interface service point between a user and Web-SynDic. The Web server receives requests and input data from a client part, converts them from input format to internal, redirects the request to an appropriate subsystem. Also it works in the reverse direction: the web server sends Web-SynDic replies and outcomes to a user, converting the data into an appropriate output format (visualization). The Web server generates all web forms to be used in a client part. The input formats are designed in section 3.3.

Subsystem “Session processing” manages all started sessions, coordinates and controls user data flows between the subsystems. In principal, this subsystem may be considered as a part of the Web server, but in the architecture it is presents as a separated subsystem due to its key role in the coordination.

Subsystem “Algorithm server” performs processing of problem domain objects according with the aimed demonstarting and testing issues. This server executes external algorithms whenever it is required by a user during her/his session, and passes the outcome to subsystem “Session processing”.

Subsystem “Management” manages user profiles and default limits. The necessary data are stored by subsystem “Data store”. Each registered user may manage her/his profile only. The system administrator may perform management for any user profile and for default limits.

Subsystem “Activity statistics” monitors and updates the activity of each user on a session basis stored by subsystem “Data store” and computes the activity statistics for the system administrator. Only the system administrator may perform this function. The session basis for data moninoring requires a synchronization with susbsystem “Session processing”.

2.1.2 Client Part

Subsystem “Client part” is an interface between a browser (external entity) and subsystem “Web server”. This is a starting point to access all functions that available to a user. This subsystem does not require explicit implementation: standard mechanisms of a browser and a web server are used to start the functions by a user.

Functions of the Client part are classified onto five groups according to use cases (see Figure 2 here for the classification and the Requirements Specification for the use case description).

Processing supports functions for sending ANLDE systems, ANLDE system reports and messages on the processing, according with Process an ANLDE system and Process a set of ANLDE systems use cases.

Notes supports function sendUserNotes() according with Send a note use case.

Management & statistics supports a set of functions according to Manage users, Get statistics, Manage user limits, and Manage default limits use cases.

Registration & login supports registerUser() and logInUser() functions according to Register a user and Login use cases.

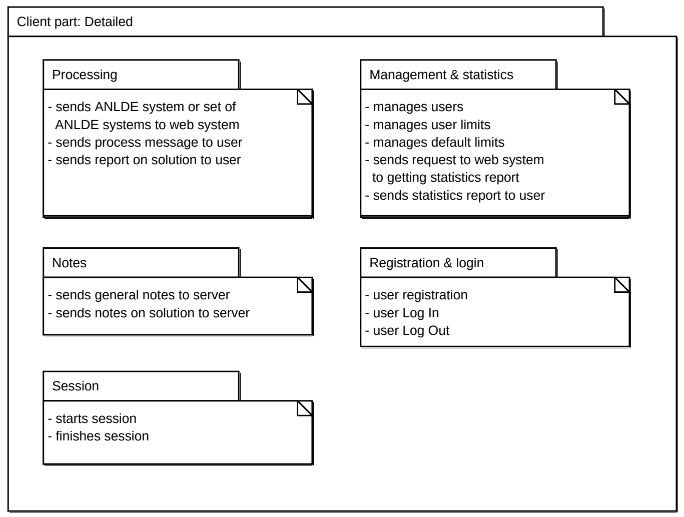

Client part: Detailed
Processing
- sends ANLDE system or set of ANLDE systems to web system
- sends process message to user
- sends report on solution to user
Management & statistics
- manages users
- manages user limits
- manages default limits
- sends request to web system to getting statistics report
- sends statistics report to user
Notes
- sends general notes to server
- sends notes on solution to server
Registration & login
- user registration
- user Log In
- user Log Out
Session
- starts session
- finishes session

Figure 2: Detailed architecture of Client Part entity

Session supports functions startSession() and finishSession() according to Work with Web-SynDic use case.

Each group must be assigned to a corresponding subsystem of the web server (JSP, sevlet or java class/package), see section 5 for details.

2.1.3 Data Store

Subsystem "Data store" is divided into three modules, see Figure 3.

User profile User profile is a set of user profiles; it contains data of registered users, e.g. nickname and password of each user.

A user profile of the Web-system contains information about a registered user. It is stored as a separate file with name equal to user nickname; this allows the Web-system to easily identify and manage the user profiles. The set of these files contains full information about registered users of the Web-system.

Draft format of a file with user profile:

1. version of user profile format
2. password

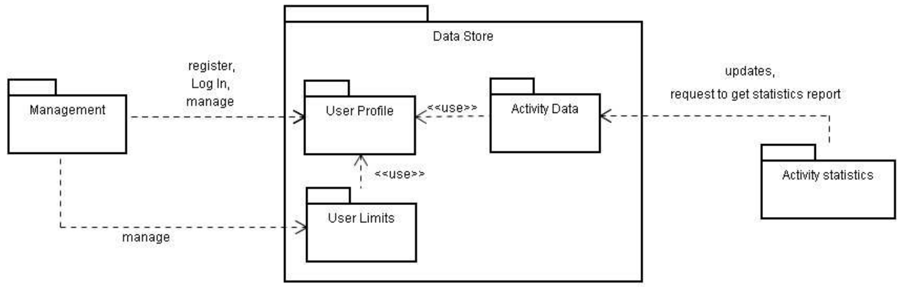


Figure 3: Detailed architecture of the Data Store

3. email
4. limits (see sect. 3.3.9)
5. short information about the user

Special case of a user with profile is *anonymous*. This profile is used for any unregistered user and may not be modified by a user.

Activity Data Activity data is a list of records. Each record corresponds to an atomic usage of the Web system like "log in", "ANLDE system solving", "sending a note", etc. A record includes time stamp, user ID (nickname for a registered user, or IP address for a regular user) and an activity parameter (type of user activity and particular attributes of this activity).

Activity data is a set of log files. Each log file is list of rows, each row is a record and contains detailed information about an atomic usage of the Web-system.

Draft format of row of a log file:

1. user nickname (for registered users only)
2. IP address
3. session start time stamp
4. session duration
5. activity metrics (see sect. 3.3.7)

For example of log file see sect. 5.6.5.

It is assumed that a log file contains activity information about one fixed time period, there is no intersection between periods of different files. The set of all log files of the Web-SynDic system contains full activity coverage in lifetime of the Web-system. Value of log file period is one month.

User limits User limits is a set of numeric values to define upper bounds on ANLDE systems representation and generating/solving process (see sect. 5.6.4 and 5.6.1). Any registered user has her/his own set of limits and they are stored in user profile. A non-registered user is anonymous user with the corresponding limits (the same for all such users).

When user registers in the Web system, corresponding user profile is created and added to the set of user profiles (at first time the user limits are as for anonymous user). The profile contains all data of the user including his/her limits. A registered user may change these limits.

Limits of a non-registered user may be changed during a current session, but in the data store they are not changed (user limits for anonumous user) and the changes are valid for this session only.

2.2 Subsystems Interface

The composition model for subsystems interface is shown in Figure 4.

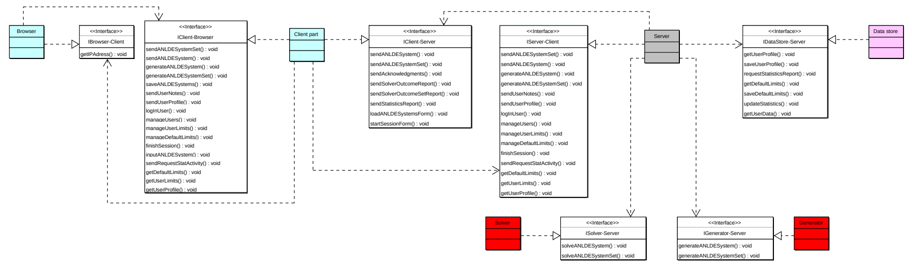

Browser
<<Interface>>
IBrowser-Client
getIPAdress() : void
<<Interface>>
IClient-Browser
sendANLDESystemSet() : void
sendANLDESystem() : void
generateANLDESystem() : void
generateANLDESystemSet() : void
saveANLDESystems() : void
sendUserNotes() : void
sendUserProfile() : void
logInUser() : void
manageUsers() : void
manageUserLimits() : void
manageDefaultLimits() : void
finishSession() : void
inputANLDESystem() : void
sendRequestStatActivity() : void
getDefaultLimits() : void
getUserLimits() : void
getUserProfile() : void
Client part
<<Interface>>
IClient-Server
sendANLDESystem() : void
sendANLDESystemSet() : void
sendAcknowledgments() : void
sendSolverOutcomeReport() : void
sendSolverOutcomeSetReport() : void
sendStatisticsReport() : void
loadANLDESystemsForm() : void
startSessionForm() : void
<<Interface>>
IServer-Client
sendANLDESystemSet() : void
sendANLDESystem() : void
generateANLDESystem() : void
generateANLDESystemSet() : void
sendUserNotes() : void
sendUserProfile() : void
logInUser() : void
manageUsers() : void
manageUserLimits() : void
manageDefaultLimits() : void
finishSession() : void
sendRequestStatActivity() : void
getDefaultLimits() : void
getUserLimits() : void
getUserProfile() : void
Server
<<Interface>>
IDataStore-Server
getUserProfile() : void
saveUserProfile() : void
requestStatisticsReport() : void
getDefaultLimits() : void
saveDefaultLimits() : void
updateStatistics() : void
getUserData() : void
Data store
Solver
<<Interface>>
ISolver-Server
solveANLDESystem() : void
solveANLDESystemSet() : void
<<Interface>>
IGenerator-Server
generateANLDESystem() : void
generateANLDESystemSet() : void
Generator

Figure 4: Detailed subsystems interface

2.2.1 Browser

The browser along with HTTP protocol has interface “IBrowser-Client”. This interface is used by client part to get user identification, such as IP address. The interface consists of standard HTTP functions and does not need extra installations. For the further details see sect. 4. Interface “IBrowser-Client” consists of the following functions:

getIPAdress() returns user IP address.

2.2.2 Client part

The client part has interfaces “IClient-Browser” and “IClient-Server”. “IClient-Browser” is used to extract data from web forms and send it to server. This is a virtual interface because all functions will be implemented in web forms or on server. Interface “IClient-Browser” consists of following functions:

sendANLDESystemSet() sends to the server a set of ANLDE systems,

sendANLDESystem() sends to the server an ANLDE system,

generateANLDESystem() sends a signal to generating ANLDE system,

generateANLDESystemSet() sends a signal to generating a set of ANLDE systems,

saveANLDESystems() saves a set of ANLDE systems in traditional mathematical style,

sendUserNotes() sends a user note to the server,

sendUserProfile() sends user’s registration data: nickname, password and order information, to server,

logInUser() sends to server user’s nickname and password to create new session,

manageUsers() sends changes in registered user profiles to server for modification,

manageUserLimits() sends new user limits to server for modification,

manageDefaultLimits() sends new default limits to server for modification,

registerUser() sends registration data to server for addition,

finishSession() sends a signal to finish session,

inputANLDESystem() calls a web form to input ANLDE system,

sendRequestStatActivity() sends a request to get statistics activity,

getDefaultLimits() gets default limits from the data store,

getUserLimits() gets user limits from the data store or from the session,

getUserProfile() gets user profile for management.

Interface “IClient-Server” is used to insert data into web forms. This is a virtual interface, because all functions are implemented on the server. Interface “IClient-Server” consists of following functions:

sendANLDESystem() sends ANLDE system to the browser,

sendANLDESystemSet() sends a set of ANLDE systems to the browser,

sendAcknowledgments() sends acknowledgment to browser,

sendProcessMessage() sends process message to the browser,

sendSolverOutcomeReport() sends report on solution to browser,

sendSolverOutcomeSetReport() sends report on solution to browser,

sendStatisticsReport() sends a report on statistics activity,

loadANLDESystemsForm() sends form to load a set of ANLDE systems,

startSessionForm() sends start session form.

2.2.3 Server

The server has "IServer-Client" interface. This interface is used to send data to the external algorithms or data store. Interface "IServer-Client" consists of following functions:

sendANLDESystemSet() sends a set of ANLDE systems to the solver,

sendANLDESystem() sends an ANLDE system to the solver,

generateANLDESystem() sends a signal to generating ANLDE system,

generateANLDESystemSet() sends a signal to generating a set of ANLDE systems,

sendUserNotes() sends user note to the data store,

sendUserProfile() saves user profile in the data store,

logInUser() checks nickname and password, and starts session,

manageUsers() sends changes in registered user profiles to the data store,

manageUserLimits() sends new user limits to the data store,

manageDefaultLimits() sends new default limits to the data store,

finishSession() finishes session and updates user activity,

sendRequestStatActivity() gets activity statistics from the data store and sends it to the client part,

getDefaultLimits() gets default limits from the data store,

getUserLimits() gets user limits from the data store or from the session,

getUserProfile() gets user profile for management.

2.2.4 Solver

The solver has "ISolver-Server" interface. This is an external interface and will not be implemented. Interface "ISolver-Server" consists of following functions:

solveANLDESystem() solves an ANLDE system,

solveANLDESystemSet() solves a set of ANLDE systems.

2.2.5 Generator

The generator has "IGenerator-Server" interface. This is an external interface and is implemented as a part of Web-SynDic. Interface "IGenerator-Server" consists of following functions:

generateANLDESystem() generates an ANLDE system,

generateANLDESystemSet() generates a set of ANLDE systems.

2.2.6 Data store

The data store has "IDataStore-Server" interface. This interface used for loading and extracting data from the data store. Interface "IDataStore-Server" consists of following functions:

getUserProfile() gets user profile,

saveUserProfile() saves user profile,

requestStatisticsReport() gets activity statistics,

getDefaultLimits() gets default limits,

saveDefaultLimits() saves default limits,

updateStatistics() updates user activity.

2.3 Server Subsystems Interface

The composition model for server subsystems interface is shown in Figure 5.

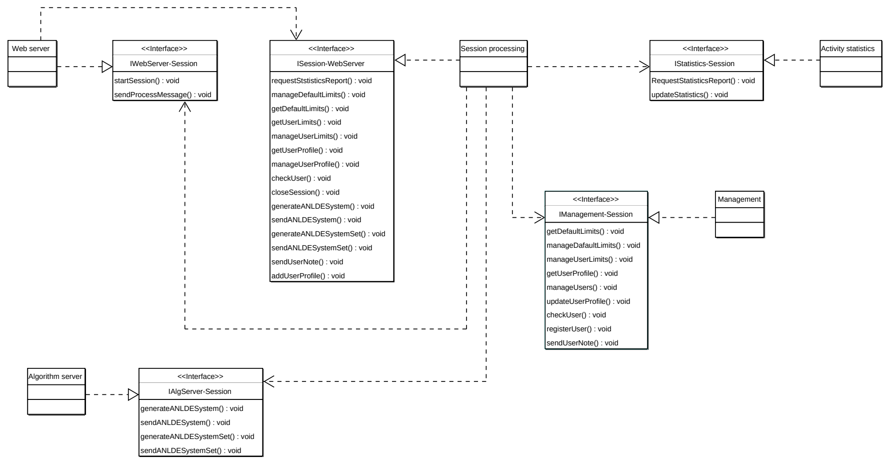


Figure 5: Detailed server subsystems interface

2.3.1 Web server

The web server has "IWebServer-Session" interface. This interface is used for sending to a user more than one message like process messages. Interface "IWebServer-Session" consists of functions:

startSession() sends initialized data and main page to user,

sendProcessMessage() sends process message to user.

2.3.2 Session processing

The session processing has one interface. This interface is used for getting and forwarding user data to others server subsystems. Interface "ISession-WebServer" consists of functions:

requestStatisticsReport() sends request for getting statistics report,

getDefaultLimits() gets default limits,

manageDefaultLimits() sends to change new default limits,

getUserLimits() gets user limits,

manageUserLimits() sends new user limits,

getUserProfile() gets user profile,

addUserProfile() adds new user profile,

manageUserProfile() sends to change new user data,

checkUser() sends request to check user,

closeSession() sends request to close session,

generateANLDESystem() sends parameters to generate ANLDE system,

generateANLDESystemSet() sends parameters to generate a set of ANLDE systems,

sendANLDESystem() sends ANLDE system to solve,

sendANLDESystemSet() sends a set of ANLDE systems to solve,

registerUser() sends user data for addition,

sendUserNote() sends user note.

2.3.3 Algorithm server

The algorithm server has one interface. This interface is used for solving and generating ANLDE systems. Interface "IAlgServer-Session" consists of functions:

generateANLDESystem() sends parameters to generate and starts generating process,

generateANLDESystemSet() sends parameters to generate a set of ANLDE systems and starts generating process,

sendANLDESystem() sends ANLDE system to solve and starts solving process,

sendANLDESystemSet() sends a set of ANLDE systems to solve and starts solving process,

2.3.4 Activity statistics

The activity statistics has one interface. This interface is used for update statistics. Interface "IStatistics-Session" consists of functions:

RequestStatisticsReport() sends request to get a statistics report,

updateStatistics() sends update of statistics.

2.3.5 Management

The management has one interface. This interface is used for management of users. Interface "IManagement-Session" consists of functions:

getDefaultLimits() gets default limits,

manageDefaultLimits() sends new default limits,

manageUserLimits() sends new user limits,

getUserData() gets user profile,

manageUsers() sends updates of user profile,

updateUserProfile() sends update of user profile,

checkUser() sends request to check user,

registerUser() sends user profile to registration,

sendUserNote() sends user note.

3 User Interface

This section describes the interface between a user and the Web-SynDic system. The key part of the user interface specification is section "Forms" (sect. 3.2). The forms are used to exchange data between a user and the Web system. Data formats, used in forms, are described in section "IO Data Formats" (sect. 3.3). Also this document includes section "Style Convention" (sect. 3.1) and section "Notes on Implementation" (sect. 3.4) to describe the corresponding issues. The detailed description of the user interface and the related use cases can be found in the next section "Behavioral model" (sect. 4).

3.1 Style Convention

3.1.1 Colors

Black text on a white background; blue hyperlinks; red warnings, important notes and required field marks and gray buttons will be used for Web-SynDic web pages. Main menu, user menu and log in/status page parts (see section 3.1.3 for Page Layout) have light gray background.

3.1.2 Forms

Key form design principles are ease to use, functionality and simplicity of implementation. Only standard controls (buttons, text field, text areas, check boxes and radio buttons) are used.

3.1.3 Page Layout

Each page consists of the following parts (see Fig. 6):

Header: web system's title;

Main menu: links to the main web system functions (Req. F1–F3) and related documentation;

User menu: links to user information and limits form, administration links (Req. F5–F6);

Log in form: displayed until user logs in. It also shows user's IP address (sect. 3.2.3 Log In Form, Req. F4);

Log in status: current user nickname and "Log out" link and user's IP address (displayed after user logs in);

Content: information and forms the user is currently working with (sect. 3.2). Also content contains brief textual description for each current form.

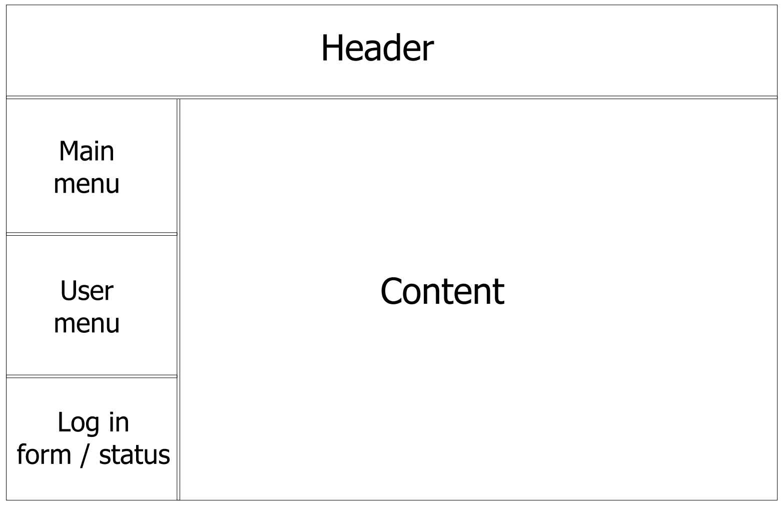


Figure 6: Page layout

3.2 Forms

3.2.1 Process an ANLDE System

Description: This form allows a user to send one ANLDE system, which may be generated or written manually, to server for solving. In this case a user can see her/his own limits on the same form and change them, whenever she/he wants (using "Change limits" button). Limits values are used as parameters for generating a new ANLDE system. See Fig. 7.

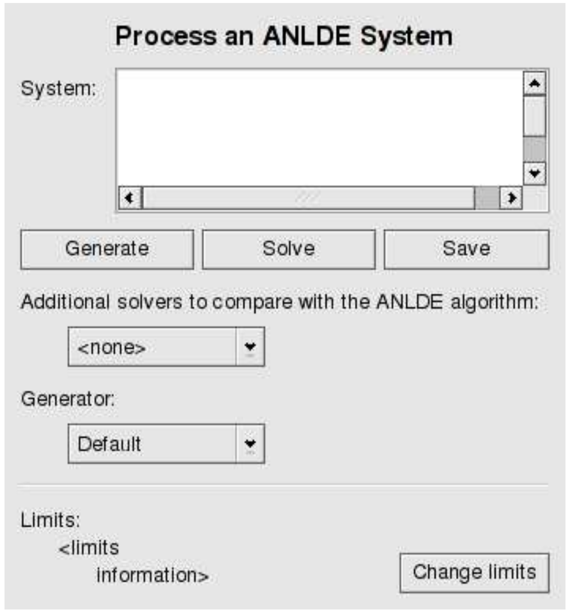


Figure 7: Process an ANLDE System form

Components:

- text area for manual input of ANLDE system (sect. 3.3.1 ANLDE System Format),
- button "Generate",
- button "Solve",
- button "Save" ,
- list "Alternative solver" (sect. 3.3.5 Alternative Solver List Format),

- button "Change limits" (sect. 3.2.7 User Limits),
- limits information (sect. 3.3.9 Limits Format).

References:

Use case: Process an ANLDE System.

Requirements: EU1a.

3.2.2 Process a Set of ANLDE Systems

Description: Process form gives a user an opportunity to work with a set of ANLDE systems, which can be generated or loaded from a file, and then send the systems to the server for solving. A user can see her/his limits and may change them, if necessary, with a "Change limits" button. Limits values are used as parameters for generating ANLDE system. See Fig. 8.

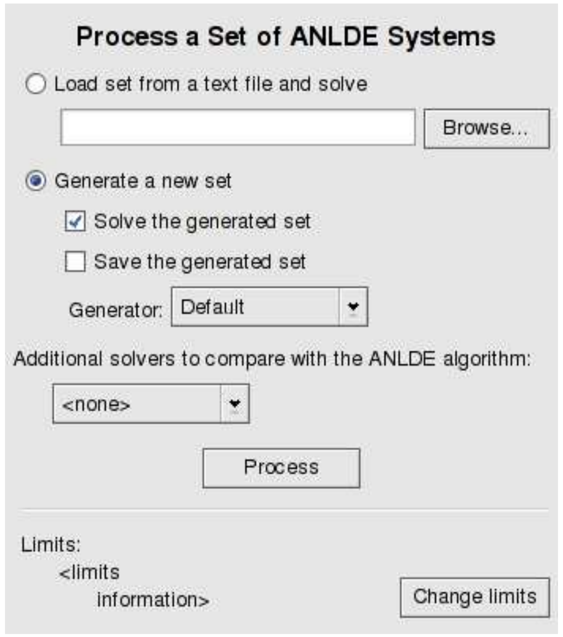


Figure 8: Process a Set of ANLDE Systems form

Components:

- radio button ”Load a set from a text file”,
- text field for file name (format depends on operating system, sect. 3.3.2 ANLDE System Set Format),
- radio button “Generate a new set” (selected by default),
- check box “Solve the generated set” (checked by default),
- check box “Save generated set” (does not checked by default),
- list “Alternative solver” (sect. 3.3.5 Alternative Solver List Format),
- button “Process”,
- limits information (sect. 3.3.9 Limits Format),
- button “Change limits” (sect. 3.2.7 User Limits).

References:

Use case: Process a Set of ANLDE Systems.

Requirements: EU1b.

3.2.3 Log In

Description: Log In form allows a registered user to identify her/himself in the web-system. A registered user enters her/his nickname and password. There is a user who has an opportunity to Log In the web-system as a system administrator with access to the administration forms. A ”Register” button is also available; thus using this form, a user can register her/himself whenever she/he wants. See Fig. 9.

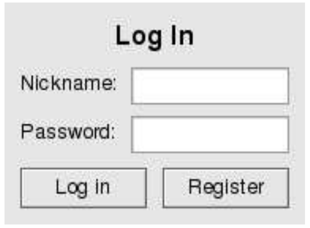


Figure 9: Log In form

Components:

- text field for nickname (sect. 3.3.10 User Information format),
- text field for password (sect. 3.3.10 User Information format),
- button ”Log In”,
- button ”Register”.

References:

Use case: Log In.

Requirements: EU2d.

3.2.4 Activity Statistics

Description: This form is accessible only for the system administrator. It allows to select necessary characteristics and get a report on activity statistics for the current month. Two lists are available:

- domain: nicknames or IP addresses;
- metrics: number of generated systems, input systems, solved systems, acknowledged systems, resources, etc (sect. 3.3.7).

See Fig. 10.

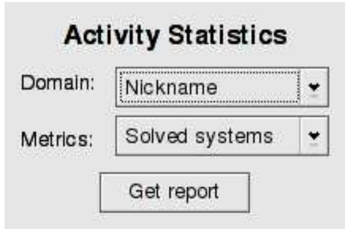


Figure 10: Activity Statistics Form

Components:

- list of activity domains (sect. 3.3.6 Activity Domain format),
- list of activity metrics (sect. 3.3.7 Activity Metrics format),
- button ”Get report”.

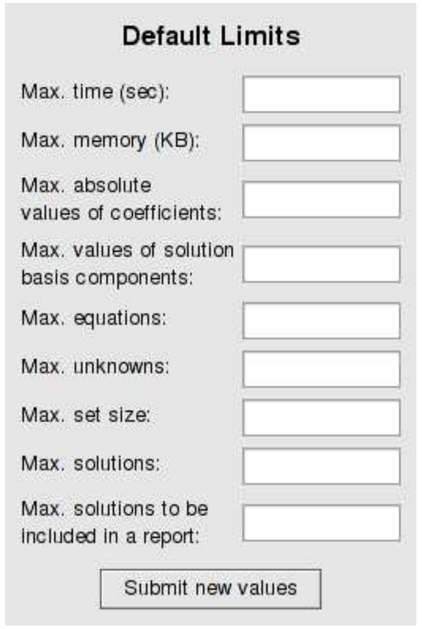
References:

Use case: Get Statistics.

Requirements: EU3b.

3.2.5 Default Limits

Description: This form is accessible only for the system administrator. It allows changing the default limits of solving ANLDE systems. The form consists of a set of text fields, where necessary default limits are written (only nonnegative integer values). See Fig. 11.


Default Limits

Max. time (sec):	
Max. memory (KB):	
Max. absolute values of coefficients:	
Max. values of solution basis components:	
Max. equations:	
Max. unknowns:	
Max. set size:	
Max. solutions:	
Max. solutions to be included in a report:	

Figure 11: Default Limits form

Components:

- text field for maximal time of solution (sect. 3.3.9 Limits format);
- text field for maximal memory used in solution (sect. 3.3.9 Limits format);
- text field for maximal value of coefficients (sect. 3.3.9 Limits format);

- text field for maximal value of basis components in solution (sect. 3.3.9 Limits format);
- text field for maximal number of equations in ANLDE system (sect. 3.3.9 Limits format);
- text field for maximal number of unknowns in ANLDE system (sect. 3.3.9 Limits format);
- text field for maximal size of set of ANLDE systems, which are solved (sect. 3.3.9 Limits format);
- text field for maximal number of solutions (sect. 3.3.9 Limits format);
- text field for maximal number of solutions to be included in a report (sect. 3.3.9 Limits format);
- button ”Submit new values”.

References:

Use case: Manage Default Limits.

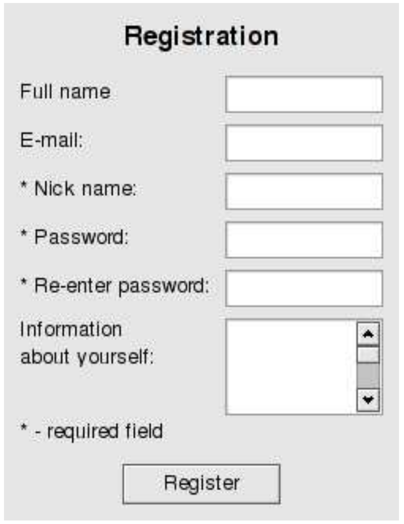
Requirements: EU3c.

3.2.6 Registration

Description: The form allows a user to register in the web system. In order to do it he/she should enter his/her personal information, nickname and password; text fields that marked with '*' are required. Brief textual description for this form also contains restrictions on field sizes. See Fig. 12.
Components:

- text field for nickname (sect. 3.3.10 User Information format), required;
- text field for password (sect. 3.3.10 User Information format), required;
- text field for re-entering password (sect. 3.3.10 User Information format), required;
- text field for full name (sect. 3.3.10 User Information format), the same as a nickname by default;
- text field for e-mail (sect. 3.3.10 User Information format);
- text area for personal information (sect. 3.3.10 User Information format);
- Button ”Register”.

References:



Registration	
Full name	
E-mail:	
* Nick name:	
* Password:	
* Re-enter password:	
Information about yourself:	
* - required field	
Register	

Figure 12: Registration form

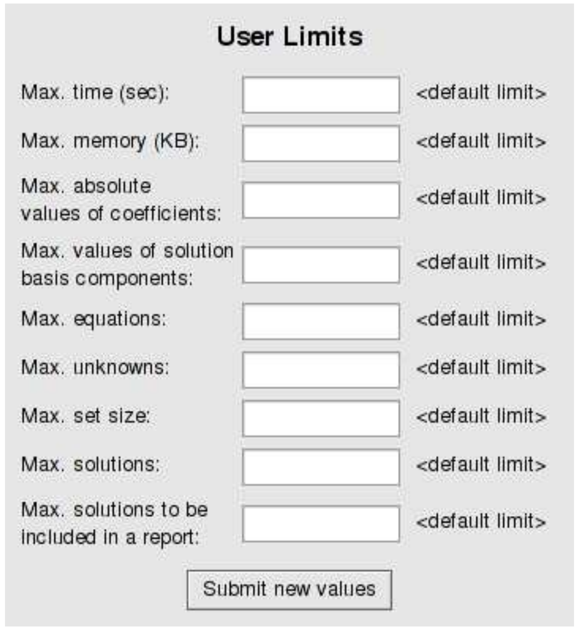
Use case: Register a User.

Requirements: EU2a.

3.2.7 User Limits

Description: The form allows a user to change her/his own limits on solving ANLDE systems. It consists of the set of text fields, where user limits are entered. But they must not exceed the default limits. For user's convenience default limit corresponding to each text field for entering limit is shown. See Fig. 13.
Components:

- text field for maximal time of solution (sect. 3.3.9 Limits format);
- text field for maximal memory used in solution (sect. 3.3.9 Limits format);
- text field for maximal values of coefficients (sect. 3.3.9 Limits format);
- text field for maximal value of basis components in solution (sect. 3.3.9 Limits format);
- text field for maximal number of equations in ANLDE system (sect. 3.3.9 Limits format);
- text field for maximal number of unknowns in ANLDE system (sect. 3.3.9 Limits format);



User Limits		
Max. time (sec):		<default limit>
Max. memory (KB):		<default limit>
Max. absolute values of coefficients:		<default limit>
Max. values of solution basis components:		<default limit>
Max. equations:		<default limit>
Max. unknowns:		<default limit>
Max. set size:		<default limit>
Max. solutions:		<default limit>
Max. solutions to be included in a report:		<default limit>
Submit new values		

Figure 13: User Limits form

- text field for maximal size of set of ANLDE systems, which are solved (sect. 3.3.9 Limits format);
- text field for maximal number of solutions (sect. 3.3.9 Limits format);
- text field for maximal number of solutions to be included in a report (sect. 3.3.9 Limits format);
- button "Submit new values".

References:

Use case: Manage User Limits.

Requirements: EU2c.

3.2.8 Manage Users

Description: Accessible only for a system administrator. She/He enters the nickname of a target user and presses "Edit" button. See Fig. 14.

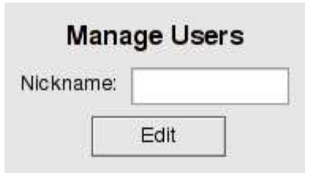


Figure 14: Manage Users form

Components:

- text field for user's nickname (sect. 3.3.9 User Information format),
- button "Edit" (sect. 3.2.9 User Information).

References:

Use case: Manage Users.

Requirements: EU3a.

3.2.9 User Information

Description: The web system sends this form to the user. System administrator may change this information for any user; other users may change only her/his own information. See Fig. 15.

Components:

- text field for user's nickname (sect. 3.3.10 User Information format),
- text field for user's E-mail (sect. 3.3.10 User Information format),
- text area for extra user's information (sect. 3.3.10 User Information format),
- check box "Change password" (doesn't checked by default),
- text field for user's password (sect. 3.3.10 User Information format),
- text field for re-entering password (sect. 3.3.10 User Information format),
- check box "Remove account" (doesn't checked by default, displayed in admin form only, regular user don't see this check box),
- button "Submit new values".

References:

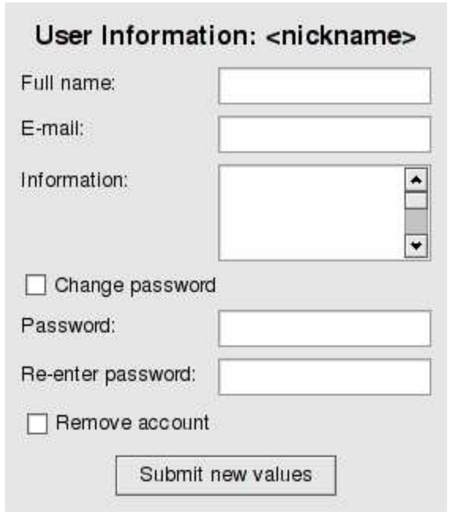


Figure 15: User Information form

Use case: Manage Users.

Requirements: EU3a.

3.2.10 General Notes

Description: This form allows a user to send an opinion about the web-system (as whole). See Fig. 16.

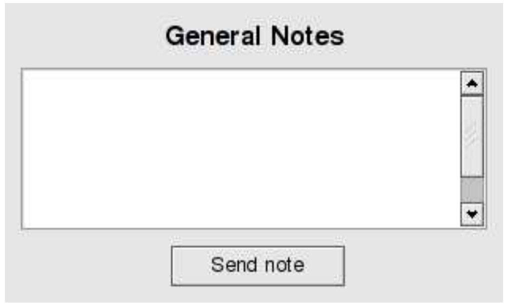


Figure 16: General Notes

Components:

- text area for a textual note (sect. 3.3.11 Note format),
- button "Send note".

References:

Use case: Send a Note.

Requirements: EU2b.

3.2.11 Notes on Solution

Description: This form allows a user to send an opinion about the last processed ANLDE system (or ANLDE systems set). The user chooses one of two possibilities:

1. Agree with solution of the processed ANLDE system (set). In this case the user may or may not attach the ANLDE system (set) to the note.
2. Disagree with solution of the processed ANLDE system (set). In this case the processed ANLDE system (set) is always attached to the note (automatically).

See Fig. 17.

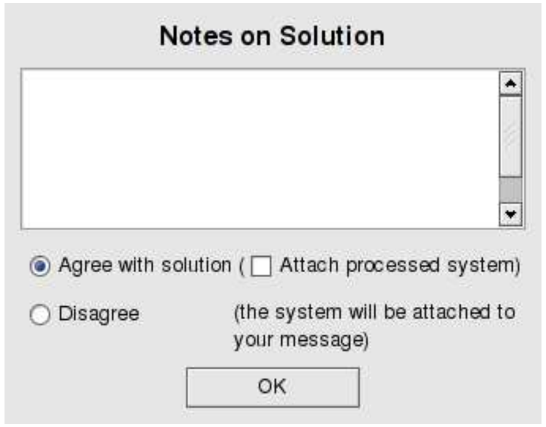

Notes on Solution
● Agree with solution (☐ Attach processed system)
○ Disagree
(the system will be attached to your message)
OK

Figure 17: Notes on Solution

Components:

- text area for a textual note (sect. 3.3.11 Note format),
- radio button "Agree with solution" (selected by default),
- radio button "Disagree",
- check box "Attach solved system" (doesn't checked by default),
- button "OK".

References:

Use case: Send a Note.

Requirements: EU2b.

3.3 IO Data Formats

3.3.1 ANLDE System Format

Format:

```
# Comment
x1 + x2 + ...  + xK2 = c11*x1 + c12*x2 + ...  + c1N*xN
x[K2+1] + x[K2+2] + ...  + xK3 = c21*x1 + c22*x2 + ...  + c2N*xN
...
x[KM+1] + x[KM+2] + ...  + xN = cM1*x1 + cM2*x2 + ...  + cMN*xN
```

Description: The format represents an ANLDE system. c11, c12, ..., c1N, c21, c22, ..., cMN are coefficients (optional, default value is 1). x1, x2, ..., xN are unknowns, may appear in any order, some may be skipped. If there is no unknowns after the "=" sign, write "0". Each unknown must appear in the left-hand side of some equation at most one time. Blank and comment lines are ignored.

Sample:

```
# Sample ANLDE system
x1 + x4 = 2*x1 + 3*x3
x2 + x3 = x1 + 2*x2 + x3
```

Corresponding forms: sect. 3.2.1 Process an ANLDE System.

3.3.2 ANLDE System Set Format

Format:

```
<ANLDE system 1>
%
```

```
<ANLDE system 2>
%
...
<ANLDE System N>
```

Description: The format represents ANLDE System Set ⟨ANLDE System 1⟩, ..., ⟨ANLDE System N⟩. Each system is in the ANLDE System Input Format (sect. 3.3.1). Blank and comment lines are ignored. String with symbols(s) '`%`' is a delimiter for ANLDE systems. These strings may additionally contain blank symbols ('␣', '`\t`') only.
Corresponding forms: sect. 3.2.2 Process a set of ANLDE systems.

3.3.3 ANLDE System Solution Format

Description: Format represents a report on solution of an ANLDE System. Each report includes

1. Test ANLDE system (ANLDE system format, sect. 3.3.1).
2. Number of solutions (nonnegative integer).
3. Algorithm name, system time, work time, memory usage, solving result (one of the following: solved, limit exceeded, abnormal solver termination), see sect. 3.3.9 for the these attributes.
4. Server hardware and software characteristics where solving algorithms work (Requirement Specification, Configuration Requirements).
5. List of solutions found by each algorithm (depends on limit view in limits format, see sect. 3.3.9).

Sample:

1. Test ANLDE system:

   ```
   x1 + x4 = 2*x1 + 3*x3
   x2 + x3 = x1 + 2*x2 + x3
   ```
   Number of solutions: 1

2. Performance metrics of the algorithms:

Algorithm	System time(sec)	Work time(sec)	Memory usage(KB)	Solving result
anlde	0	0.580	2192	solved
slopes	0	1.168	2192	solved

3. Solving machine characteristics:

- **CPU:** IA32, 1200 MHz;
- **RAM:** 256 MB.
- **Operating system:** Linux 2.4.19
- **Priority(nice):** 12

4. Solutions of test ANLDE system:

anlde:

$$\begin{array}{c|cccc} & x_1 & x_2 & x_3 & x_4 \\ h^{(1)} & 0 & 0 & 1 & 3 \end{array}$$

slopes:

$$\begin{array}{c|cccc} & x_1 & x_2 & x_3 & x_4 \\ h^{(1)} & 0 & 0 & 1 & 3 \end{array}$$

Corresponding forms: sect. 3.2.1 Process an ANLDE System.

3.3.4 ANLDE System Set Solution Format

Description: Format represents a report on solution for a set of ANLDE systems. Each report includes

1. Number of ANLDE systems in the set;
2. ANLDE systems metrics: minimum (positive int), average (positive real,one digit for fractional part) and maximum (positive int) dimensions (equations and unknowns) of test ANLDE systems; average (positive real, one digit for fractional part) and maximum (positive int) coefficients of test ANLDE systems; minimum (nonnegative int), average (nonnegative real, one digit for fractional part); maximum (nonnegative int) number of solutions.
3. Algorithm metrics: algorithm name, sum system time (Limits format), sum work time (Limits format), maximal memory usage (Limits format), solving result (one of the following: solved, limit exceeded, abnormal solver termination).
4. Server hardware and software characteristics (Requirement Specification, Configuration Requirements).

Sample:

1. Number of ANLDE systems in the set: 3
2. The set characteristics:

   minimum equations: 3

average equations: 5.0

maximum equations: 7

minimum unknowns: 4

average unknowns: 7.0

maximum unknowns: 10

average coefficients: 4.2

maximum coefficients: 10

minimum number of solutions: 0

average number of solutions: 3.3

maximum number of solutions: 10

3. Algorithm metrics:

Algorithm name	Summary system time(sec)	Summary work time(sec)	Maximum memory usage(KB)	Solving result
anlde	0	0.580	2192	solved
slopes	0.23	5.172	2192	solved

4. Solving machine characteristics:

- CPU: IA32, 1200 MHz;
- RAM: 256 MB.
- Operating system: Linux 2.4.19
- Priority(nice): 12

3.3.5 Alternative Solver List Format

Description: This is a list of available alternative solvers to compare with the ANLDE algorithm. The List includes special "none" item to disable the comparison feature. At most one alternative solver may be chosen.

3.3.6 Activity Domain List Format

Description: The format represents an activity domain list. There are two items in this list: "Nickname" and "IP address". The first one corresponds to statistics by all registered users, the second one is for statistics by all hosts.
Corresponding forms: sect. 3.2.4 Activity statistics.

3.3.7 Activity Metrics List Format

Description: The format represents an activity metrics list. The list contains the items:

Sessions is the total number of sessions.

Processed ANLDE systems is the total number of processed (input with an attempt to solve) ANLDE systems,

Generated ANLDE systems is the total number of generated (by Web-SynDic) ANLDE systems,

Solved ANLDE systems is the total number of successfully solved (from point of view of the anlde solver) ANLDE systems,

Acknowledged ANLDE systems is the total number of acknowledged (explicitly by user(s)) systems,

Discrepancies for solvers is the total number of ANLDE systems that are solved with a discrepancy (alternative algorithm gives a different solution comparing with the anlde solver),

Sum system time is sum (for all ANLDE systems and their sets) system (from OS point of view) time usage,

Sum work time is sum (for all ANLDE systems and their sets) work (from user point of view) time usage,

Sum session time is sum session work time (time, used by all sessions),

Corresponding forms: sect. 3.2.4 Activity Statistics.

3.3.8 Statistics Report Format

Description: The format represents a statistics report. The report is a table with 2 columns. The first column contains items according to the selected domain (nickname or IP address). The second column contains statistics metrics.
Sample: Statistics report during the October.

Generated at Fri Oct 31 12:22:29 MSK 2003.

nickname	summary used work time
guest	49.40
guest01	10.10
guest2	50.50
guest_2	40.40
user13	20.20
user2	111.00
user21	30.30
user3	50.50
user31	9.0

Corresponding forms: sect. 3.2.4 Activity Statistics.

3.3.9 User limits Format

Description: The format represents user limits. There are eight items in this format:

1. Maximal time allowed for solving process (in seconds, positive integer, checked during a solving process).
2. Maximal memory allowed for solving process (in kilobytes, positive integer, checked during a solving process).
3. Maximal value for coefficients of ANLDE system(s) (positive integer, checked i) after inputing the ANLDE system and ii) during generating process).
4. Maximal value for any component in a basis solution (positive integer, checked during generating and solving processes).
5. Maximal number of equations in ANLDE system (positive integer, checked after inputing the ANLDE system and during generating process).
6. Maximal number of unknowns in ANLDE system (positive integer, checked after inputing the ANLDE system and during generating process).
7. Maximal size of an ANLDE systems set (positive integer, checked after inputing the set and during generating process).
8. Maximal number of basis solutions (positive integer, checked during generating and solving processes).
9. Maximal number of basis solutions to be included in a report on solution (non-negative integer, untestable because using only for output solutions to a user).

Corresponding forms: sect. 3.2.5 Default Limits, sect. 3.2.7 User Limits.

3.3.10 User Information Format

Description: The format used for user information data. There are six items in this format.

1. Nickname consists of following characters: Latin letters, digits and '_' symbol. It is case sensitive. Maximum length is 32 characters.
2. Password consists of following characters: Latin letters, digits and punctuation marks. It is case sensitive. Maximum length is 32 characters.
3. Re-entering password format is the same as password format.
4. Full name is a text (text). Maximum length is 256 characters.
5. E-mail is a text (text). Maximum length is 256 characters.
6. Personal information is a text (no more than 256 characters).

Corresponding forms: sect. 3.2.6 Registration, sect. 3.2.8 Manage Users, sect. 3.2.9 User Information.

3.3.11 Note Format

Description: The note format is a text (no longer than 4096 symbols).
Corresponding forms: sect. 3.2.10 General Notes, sect. 3.2.11 Notes on Solution.

3.3.12 Acknowledgment Format

Description: This format represents a different types of acknowledgments. The acknowledgment format is a one of following types:

1. Log in acknowledgment ("You are logged in as 'nickname'.").
2. Modification acknowledgment ("Changes have been submitted. Thank you.").
3. Sending note acknowledgment ("Your message have been sent. Thank you.").
4. Registration acknowledgment ("Thank you for registering.").

Corresponding forms: sect. 3.2.3 Log In, sect. 3.2.5 Default Limits, sect. 3.2.7 User Limits, sect. 3.2.8 Manage Users, sect. 3.2.10 General Notes, sect. 3.2.11 Notes on Solution, sect. 3.2.6 Registration.

3.3.13 Error Message Format

Description: Format represents a different types of error messages. The error message format is one of following types:

1. Invalid nickname or password.
2. Invalid value (also empty).
3. Too big message.
4. Invalid ANLDE system format.
5. ??? Chosen be absent.
6. Registration error.
7. The ANLDE system does not satisfy the user limits.
8. Server is already processing your task.

Corresponding forms: sect. 3.2.3 Log In, sect. 3.2.5 Default Limits, sect. 3.2.7 User Limits, sect. 3.2.8 Manage Users, sect. 3.2.10 General Notes, sect. 3.2.11 Notes on Solution, sect. 3.2.1 Process an ANLDE System, sect. 3.2.2 Process a Set of ANLDE Systems.

3.3.14 Process Message Format

Description: The process message has to be shown to a user when ANLDE solving or generating process takes more than 20 seconds (Req. AP3). Possible message types are:

1. Solving your ANLDE system(s). Wait, please...
2. Generating ANLDE system(s) to you. Wait, please...

Corresponding forms: sect. 3.2.1 Process an ANLDE System, sect. 3.2.1 Process a Set of ANLDE Systems.

3.4 Notes on implementation

User interface for Web-SynDic will be implemented using Java Server Pages and Servlets technologies. Web server produces HTML 4.01 compliant pages and send them to a client (browser). Forms will be represented using HTML 4.01 form tags
(see `http://www.w3.org/TR/html401/interact/forms.html`).

4 Behavioral model

4.1 Work with Web-SynDic

Working with Web-SynDic requires a user to start a session; she/he may work only within a session. Figure 18 shows the behavior of session starting and finishing.

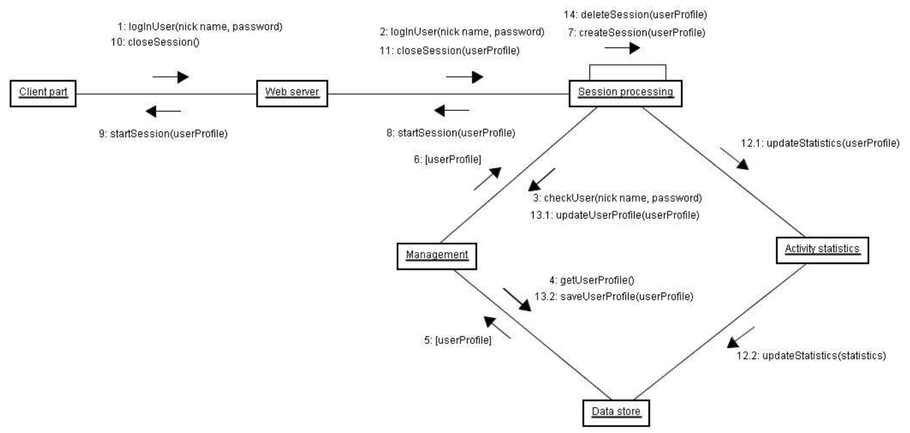

1: logInUser(nick name, password)
10: closeSession()
2: logInUser(nick name, password)
11: closeSession(userProfile)
14: deleteSession(userProfile)
7: createSession(userProfile)
Client part
Web server
Session processing
9: startSession(userProfile)
8: startSession(userProfile)
6: [userProfile]
3: checkUser(nick name, password)
13.1: updateUserProfile(userProfile)
12.1: updateStatistics(userProfile)
Management
Activity statistics
4: getUserProfile()
13.2: saveUserProfile(userProfile)
5: [userProfile]
12.2: updateStatistics(statistics)
Data store

Figure 18: Work with Web-SynDic Collaboration Diagram

1: logInUser(nickName,password) sends user's nickname and password to log In,

2: logInUser(nickName,password) forwards the nickname and password to session processing (a new session is going to be started),

3: checkUser(nickName,password) sends user's nickname and password to Management subsystem for checking (registered user or not),

4: getUserProfile() is a request to get the user profile from Data Store,

5: [userProfile] is taken from Data store, if any. If the profile does not exist, then the corresponding message ir returned,

6: [userProfile] is transmitted to session processing (or the message on nonexistance),

7: createSession(userProfile) creates session, identified with the user nickname, or creates a session for anonymous (nonregistered) user, 8: startSession(userProfile) sends initial data to the web server,

9: startSession(userProfile) sends initial web form to the user,

10: closeSession() sends message to close the session,

11: closeSession(userProfile) sends message to close the session,

12.1: updateStatistics(userProfile) sends update statistics to Activity statistics subsystem,

12.2: updateStatistics(statistics) updates statistics in Data store,

13.1: updateUserProfile(userProfile) sends update user profile to management for the registered user,

13.2: saveUserProfile(userProfile) saves the user profile,

14: deleteSession(userProfile) removes the current session.

This form is a main part for presentation Web-SynDic to user. It is displayed whenever a user starts her/his web client and is closed after explicit finishing the work or implicit closing the browser.

4.2 Log In

Log In behaviour is shown in Figure 19. A user inputs her/his nikname and password

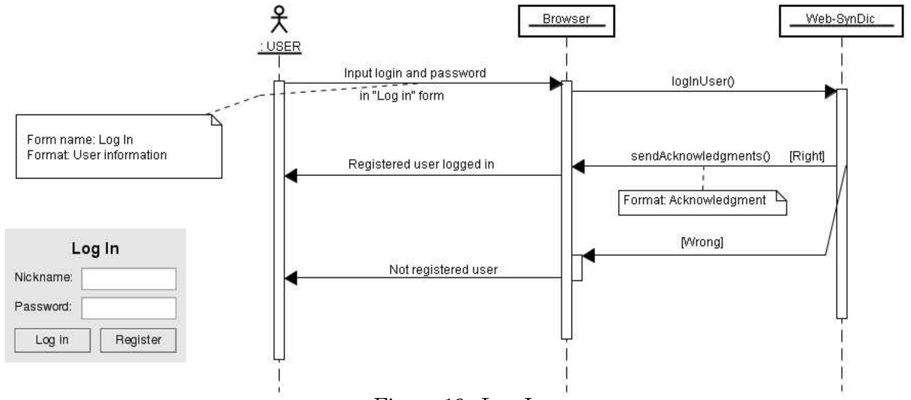


Figure 19: Log In

(sect. 3.3.10 User information format). If they are correct, then the login is successful and

the user may work with Web-SynDic as a registered one. Othewise, the invalid login message is displayed (sect. 3.3.13 Error message format, 1) in the “Content” part and the user may work only as anonymous (nonregistered) user.

The “Log in / status” part of the main Web-SynDic page displays the result of the user login and user’s current status. “Log in” form is always displayed in the ”Log in / status” part during the session. For a registered user the user menu also contains access to additional management functions.

4.3 Process an ANLDE system

For processing a test ANLDE system, a user may initialize four flows, see Figure 20.

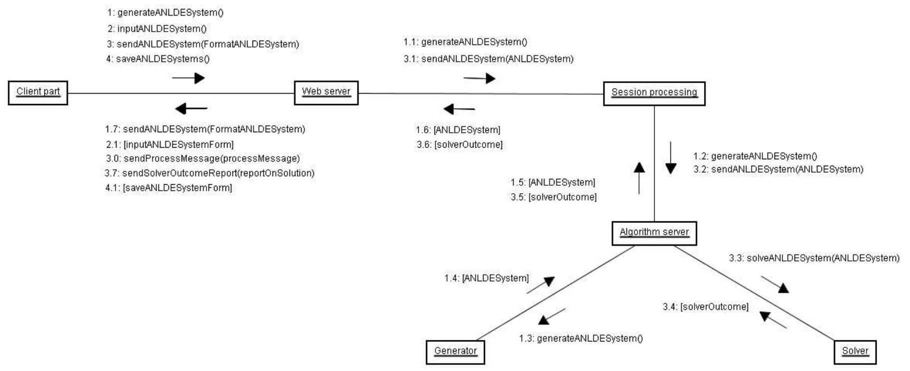

1: generateANLDESystem()
2: inputANLDESystem()
3: sendANLDESystem(FormatANLDESystem)
4: saveANLDESystems()
1.1: generateANLDESystem()
3.1: sendANLDESystem(ANLDESystem)
Client part
Web server
Session processing
1.7: sendANLDESystem(FormatANLDESystem)
2.1: [inputANLDESystemForm]
3.0: sendProcessMessage(processMessage)
3.7: sendSolverOutcomeReport(reportOnSolution)
4.1: [saveANLDESystemForm]
1.6: [ANLDESystem]
3.6: [solverOutcome]
1.2: generateANLDESystem()
3.2: sendANLDESystem(ANLDESystem)
1.5: [ANLDESystem]
3.5: [solverOutcome]
Algorithm server
3.3: solveANLDESystem(ANLDESystem)
1.4: [ANLDESystem]
3.4: [solverOutcome]
1.3: generateANLDESystem()
Generator
Solver

Figure 20: Process an ANLDE system Collaboration Diagram

The first flow is for generating a test ANLDE system.

1: generateANLDESystem() sends a signal to the web server that a user requires to generate an ANLDE system,

1.1: generateANLDESystem() forwards the request to Session processing (generation is within session activity),

1.2: generateANLDESystem() forwards the request to the algorithm server,

1.3: generateANLDESystem() calls an appropriate generator and the generating process is started,

1.4: [ANLDESystem] (generated system) is returned to the algorithm server, 1.5: [ANLDESystem] is transmitted to the current session,

1.6: [ANLDESystem] is sent back to the web server for producing the web form,

1.7: sendANLDESystem(ANLDESystem) sends the form with the generated ANLDE system to the user.

The second flow for manual input user's ANLDE system.

2: inputANLDESystem() sends request for a web form to input a test ANLDE system,

2.1: [inputANLDESystemForm] (web form for the input) is sent to the user; then she/he can input a test ANLDE system.

The third flow is for solving a given test ANLDE system.

3: sendANLDESystem(FormatANLDESystem) sends the given ANLDE system (generated by Web-Syndic or input by the user) in ANLDE format,

3.0: sendProcessMessage(processMessage) sends a process message,

3.1: sendANLDESystem(ANLDESystem) forwards the given ANLDE system to session processing subsystem,

3.2: sendANLDESystem(ANLDESystem) forwards ANLDE System to the algorithm server,

3.3: solveANLDESystem(ANLDESystem) calls an appropriate solver and starts the solving process,

3.4: [solverOutcome] (results of solving) is returned from solver to the algorithm server,

3.5: [solverOutcome] is sent to session processing,

3.6: [solverOutcome] is forwarded to the web server for producing the corresponding web form (page) with the report on solution,

3.7: sendSolverOutcomeReport(reportOnSolution) visualizes the report on solution for the user.

The last flow is for saving ANLDE system locally (as a text file).

4: saveANLDESystems is a request for saving the given ANLDE system,

4.1: [saveANLDESystemForm] is a web form to a user for saving the ANLDE system.

A user chooses a feature “process an ANLDE system” using a corresponding item in the main menu. The ”Process an ANLDE system” form will be displayed in the ”Content” part of the main web page.

The user enters an ANLDE system manually in the text area or generate it (sect. 3.3.1 ANLDE system format). For generation process the user may choose a generator (corresponding item “Generator”). If the user has not choosen a generator, then “gauss” algorithm is used as default. If the user presses “Generate” button, then the process message is displayed in the “Content part” (sect. 3.3.14 Process message format, 2); after the generation, form “Process an ANLDE system” is reloaded with the generated ANLDE system in the text area.

Also the user can choose an alternative solving algorithm (corresponding list “Alternative Solver”). After entering the ANLDE system, the user presses the ”Solve” button. If the ANLDE system is incorrect, then the error message is displayed in the “Content” part (sect. 3.3.13 Error message format, 4). If the set of ANLDE systems does not satisfy the user limits, then the error message is also displayed in the ”Content” part (sect. 3.3.13 Error message format, 7).

If the entered ANDLE system is valid, then the process message is displayed in the “Content” part (sect. 3.3.14 Process message format). After solving the system, a report on solution is displayed in “Content” (sect. 3.3.3 ANLDE system solution format).

If the user presses “Save” button, then the ANLDE system is opened in a new browser window (ANLDE system format, comments on saving will be added).

If there is an error in solving process, then the error is displayed in the report on solution (sect. 3.3.3 ANLDE system solution format).

For convenience, user limits information is shown in this form. If the user presses ”Change limits” button, then the ”User limits” form is displayed in the ”Content” part and after submitting changes ”Process a set of ANLDE systems” form is loaded in “Content” with new user limits.

This form corresponds to the “Process an ANLDE systems” use case. See Fig. 21.

4.4 Process a set of ANLDE systems

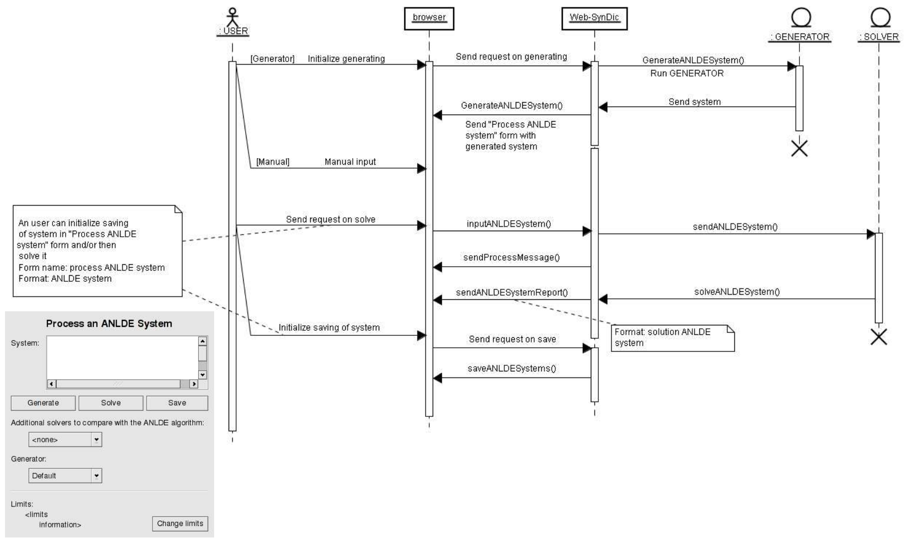
For processing a set of test ANLDE systems, a user may initialize four flows, see Figure 22.

The first flow is for generating such a set.

1: generateANLDESystemSet() sends a signal to the web server that a user requires to generate a set of test ANLDE systems,

1.1, 1.2: generateANLDESystemSet() sends/forwards the request to the algorithm server,

1.3: generateANLDESystemSet() calls an appropriate generator and starts the generating process,



An user can initialize saving of system in "Process ANLDE system" form and/or then solve it
Form name: process ANLDE system
Format: ANLDE system

Figure 21: Process an ANLDE System

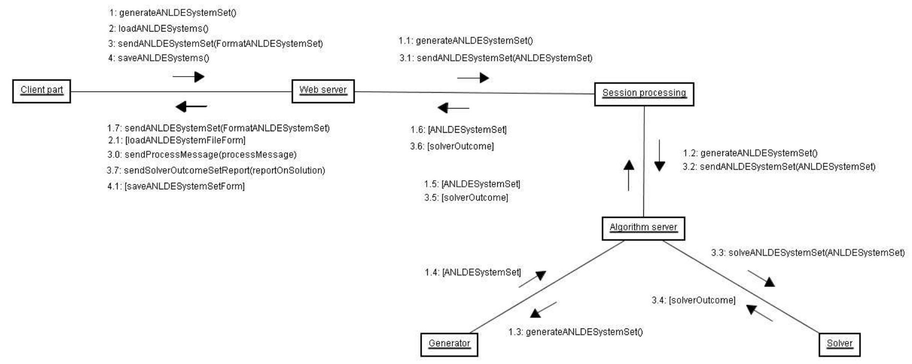

1: generateANLDESystemSet()
2: loadANLDESystems()
3: sendANLDESystemSet(FormatANLDESystemSet)
4: saveANLDESystems()
1.1: generateANLDESystemSet()
3.1: sendANLDESystemSet(ANLDESystemSet)
Client part
Web server
Session processing
1.7: sendANLDESystemSet(FormatANLDESystemSet)
2.1: [loadANLDESystemFileForm]
3.0: sendProcessMessage(processMessage)
3.7: sendSolverOutcomeSetReport(reportOnSolution)
4.1: [saveANLDESystemSetForm]
1.6: [ANLDESystemSet]
3.6: [solverOutcome]
1.2: generateANLDESystemSet()
3.2: sendANLDESystemSet(ANLDESystemSet)
1.5: [ANLDESystemSet]
3.5: [solverOutcome]
Algorithm server
3.3: solveANLDESystemSet(ANLDESystemSet)
1.4: [ANLDESystemSet]
3.4: [solverOutcome]
1.3: generateANLDESystemSet()
Generator
Solver

Figure 22: Process a set of ANLDE systems Collaboration Diagram

1.4: [ANLDESystemSet] is a required generated set of ANLDE systems,

1.5: [ANLDESystemSet] is forwarded to the session,

1.6: [ANLDESystemSet] is returned to the web server,

1.7: sendANLDESystemSet(ANLDESystemSet) sends the form with the set of ANLDE systems to the user.

The second flow is for loading a set of ANLDE systems from user's file.

2: loadANLDESystem() sends request to getting a web form for loading a set of ANLDE systems,

2.1: [inputANLDESystemForm] (the web form) is sent to the user for loading a set of ANLDE systems.

The third flow is for solving a set of ANLDE systems.

3: sendANLDESystemSet(FormatANLDESystemSet) sends a given set of ANLDE systems in ANLDE format to,

3.0: sendProcessMessage(processMessage) sends a process message (about solving process),

3.1: sendANLDESystemSet(ANLDESystemSet) sends a given set of ANLDE systems to the web server,

3.2: sendANLDESystemSet(ANLDESystemSet) forwards the set to the algorithm server,

3.3: sendANLDESystemSet(ANLDESystemSet) calls an appropriate solver and starts the solving process,

3.4: [solverOutcome] (solution result) is returned by the solver to the algorithm server,

3.5: [solverOutcome] is processed and sent into the current session,

3.6: [solverOutcome] is processed and sent to the web server,

3.7: sendSolverOutcomeSetReport(reportOnSolution) produces a report on solution and sends th corresponding form to the user.

The last flow is for saving ANLDE system.

4: saveANLDESystems() is a request for saving a given set of ANLDE systems,

4.1: [saveANLDESystemForm] (the form for saving) is produced and sent by the web server to user.

A user chooses processing a set of ANLDE systems using the corresponding item in the main menu. The "Process a set of ANLDE systems" is displayed in the "Content" part of the main web page.

The first possibility for user is to input a set of ANLDE systems. The set may be loaded from file. The user chooses "Load set from a text file and solve" item in the form. Then the user presses the “Browse...” button and chooses the file. Also the user can choose an alternative solving algorithm (corresponding item in the list “Alternative Solver”).

The second possibility is to generate a set of ANLDE systems. The user chooses “Generate new set” item. Also the user can choose a generator (corresponding item in the “Generator”). If the user does not choose a generator, then “gauss” algorithm is used by default. After the generation, the user can choose solve and/or save a set of ANLDE systems by selecting corresponding check boxes.

For processing a set of ANLDE systems, a user should press the “Process” button. The process message will be displayed in the “Content” part (sect. 3.3.14 Process message format). Then the web system checks the given set of ANLDE systems. If the set is incorrect, then an error message is displayed in “Content” (sect. 3.3.13 Error message format, 4). If the set does not satisfy user limits, then an error message is also displayed in “Content” (sect. 3.3.13 Error message format, 7).

If save item was selected, then after generating ANLDE systems the set is opened in a new browser window (ANLDE system set format, comments on saving will be added). If no check box is selected, then an error message is displayed in the “Content” part (3.3.13 Error message format, 5).

If the set of ANLDE systems is valid, then the report on solution is displayed in the “Content” part after processing (sect. 3.3.2 ANLDE system set solution format). If there was an error in the solving process, then the error is displayed in the report on solution (sect. 3.3.4 ANLDE system set solution format).

For convenience, user limits information is shown in this form too. If the user presses “Change limits” button, then the “User limits” form is displayed in the ”Content” part and, after submitting changes ”Process a set of ANLDE systems” form, is loaded in the ”Content” part with new user limits.

This form corresponds to the ”Process a set of ANLDE systems” use case, see Fig. 23

4.5 Register a user

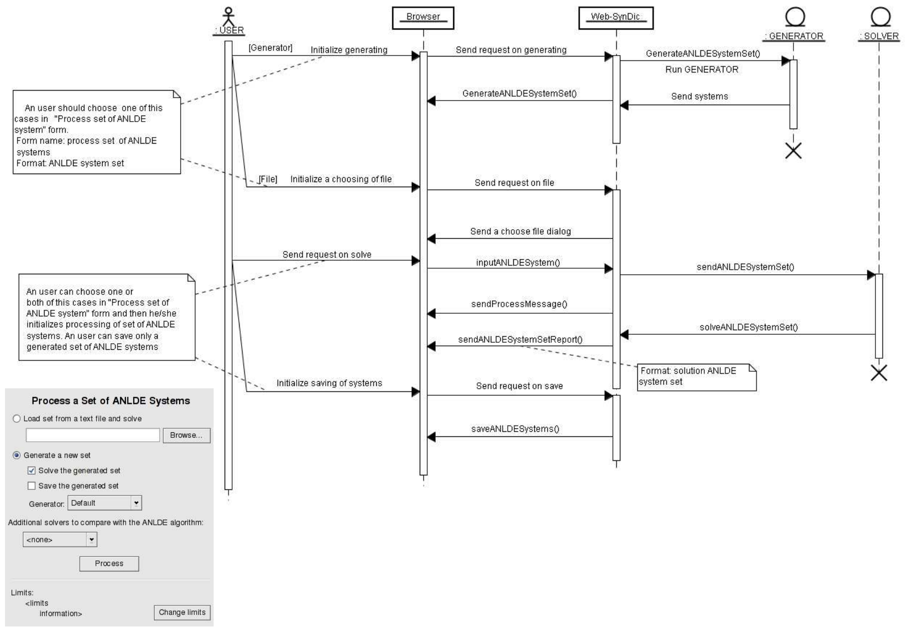
The web system allows a non-registered user to register when she/he wishes. The registration process has the following behaviour, see Figure 24.

1: registerUser(userProfile) , a user sends a request on registration to the web server;

2: registerUser(userProfile) sends the request to the session processing;

3: registerUser(userProfile) forwards the request to the management;

4: addUserProfile(userProfile) adds a new user profile to data store;


: USER
Browser
Web-SynDic
: GENERATOR
: SOLVER
[Generator]
Initialize generating
Send request on generating
GenerateANLDESystemSet()
Run GENERATOR
GenerateANLDESystemSet()
Send systems
An user should choose one of this cases in "Process set of ANLDE system" form.
Form name: process set of ANLDE systems
Format: ANLDE system set
[File]
Initialize a choosing of file
Send request on file
Send a choose file dialog
Send request on solve
inputANLDESystem()
sendANLDESystemSet()
An user can choose one or both of this cases in "Process set of ANLDE system" form and then he/she initializes processing of set of ANLDE systems. An user can save only a generated set of ANLDE systems
sendProcessMessage()
solveANLDESystemSet()
sendANLDESystemSetReport()
Format: solution ANLDE system set
Initialize saving of systems
Send request on save
saveANLDESystems()

Process a Set of ANLDE Systems

( ) Load set from a text file and solve

[ ] Browse...

(●) Generate a new set

[✓] Solve the generated set

[ ] Save the generated set

Generator: Default

Additional solvers to compare with the ANLDE algorithm:

<none>

Process

Limits:
<limits
information>

Change limits

Figure 23: Process a Set of ANLDE Systems

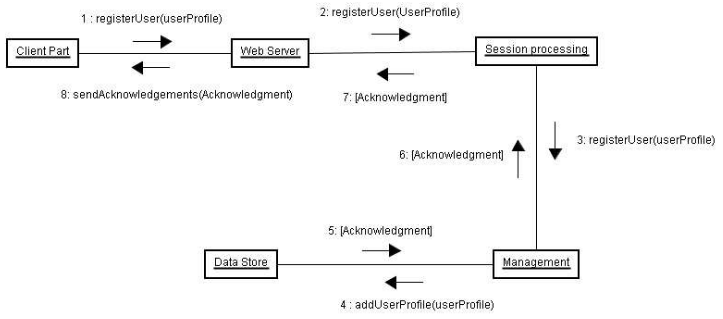

Client Part
Web Server
Session processing
Data Store
Management
1 : registerUser(userProfile)
2: registerUser(UserProfile)
8: sendAcknowledgements(Acknowledgment)
7: [Acknowledgment]
3: registerUser(userProfile)
6: [Acknowledgment]
5: [Acknowledgment]
4 : addUserProfile(userProfile)

Figure 24: Register a user Collaboration Diagram

5: [Acknowledgement] returns the acknowledgment to management;

6: [Acknowledgement] forwards the acknowledgment to session processing;

7: [Acknowledgement] sends the acknowledgment to the web server;

8: sendAcknowledgments(acknowledgment) sends a web form with the acknowledgment to the user;

A user initializes the registeration by selecting the corresponding item in "Log In" form. Then the "Registration" form is displayed in the "Content" part. The user fills the required fields and, perhaps, the optional fields (sect. 3.3.10 User information format) and presses the "Register" button. If there are invalid values the "registration" form will be reloaded in the "Content" part and the error is indicated (sect. 3.3.13 Error message format, 6). If the values are correct, then the acknowledgment message is displayed in the "Content" part (sect. 3.3.12 Acknowledgment format, 4).

This form corresponds to the "Register a user" use case, see Figure 25

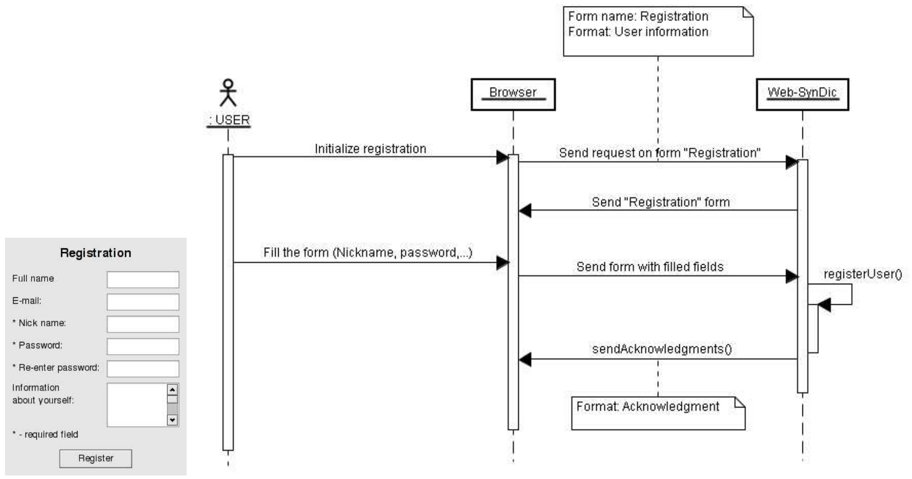


Figure 25: Register a User

4.6 Send user notes

The web system allows a user to send her/his opinion on the solution result or about the web system as a whole. The process of sending notes behaves as follows, see Figure 26.

1.1: sendUserNotes(note) , a user sends a note to the web server;

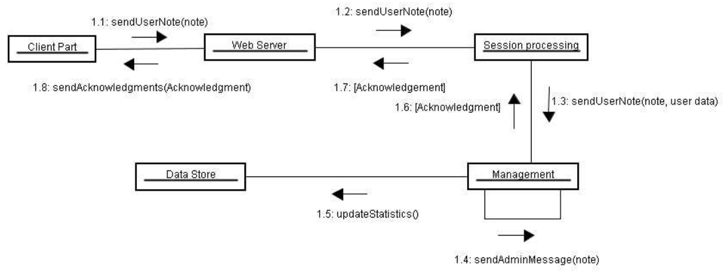


Figure 26: Send user notes Collaboration Diagram

1.2: sendUserNotes(note) forwards the note to session processing;

1.3: sendUserNotes(note) forwards the note to management;

1.4: sendAdminMessage(note, user data) sends a message with the note and extra user's data to the system administrator (by email);

1.5: updateStatistics() updates statistics in data store (a note is sent);

1.6: [Acknowledgment] returns the acknowledgment to session processing;

1.7: [Acknowledgment] returns acknowledgment to the web server;

1.8: sendAcknowledgments(Acknowledgment) the web server produces the web form with the acknowledgment and sends it to the user.

There are two types of notes: general notes (about Web-SynDic as a whole system) and a note on the outcome solution.

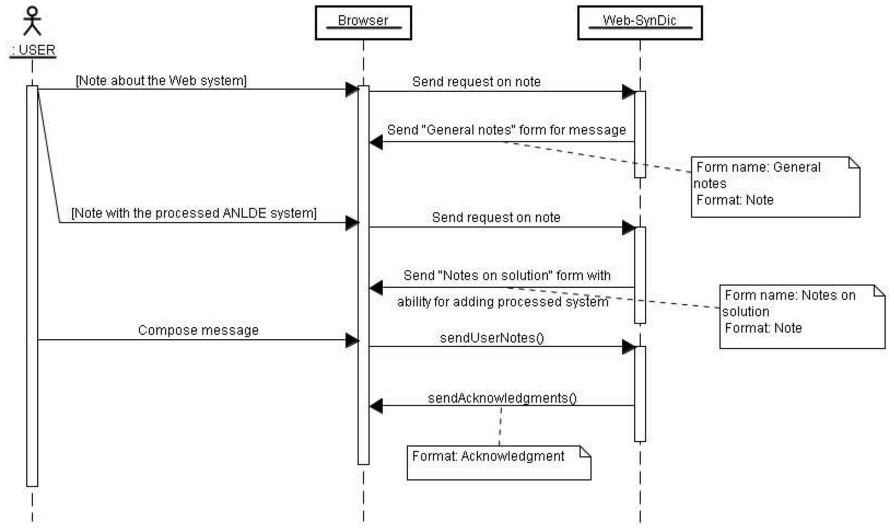
A general note. If a user chooses corresponding item in the main menu, then "General notes" form is displayed in the "Content" part. The user writes a note (sect. 3.3.11 Note format) and presses "Send note" button. If length of the note is more than 4096 symbols, then the error message is displayed in the "Content" part (sect. 3.3.13 Error message format, 3). If not, then the acknowledgment message is displayed in the "Content" part (sect. 3.3.12 Acknowledgment format, 3).

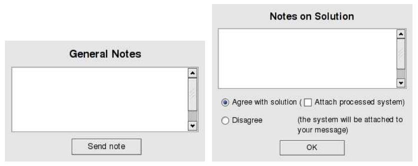
A note on solution. If a user has processed a test ANLDE system or a set of them, the "Notes on solution" form is displayed in the report on solution. The user may choose only one of two types for a note: agree with the solution or disagree with the solution.

In the case of agreement, the user can attach processed ANLDE system by selecting corresponding check box. By default the ANLDE system is not attached.

In the case of disagreement, the processed ANLDE system is always attached to the note. The user enters a note (sect. 3.3.11 Note format) and presses "Send note" button. If length of the note is more than 4096 symbols, then the error message is displayed in the "Content" part (sect. 3.3.13 Error message format, 3). If not, then the acknowledgment message is displayed in the "Content" part (sect. 3.3.12 Acknowledgment format, 3).

This form corresponds to the "Send a note" use case. See Fig. 27.


: USER
Browser
Web-SynDic
[Note about the Web system]
Send request on note
Send "General notes" form for message
Form name: General notes
Format: Note
[Note with the processed ANLDE system]
Send request on note
Send "Notes on solution" form with ability for adding processed system
Form name: Notes on solution
Format: Note
Compose message
sendUserNotes()
sendAcknowledgments()
Format: Acknowledgment


General Notes

Send note

Notes on Solution

[✓ Agree with solution] (☐ Attach processed system)

Disagree (the system will be attached to your message)

OK

Figure 27: Send a Note

4.7 Manage user limits

For management of her/his limits on generation and solution processes, a user may initialize two flows, see Fig. 28.

The first flow is a request for current limits of the user and current default system limits.

1.1: getUserLimits() requests the current limits of the user;

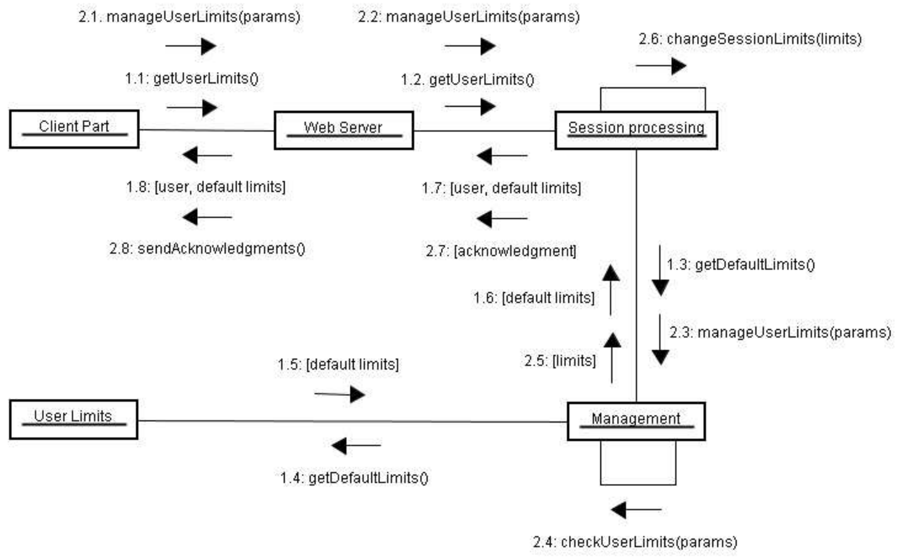

2.1. manageUserLimits(params)
2.2: manageUserLimits(params)
2.6: changeSessionLimits(limits)
1.1: getUserLimits()
1.2. getUserLimits()
Client Part
Web Server
Session processing
1.8: [user, default limits]
1.7: [user, default limits]
2.8: sendAcknowledgments()
2.7: [acknowledgment]
1.3: getDefaultLimits()
1.6: [default limits]
2.3: manageUserLimits(params)
1.5: [default limits]
2.5: [limits]
User Limits
Management
1.4: getDefaultLimits()
2.4: checkUserLimits(params)

Figure 28: Manage User Limits Collaboration Diagram

1.2: getUserLimits() requests the user limits from the current session;

1.3: getDefaultLimits() requests Management subsystem for default limits;

1.4: getDefaultLimits() requests default limits from "User Limits" data store module;

1.5: [default limits] returns the result to Management;

1.6: [default limits] returns the limits (user and default) to Session processing;

1.7: [user, default limits] are given to the web server;

1.8: [user, default limits] are sent in the web form to the browser.

The second one is a request of a user to change her/his user limits.

2.1: manageUserLimits(user limits) sends new values of user limits to the web server;

2.2, 2.3: manageUserLimits(user limits) forwards the user limits to Management subsystem;

2.4: checkUserLimits(user limits) checks the new user limits for correctness and compares with current default limits;

2.5: [user limits] is forwarded for update to “session processing”;

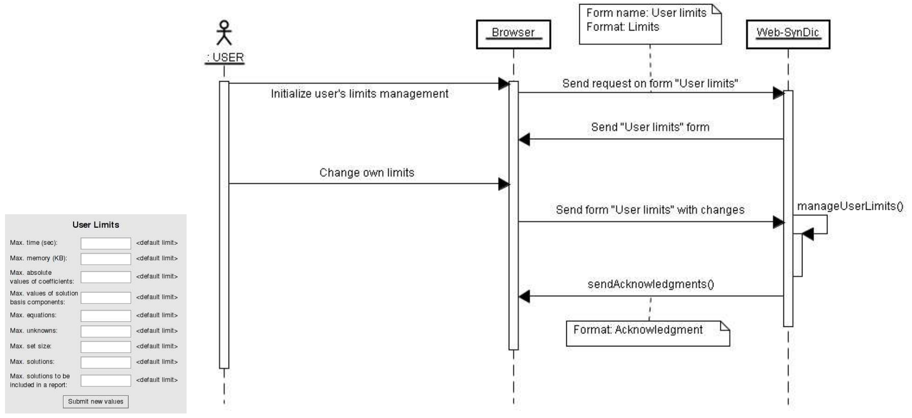
2.6: changeSessionLimits(user limits) changes the user limits for the active session of the user;

2.7: [acknowledgment] sends the acknowledgment about the change status to the web server;

2.8: sendAcknowledgments() sends a web page with the acknowledgment on the change to the browser.

A user initiates ”User limits” form using corresponding item in the user menu. Then the update limits form is displayed in the ”Content” part (sect. 3.3.9 Limits format). The user changes values of the current user limits. If the new values are correct, then acknowledgment message (sect. 3.3.12 Acknowledgment format, 2) is displayed in the ”Content” part. If there is an invalid value, then “User limits” form will be reloaded in the ”Content” part and the error is indicated (sect. 29 Error message format, 2).

This form corresponds to the “Manage user limits” use case, see Fig. 29.



User Limits

Max. time (sec):		<default limit>
Max. memory (KB):		<default limit>
Max. absolute values of coefficients:		<default limit>
Max. values of solution basis components:		<default limit>
Max. equations:		<default limit>
Max. unknowns:		<default limit>
Max. set size:		<default limit>
Max. solutions:		<default limit>
Max. solutions to be included in a report:		<default limit>

Submit new values

Figure 29: Manage User Limits

4.8 Manage default limits

The system administrator may initialize two flows as it is shown in Figure 30.

The first flow is a request for current default limits in Web-SynDic.

1.1: getDefaultLimits() requests for current default limits;

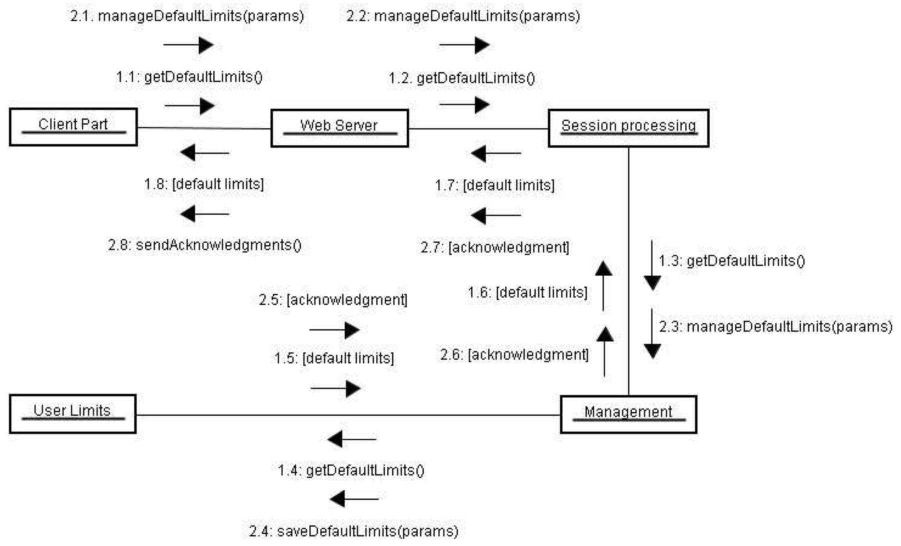


Figure 30: Manage Default Limits Collaboration Diagram

1.2, 1.3: getDefaultLimits() forwards the request to Management subsystem;

1.4: getDefaultLimits() requests a list of current values for default limits in Data store;

1.5: [default limits] are returned to Management;

1.6: [default limits], 1.7: [default limits] are returned/forwarded to the web-server;

1.8: [default limits] is sent in the web form (produced by the web server) to the browser.

The second flow is a request for updating default limits for Web-SynDic.

2.1: manageDefaultLimits() sends changed default limits “params” and requests actual changes inside the Web-system;

2.2, 2.3: manageDefaultLimits(params) forwards limits “params” to “Management”, server subsystem;

2.4: saveDefaultLimits(params) updates default limits, and user profiles in Data store;

2.5: [acknowledgment] returns the acknowledgment about the update;

2.6, 2.7: [acknowledgment] returns/forwards the acknowledgment to the web-server;

2.8: sendAcknowledgments() sends a page with the acknowledgment to the browser.

For updating default limits, a user has to log in as a system administrator. Item for update of default limits is displayed in the user menu and the system administrator may choose it. After that, the form “Default limits” is displayed with current values in the form elements (sect. 3.3.9 Limits format) in the “Content” part. The system administrator changes values of default limits and presses “submit new values” button. If the new values are correct, then the acknowledgment message (sect. 3.3.12 Acknowledgment format, 2) is displayed in the “Content”. Otherwise, “Default limits” form will be reloaded in the “Content” part and the error is indicated (sect. 3.3.13 Error message format, 2).

This form corresponds to the “Manage default limits” use case, see Fig. 31.

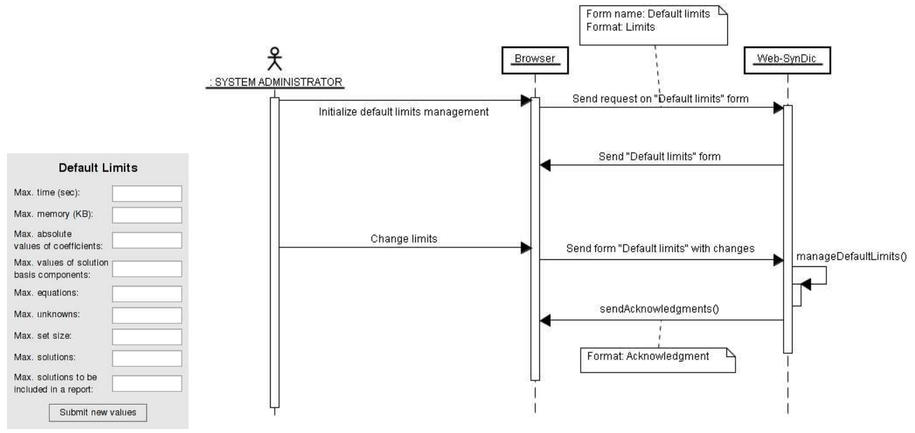

Default Limits
Max. time (sec):
Max. memory (KB):
Max. absolute values of coefficients:
Max. values of solution basis components:
Max. equations:
Max. unknowns:
Max. set size:
Max. solutions:
Max. solutions to be included in a report:
Submit new values
Form name: Default limits
Format: Limits
: SYSTEM ADMINISTRATOR
Browser
Web-SynDic
Initialize default limits management
Send request on "Default limits" form
Send "Default limits" form
Change limits
Send form "Default limits" with changes
manageDefaultLimits()
sendAcknowledgments()
Format: Acknowledgment

Figure 31: Manage Default Limits

4.9 Get statistics

The system administrator may initialize the only flow to request statistics with chosen metrics, see Fig. 32.

1.1: sendRequestStatActivity(params) sends parameters “params” (see sect. 3.2.4 Activity Statistics) and requests to produce the activity statistics report;

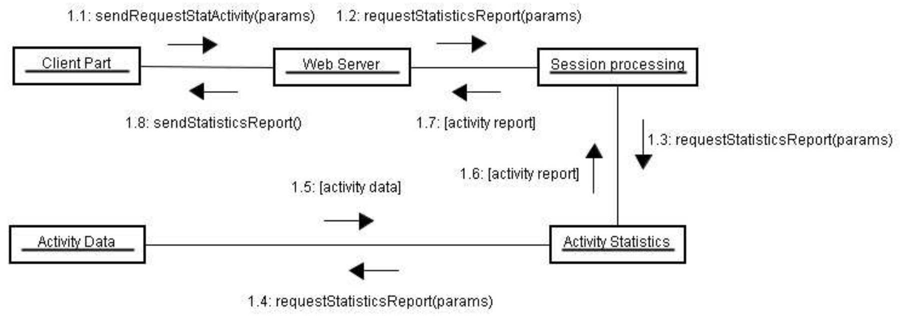

1.1: sendRequestStatActivity(params)
1.2: requestStatisticsReport(params)
Client Part
Web Server
Session processing
1.8: sendStatisticsReport()
1.7: [activity report]
1.3: requestStatisticsReport(params)
1.6: [activity report]
1.5: [activity data]
Activity Data
Activity Statistics
1.4: requestStatisticsReport(params)

Figure 32: Get Statistics Collaboration Diagram

1.2, 1.3: requestStatisticsReport(params) calls “activity statistics”, sends parameters “params” and forwards the request on the statistics report to Activity Statistics subsystem;

1.4: requestStatisticsReport(params) requests statistics data from Activity Data module in Data store;

1.5: [activity data] is returned chosen data to “Activity Statistics”;

1.6: [activity report] is returned evaluated activity statistics;

1.7: [activity report] is passed to the web server;

1.8: sendStatisticsReport() sends the statistics report (as web page) on user activity to the browser.

A user logs in as a system administrator in the ”Log In” form (sect. 3.3.10 User information format). Only in this case “Activity statistics” form is available in the user menu. Then the system administrator starts “Activity statistics” form using this item in the user menu. The “Activity statistics” form (sect. 3.3.6 Activity domain list format, sect. 3.3.7 activity metrics list format) is displayed in the “Content” part. After choosing activity domain and activity metrics, she/he presses “Get report” button and the report (sect. 3.3.8 Statistics report format) is loaded to the “Content” part.

This form corresponds to the ”Get statistics” use case, see Fig. 33.

4.10 Manage users

For user management, the system administrator may initialize two flows, see Figure 34.

The first flow is a request for data of the user with given nickname.

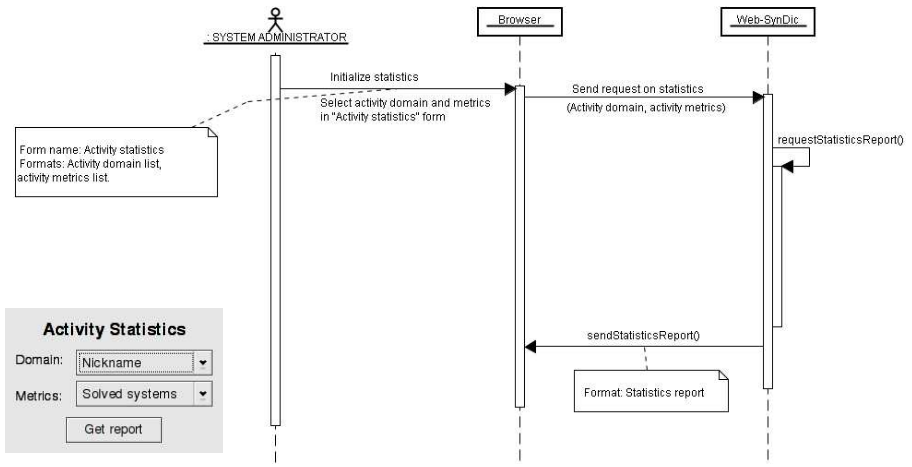

: SYSTEM ADMINISTRATOR
Browser
Web-SynDic
Initialize statistics
Select activity domain and metrics in "Activity statistics" form
Send request on statistics (Activity domain, activity metrics)
requestStatisticsReport()
Form name: Activity statistics Formats: Activity domain list, activity metrics list.
sendStatisticsReport()
Format: Statistics report
Activity Statistics
Domain: Nickname
Metrics: Solved systems
Get report

Figure 33: Get Statistics

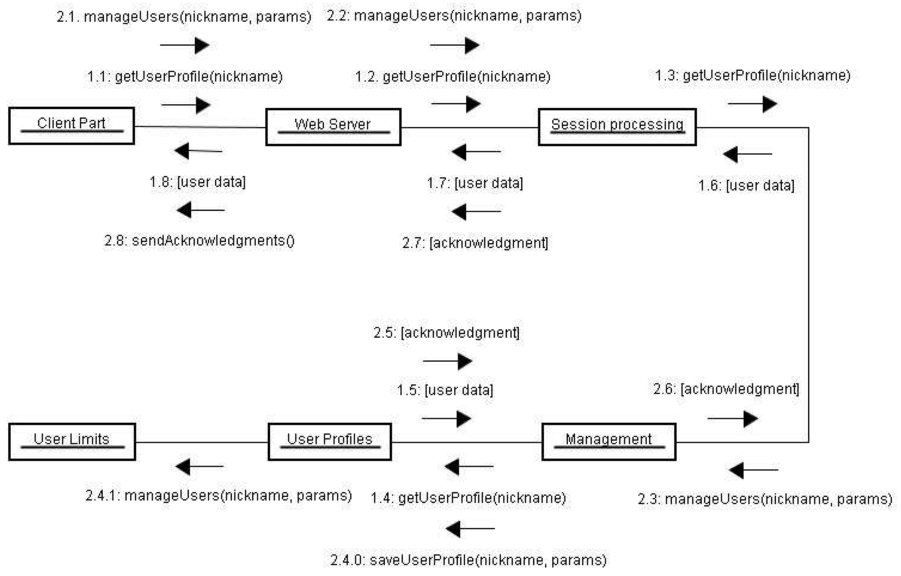

2.1. manageUsers(nickname, params)
2.2: manageUsers(nickname, params)
1.1: getUserProfile(nickname)
1.2. getUserProfile(nickname)
1.3: getUserProfile(nickname)
Client Part
Web Server
Session processing
1.8: [user data]
1.7: [user data]
1.6: [user data]
2.8: sendAcknowledgments()
2.7: [acknowledgment]
2.5: [acknowledgment]
1.5: [user data]
2.6: [acknowledgment]
User Limits
User Profiles
Management
2.4.1: manageUsers(nickname, params)
1.4: getUserProfile(nickname)
2.3: manageUsers(nickname, params)
2.4.0: saveUserProfile(nickname, params)

Figure 34: Manage Users Collaboration Diagram

1.1: getUserProfile(nickname) requests the current data of the registered user with the given "nickname";

1.2, 1.3: getUserProfile(nickname) forwards the request to "Management" subsystem;

1.4: getUserProfile(nickname) requests the data from "User Profile" module of Data store;

1.5: [user data] returns chosen data to "Management";

1.6, 1.7: [user data] is returned/forwarded to the web server;

1.8: [user data] is sent as a web page to the browser.

The second flow is a request for updating the user data.

2.1: manageUsers(nickname, params) sends new data for user with "nickname" as "params";

2.2, 2.3: manageUsers(nickname, params) forwards the user data "params" to "Management" subsystem;

2.4.0: saveUserProfile(nickname, params) makes actual update of user profile, stored in "User Profile" module of Data store;

2.4.1: manageUsers(nickname, params) makes additional changes of user limits (if necessary), stored in "User Limits" module of Data store;

2.5: [acknowledgment] is returned the acknowledgment to "Management" module;

2.6: [acknowledgment], 2.7: [acknowledgment] are returned/forwarded to the web server;

2.8: sendAcknowledgments() sends a web page with the acknowledgment to the browser.

A registered user may change only her/his own information in the "User information" form. The system administrator may change user information for any user.

The system administrator starts this management using corresponding item in the user menu. "Manage users" form is displayed in the "Content" part. The system administrator enters a nickname (sect. 3.3.10 User information format) and presses "Edit" button. If the nickname is correct, then "User information" form is displayed in the "Content" part. Otherwise, "User limits" form is reloaded with the error message (sect. 3.3.13 Error message format, 1) in the "Content" part.

Any other user changes her/his account information in the "User information" form and presses the "Submit new values". If there are invalid values (for example password and re-entering password do not match, etc.), then "User information" form is reloaded in the "Content" part and the error is indicated (sect. 3.3.13 Error message format, 2).

These forms correspond to the "Manage users" use case, see Fig. 35.

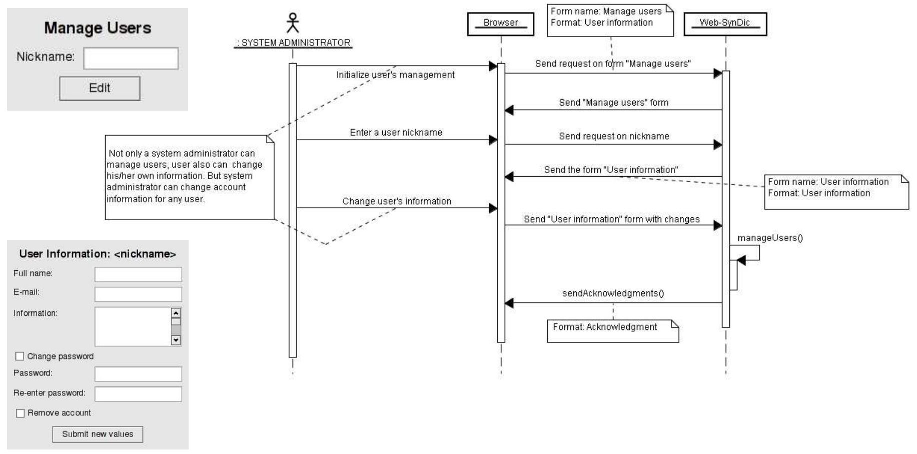

Manage Users
Nickname:
Edit
User Information: <nickname>
Full name:
E-mail:
Information:
Change password
Password:
Re-enter password:
Remove account
Submit new values
: SYSTEM ADMINISTRATOR
Browser
Web-SynDic
Form name: Manage users
Format: User information
Initialize user's management
Send request on form "Manage users"
Send "Manage users" form
Enter a user nickname
Send request on nickname
Send the form "User information"
Form name: User information
Format: User information
Not only a system administrator can manage users, user also can change his/her own information. But system administrator can change account information for any user.
Change user's information
Send "User information" form with changes
manageUsers()
sendAcknowledgments()
Format: Acknowledgment

Figure 35: Manage Users

5 Subsystems

The color style for subsystem design models is shown in Figure 36.

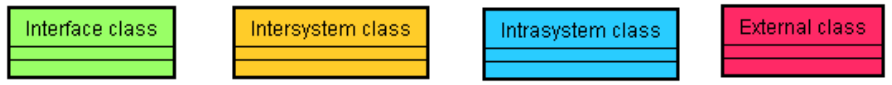


Figure 36: Color style for subsystem design models

5.1 Web Server

The Web Server subsystem is responsible for data representation in interaction with users. The architecture of the web server is shown in Fig. 37.

5.1.1 JSP Files

The user interface is implemented using Java Server Pages technology.

main.jsp the only page directly accessible from the user's web browser, implements page layout (sect. 3.1.3), responses of all other JSP's are included in the content part, except responses of the login.jsp are included in the login part;

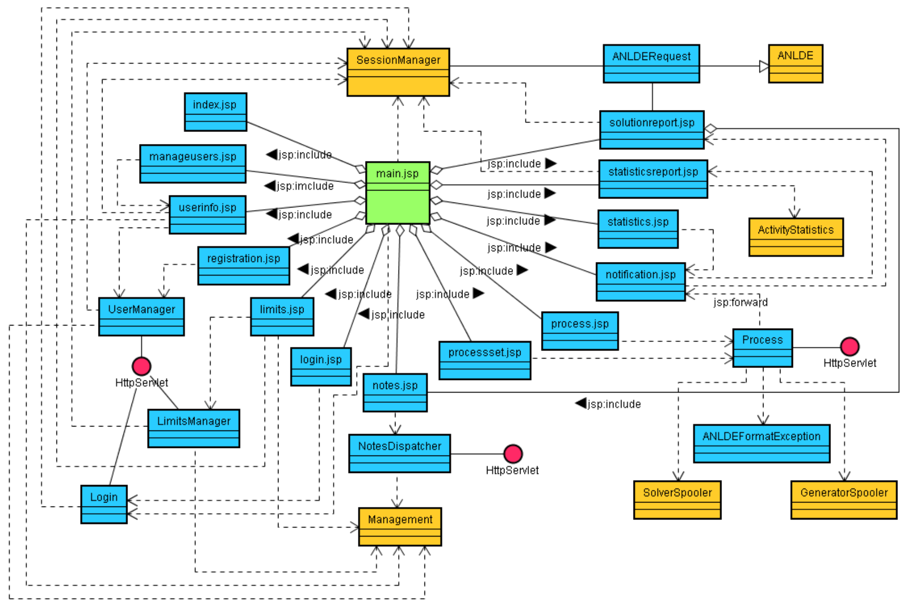


Figure 37: Web Server Architecture

index.jsp displays start page;

manageusers.jsp displays Manage Users form (sect. 3.2.8);

userinfo.jsp displays User Information form (sect. 3.2.9);

registration.jsp displays Registration form (sect. 3.2.6);

limits.jsp displays User Limits (sect. 3.2.7) or Default Limits (sect. 3.2.5) forms;

login.jsp displays Login form (sect. 3.2.3);

notes.jsp displays General Notes (sect. 3.2.10) of Notes on Solution sect. 3.2.11) forms;

process.jsp displays Process an ANLDE System form (sect. 3.2.1);

processset.jsp displays Process a Set of ANLDE Systems form (sect. 3.2.2);

notification.jsp displays notification message (sect. 3.3.12 Acknowledgment Format, sect. 3.3.13 Error Message Format, sect. 3.3.14 Process Message Format);

statistics.jsp displays Statistics form (sect. 3.2.4);

statisticsreport.jsp displays a statistics report (sect. 3.3.8 Statistics Report Format);

solutionreport.jsp displays solution report (sect. 3.3.3 ANLDE System Solution Format, sect. 3.3.4 ANLDE System Set Solution Format).

5.1.2 Servlets

Servlets are used to handle various user requests. Servlets classes extend javax.servlet.http.HttpServlet and override doPost() method to handle POST requests.

UserManager receives information from User Information and Registration forms, invokes the Management subsystem to add or change user accounts;

LimitsManager receives information from User Limits and Default Limits forms;

Login receives information from Login form and logout request;

NotesDispatcher receives information from General Notes and Notes on Solution forms;

Process receives information from Process an ANLDE System and Process a Set of ANLDE Systems forms, translates data in ANLDE System Format (sect. 3.3.1) to the internal format (class ANLDE).
Additional methods:

- `private ANLDERequest parseANLDE(String source, boolean set, Limits limits) throws ANLDEFormatException;`
  translates ANLDE system (if `set` value is `false`) or ANLDE system set (if `set` value is `true`) from the ANLDE System Format (sect. 3.3.1) or ANLDE System Set Format (sect. 3.3.2) to the internal format (instance of the ANLDERequest class). ANLDEFormatException is thrown in case of parsing error or when the ANLDE system does not conform to the limits.

5.1.3 Exceptions

ANLDEFormatException thrown by Process.parseANLDE() method when it fails to parse ANLDE system (set).

5.1.4 Other Classes

ANLDERequest extends ANLDE, used to receive notifications on solving and generating ANLDE systems and system sets from the Algorithm Server (see sect. 5.3) subsystem.

Fields:

- `private SolverOutcome solution;`
- `private processing = false;`
  true — ANLDE System (Set) is currently in process of generating or solving, false — ANLDE System (Set) is ready to be processed.

Following abstract methods are implemented:

- `public void solved(SolverOutcome);`
  called by Algorithm Server, notifies waiting thread of report.jsp, stores `SolutionOutcome` in `solution` variable;
- `public void generated();`
  called by Algorithm Server, notifies the Process servlet about finishing ANLDE generation;
- `public void error(String);`
  called by the Algorithm Server to notify waiting report.jsp or Process servlet thread about ANLDE generation or solution errors.

5.2 Session processing

Session processing entity includes one class **SessionManager**. At the implementation phase, this class will be coded as a part of web server.

5.2.1 Class SessionManager

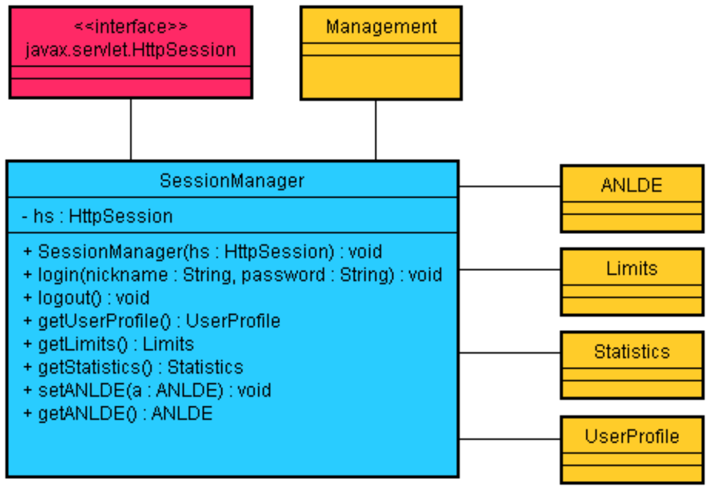
Class SessionManager is used to store user profile, limits, statistics and last solved ANLDE system information. It is shown in Fig. 38.

Fields:

private HttpSession hp; instance of HttpSession object.

Constructors:

SessionManager(HttpSession hp); creates a new SessionManager object corresponding to existing http session.



<<interface>>
javax.servlet.HttpSession
Management
SessionManager
- hs : HttpSession
+ SessionManager(hs : HttpSession) : void
+ login(nickname : String, password : String) : void
+ logout() : void
+ getUserProfile() : UserProfile
+ getLimits() : Limits
+ getStatistics() : Statistics
+ setANLDE(a : ANLDE) : void
+ getANLDE() : ANLDE
ANLDE
Limits
Statistics
UserProfile

Figure 38: Class SessionManager

Methods:

public void login(String nickname, String password); binds attributes of registered user to the existing session:

1. tries to get UserProfile object corresponding to *nickname* and *password*;
2. creates new instance of Statistics object;
3. binds newly created objects to the session. Old objects are unbound automatically and notified using HttpSessionBindingListener;
4. unbinds ANLDE object.

public void logout(); is similar to login function except that default user's profile is used instead of registered user's profile.

public UserProfile getUserProfile(); returns user profile.

public Limits getLimits(); return user limits.

public Statistics getStatistics(); returns statistics.

public void setANLDE(ANLDE a); binds new ANLDE object to the session.

public ANLDE getANLDE (); returns ANLDE object.

5.3 Algorithm server

The composition model of classes for the algorithm server is shown in Fig. 39.

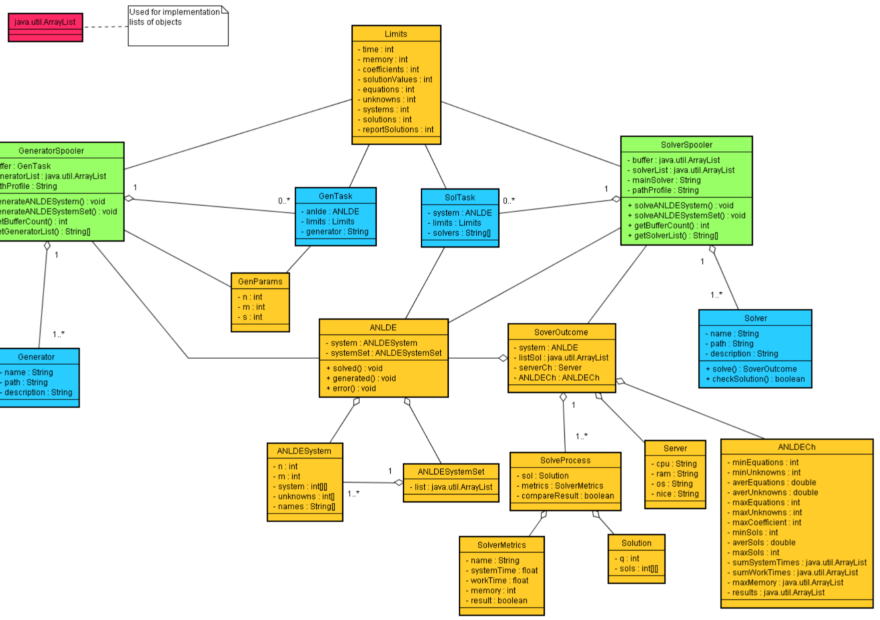


Figure 39: Algorithm server architecture

The algorithm server has two interface classes for generating and solving ANLDE systems. The generating and solving processes can be implemented as parallel or sequential processes.

In parallel approach, all systems are solved simultaneously. It means that for each solving process the web system starts separate copy of a solver. Each started copy takes away resources from other processes. As a result all ANLDE systems are solved slowly (process degradation).

For the sequential approach, all systems are solved in turn. Each solving process has maximum resources but solving time for each ANLDE system increase by summary time of ANLDE systems that were pushed in the stack earlier.

After discussion between team members we choose the sequantial approach for solving and generating ANLDE systems, because of user requirements **AP2: Web server overload** and **AP1: Concurrent user sessions**. These requirements define that the web system must not overload a base server more than 75% of the total server workload and web system must serve up to 5 users without significant reduction of the server perfomance.

After ending, the generating or solving process calls method "solved" or "generated" which sends signal to the web system and after getting this signal, the web system can work with the generated ANLDE systems or the solver outcome. This methods redefines in heritable classes of the web server.

External algorithms will include java classes to work with each of them. Each java class is a child of classes "Solver" or "Generator". When we work with external algorithms, we use methods and fields defined in classes "Solver" and "Generator". It allows to include new external algorithms without recompilation of the whole web system.

For simplicity, in generation process we use user limits as generating parameters like number of systems in set, number of equations and unknowns, etc. Generator build ANLDE systems with exact dimensions. And we suppose that generator work time is small and does not exceed time limit; the web system does not limit work time for generation process.

5.3.1 Class GeneratorSpooler

Description: class to generate ANLDE systems. It includes interface functions. In order to get maximum precision of solution process metrics, it is better to serially start generating process. Therefore, it has to use a buffer (queue) of generating tasks.

For generation process, one needs to know generator location and identificator. Therefore, when generator spooler starts, it loads the generator profile.

Fields:

private java.util.List buffer; a buffer of GenTask objects
private java.util.List generatorList; a list of Generator objects
private String pathProfile; path and name of a generator profile

Constructors:

public void GeneratorSpooler(String path); loading generators and starting generating process. Generator uses buffer for generating ANLDE systems

Methods:

public void generateANLDESystem(ANLDE, Limits, String); — generate ANLDE system using generator in String
public void generateANLDESystemSet(ANLDE, Limits, String); — generate a set of ANLDE systems using generator in String
public int getBufferCount(); — get count of tasks

public String[ ] getGeneratorList(); — gets list of generator names

5.3.2 Class SolverSpooler

Description: class for solve ANLDE systems. Includes interface functions. In order to get maximum precision of solution process metrics, it is necessary to serially start the solving process. Therefore it is necessary to work with a buffer of solving tasks. After finishing the work of external algorithms, SolverSpooler sends signal with SolverOutcome.

For solving process we need to know solver location and identificator. Therefore, when solver spooler starts, it loads the solver profile.

Fields:

private java.util.List buffer; — buffer of SolTask objects
private java.util.List solverList; — list of Solver objects
private Solver mainSolver; — main solver like anlde
private String pathProfile; — file with solver profile

Constructors:

public void SolverSpooler(String path); — loading solvers and starting solving process. Solver uses buffer for solving ANLDE systems

Methods:

public void solveANLDE(ANLDE, Limits, String[ ]); — solve ANLDE system or a set of ANLDE systems using solvers in String[ ]
public int getBufferCount(); — get count of tasks
public String[ ] getSolverList(); — gets list of solver names

5.3.3 Class Limits

Description: class for storing user limits and default limits.

Fields:

private int time; — maximum value of time
private int memory; — maximum value of memory
private int coefficients; — maximum value of coefficients
private int solutionValues; — maximum value of solution coefficients

private int equations; — maximum number of equations
private int unknowns; — maximum number of unknowns
private int systems; — maximum number of systems in set
private int solutions; — maximum number of solutions
private int reportSolutions; — maximum number of solutions in report

Constructors:
public Limits(); — storing new limits.

Methods:
public int getTime(); — gets maximum value of time
public int getMemory(); — gets maximum value of memory
public int getCoefficients(); — gets maximum value of coefficients
public int getSolutionValues(); — gets maximum value of solution coefficients
public int getEquations(); — gets maximum number of equations
public int getUnknowns(); — gets maximum number of unknowns
public int getSystems(); — gets maximum number of systems in set
public int getSolutions(); — gets maximum number of solutions
public int getReportSolution(); — gets maximum number of solutions in report
public void setTime(int); — sets maximum value of time
public void setMemory(int); — sets maximum value of memory
public void setCoefficients(int); — sets maximum value of coefficients
public void setSolutionValues(int); — sets maximum value of solution coefficients
public void setEquations(int); — sets maximum number of equations
public void setUnknowns(int); — sets maximum number of unknowns
public void setSystems(int); — sets maximum number of systems in set
public void setSolutions(int); — sets maximum number of solutions
public void setReportSolution(int); — sets maximum number of solutions in report

5.3.4 Class ANLDE

Description: class for storing a test ANLDE system or a set of ANLDE system.

Fields:
protected ANLDESystem system; — ANLDE system or null if a set of ANLDE systems was stored
protected ANLDESystemSet systemSet; — a set of ANLDE systems or null if ANLDE

system was stored

Constructors:
public ANLDE(ANLDESystem); — constructor for storing ANLDE system
public ANLDE(); — constructor nondefined ANLDE system
public ANLDE(ANLDESystemSet); — constructor for storing a set of ANLDE systems

Methods:
public abstract void solved(SolverOutcome); — sends solved signal and solver outcome. This method implemented in extended classes.
public abstract void generated(); — sends generated signal. This method implemented in extended classes.
public abstract void error(String); — sends error signal. This method implemented in extended classes.
public void setSystem(ANLDESystem); — sets ANLDE system and a set of ANLDE systems set to null
public void setSystemSet(ANLDESystemSet); — sets a set of ANLDE systems and a ANLDE system set to null
public ANLDESystem getSystem(); — gets ANLDE system
public ANLDESystemSet getSystemSet(); — gets a set of ANLDE systems

5.3.5 Class ANLDESystem

Description: class for storing ANLDE system.

Fields:
private int n; — number of equations
private int m; — number of unknowns
private int[n][m] system; — matrix of coefficients
private int[m] unknowns; — matrix of left part of ANLDE system
private String[m] names; — list of unknowns names

Constructors:
ANLDESystem(); — storing ANLDE system

Methods:

public int getN(); — gets number of equations
public int getM(); — gets number of unknowns
public int[ ][ ] getSystem(); — gets matrix of coefficients
public int[ ] getUnknowns(); — gets left part of ANLDE system
public String[ ] getNames(); — gets list of unknowns names
public void setN(int); — sets number of equations
public void setM(int); — sets number of unknowns
public void setSystem(int[ ][ ]); — sets matrix of coefficients
public void setUnknowns(int[ ]); — sets left part of ANLDE system
public void setNames(String[ ]); — sets list of unknowns names

5.3.6 Class ANLDESystemSet

Description: class for storing a set of ANLDE systems.

Fields:

private java.util.List list; — list of ANLDE objects

Constructors:

ANLDESystemSet(); — storing a set of ANLDE systems

Methods:

public int getS(); — gets number of systems
public java.util.List getList(); — gets set of ANLDE systems
public void setList(java.util.List); — sets set of ANLDE systems

5.3.7 Class SolverOutcome

Description: class for storing solver outcome.

Fields:

private ANLDE system; — ANLDE system or a set of ANLDE system
private java.util.List listSol; — list of SolverProcess objects
private Server serverCh; — server characteristics
private ANLDECh anldeCh; — characteristics of ANLDE system

Constructors:

public SolverOutcome(); — storing solver outcome

Methods:

public ANLDE getANLDE(); — gets ANLDE system
public java.util.List getListSol(); — gets solutions
public Server getServer(); — gets server characteristics
public void setANLDE(ANLDE); — sets ANLDE system
public ANLDECh getANLDECh(); — gets ANLDE system characteristics
public void setANLDECh(ANLDECh); — sets ANLDE system characteristics
public void setListSol(java.util.List); — sets solutions
public void setServer(Server); — sets server characteristics

5.3.8 Class ANLDECh

Description: class for storing ANLDE system characteristics.

Fields:

private int minEquations; — minimal number of equations
private int minUnknowns; — minimal number of unknowns
private double averEquations; — average number of equations
private double averUnknowns; — average number of unknowns
private int maxEquations; — maximum number of equations
private int maxUnknowns; — maximum number of unknowns
private int maxCoefficient; — maximum coefficient in ANLDE systems
private int minSols; — minimum number of solutions
private double averSols; — average number of solutions
private int maxSols; — maximum number of solutions
private java.util.List sumSystemTimes; — list of summary system time periods for each solvers
private java.util.List sumWorkTimes; — list of summary work time periods for each solvers
private java.util.List maxMemory; — list of maximum memories for each solvers
private java.util.List results; — list of results

Constructors:

public ANLDECh(); — storing solver outcome

Methods:

public int getMinEquations(); — gets minimal number of equations
public int getMinUnknowns(); — gets minimal number of unknowns
public double getAverEquations(); — gets average number of equations
public double getAverUnknowns(); — gets average number of unknowns
public int getMaxEquations(); — gets maximum number of equations
public int getMaxUnknowns(); — gets maximum number of unknowns
public int getMaxCoefficient(); — gets maximum coefficient in ANLDE systems
public int getMinSols(); — gets minimum number of solutions
public double getAverSols(); — gets average number of solutions
public int getMaxSols(); — gets maximum number of solutions
public java.util.List getSumSystemTimes(); — gets summary system time periods
public java.util.List getSumWorkTimes(); — gets summary work time periods
public java.util.List getMaxMemory(); — gets summary memories
public java.util.List getResults(); — gets results
public void setMinEquations(int); — sets minimal number of equations
public void setMinUnknowns(int); — sets minimal number of unknowns
public void setAverEquations(double); — sets average number of equations
public void setAverUnknowns(double); — sets average number of unknowns
public void setMaxEquations(int); — sets maximum number of equations
public void setMaxUnknowns(int); — sets maximum number of unknowns
public void setMaxCoefficient(int); — sets maximum coefficient in ANLDE systems
public void setMinSols(int); — sets minimum number of solutions
public void setAverSols(double); — sets average number of solutions
public void setMaxSols(int); — sets maximum number of solutions
public void setSumSystemTimes(java.util.List); — sets summary system time periods
public void setSumWorkTimes(java.util.List); — sets summary work time periods
public void setMaxMemory(java.util.List); — sets summary memories
public void setResults(java.util.List); — sets results

5.3.9 Class Solution

Description: class for storing a solution.

Fields:

private int q; — number of solutions
private int[q][m] sols; — matrix of solutions

Constructors:

public Solution(); — storing solution

Methods:

public void setQ(int); — sets number of solutions
public void setSols(int[ ][ ]); — sets matrix of solutions
public int getQ(); — gets number of solutions
public int[ ][ ] getSols(); — gets matrix of solutions

5.3.10 Class SolverMetrics

Description: class for storing metrics of solution process.

Fields:

private String name; — name of solver
private float systemTime; — system time used for solving process
private float workTime; — solve work time
private int memory; — memory usage
private boolean result; — solving process result (true if solved)

Constructors:

public SolverMetrics(); — creates storing for solver metrics

Methods:

public void setName(String); — sets solver name
public void setSystemTime(float); — sets system time
public void setWorkTime(float); — sets work time
public void setMemory(int); — sets memory usage
public void setResult(boolean); — sets result
public String getName(); — gets solver name
public float getSystemTime(); — gets system time
public float getWorkTime(); — gets work time
public int getMemory(); — gets memory usage
public boolean getResult(); — gets result

5.3.11 Class SolverProcess

Description: class for storing characteristics of solving process, and solution.

Fields:
private Solution sol; — solution
private SolverMetrics metrics; — solver metrics
private boolean compareResult — result of comparing solution with anlde solution

Constructors:
public SolverProcess(); — creates storing for solver process

Methods:
public void setSol(Solution); — sets solution
public void setMetrics(SolverMetrics) — sets metrics
public void setCompareResult(boolean) — sets comparison result
public Solution getSol(); — gets solution
public SolverMetrics getMetrics() — gets metrics
public boolean getCompareResult() — gets comparison result

5.3.12 Class Server

Description: class for storing server characteristics.

Fields:
private String cpu; — information about CPU
private String ram; — information about RAM
private String os; — information about OS
private String nice; — information about priority

Constructors:
public Server(); — creates storing for server characteristics

Methods:
public void setCPU(String); — sets CPU
public void setRAM(String); — sets RAM

public void setOS(String); — sets OS
public void setNice(String); — sets priority
public String getCPU(); — gets CPU
public String getRAM(); — gets RAM
public String getOS(); — gets OS
public String getNice(); — gets priority

5.3.13 Class Generator

Description: class for storing generator description, like location, conversion functions, etc.

Fields:
private String name; — generator's name
private String path; — generator's location
private String description; — generator's description

Constructors:
public Generator(); — creates storing for generator

Methods:
public void setName(String); — sets name
public void setPath(String); — sets path
public void setDescription(String); — sets description
public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public void generate(ANLDE, Limits); — generates ANLDE system
public void generateSet(ANLDE, Limits); — generates a set of ANLDE systems

5.3.14 Class Solver

Description: class for storing solver description, like location, conversion functions, etc.

Fields:
private String name; — solver's name
private String path; — solver's location

private String description; — solver's description

Constructors:

public Solver(); — creates storing for generator

Methods:

public void setName(String); — sets name
public void setPath(String); — sets path
public void setDescription(String); — sets description
public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public SolverOutcome solve(ANLDE, Limits); — converts and solves ANLDE system or a set of ANLDE systems
public boolean checkSolution(Solution mainSol, Solution currentSol); — compares current solution with main solver solution and return result.

5.3.15 Class GenTask

Description: class for storing generating tasks.

Fields:

private ANLDE anlde; — ANLDE system which will be generated
private Limits limits; — user limits
private String generator; — explotable generator

Constructors:

public GenTask(); — creates task

Methods:

public void setLimits(Limits); — sets user limits
public void setGenerator(String); — sets explotable generator
public void setANLDE(ANLDE); — sets ANLDE system
public Limits getLimits(void); — gets user limits
public String getGenerators(void); — gets explotable generator
public ANLDE getANLDE(void); — gets ANLDE system

5.3.16 Class SolTask

Description: class for storing solving tasks.

Fields:

private ANLDE system; — ANLDE system for solving
private Limits limits; — user limits
private String[ ] solvers; — explotable solvers

Constructors:

public GenTask(); — creates task

Methods:

public void setANLDE(ANLDE); — sets ANLDE system
public void setLimits(Limits); — sets user limits
public void setSolvers(String[ ]); — sets explotable solvers
public ANLDE getANLDE(void); — gets ANLDE system
public Limits getLimits(void); — gets user limits
public String[ ] getSolvers(void); — gets explotable solvers

5.4 Solver classes

5.4.1 Class Solver-anlde extends Solver

Description: class for storing anlde solver description, like location, conversion functions, etc.

Fields:

private String name="anlde"; — solver's name
private String path="./anlde"; — solver's location
private String description="anlde solver. Autor: D. G. Korzun. Find Hilbert basis"; — solver's description

Constructors:

public Solver-anlde(); — creates storing for solver

Methods:

public String getName(); — gets name

public String getPath(); — gets path
public String getDescription(); — gets description
public SolverOutcome solve(ANLDE,Limits); — converts ANLDE system or a set of ANLDE systems into anlde format and solves it. Solver returns Hilbert basis.
public boolean checkSolution(Solution mainSol, Solution currentSol); — this method always returns true, because we compare equal solutions.

5.4.2 Class Solver-slopes extends Solver

Description: class for storing slopes solver description, like location, conversion functions, etc.

Fields:
private String name="slopes"; — solver's name
private String path="./slopessys"; — solver's location
private String description="slopes solver. Autors: Ana Paula Tomas, Miguel Filgueiras. Find Hilbert basis"; — solver's description

Constructors:
public Solver-slopes(); — creates storing for solver

Methods:
public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public SolverOutcome solve(ANLDE,Limits); — converts ANLDE system or a set of ANLDE systems into slopes format and solves it. Solver returns Hilbert basis.
public boolean checkSolution(Solution mainSol, Solution currentSol); — this method compare Hilbert basises elementwise. But vectors in Hilbert basises may be in different order. Therefore required to find fit vectors before compare it.

5.4.3 Class Solver-lp_solver extends Solver

Description: class for storing lp_solver solver description, like location, conversion functions, etc.

Fields:
private String name="lp_solver"; — solver's name
private String path="./lp_solver"; — solver's location

private String description="lp_solver solver. Find particular solution."; — solver's description

Constructors:
public Solver-lp_solver(); — creates storing for solver

Methods:
public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public SolverOutcome solve(ANLDE,Limits); — converts ANLDE system or a set of ANLDE systems into lp_solver format (make optimization problem) and solves it. Solver returns particular solution.
public boolean checkSolution(Solution mainSol, Solution currentSol); — this method find particular solution in Hilbert basis.

5.4.4 Class Solver-GLPK extends Solver

Description: class for storing GLPK solver description, like location, conversion functions, etc.

Fields:
private String name="GLPK"; — solver's name
private String path="./glpk"; — solver's location
private String description="GLPK solver. Find particular solution."; — solver's description

Constructors:
public Solver-GLPK(); — creates storing for solver

Methods:
public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public SolverOutcome solve(ANLDE,Limits); — converts ANLDE system or a set of ANLDE systems into GLPK format (make optimization problem) and solves it. Solver returns particular solution.
public boolean checkSolution(Solution mainSol, Solution currentSol); — this method find particular solution in Hilbert basis.

5.5 Generator classes

5.5.1 Class Generator-Gauss

Description: class for storing gauss generator description, like location, conversion functions, etc.

Fields:

private String name="gauss"; — generator's name
private String path="./gaussgen"; — generator's location
private String description="ANLDE generator. Generate simple ANLDE system and corresponding Hilbert basis"; — generator's description

Constructors:

public Generator-Gauss(); — creates storing for generator

Methods:

public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public ANLDE generate(GenParams,Limits); — converts parameters and limits into internal format and generates ANLDE system or a set of ANLDE systems

5.5.2 Class Generator-Gordano

Description: class for storing gordano generator description, like location, conversion functions, etc.

Fields:

private String name="gordano"; — generator's name
private String path="./gordanogen"; — generator's location
private String description="ANLDE generator. Generate ANLDE system and corresponding Hilbert basis"; — generator's description

Constructors:

public Generator-Gordano(); — creates storing for generator

Methods:

public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public ANLDE generate(GenParams,Limits); — converts parameters and limits into internal format and generates ANLDE system or a set of ANLDE systems

5.5.3 Class Generator-ExpandedGordano

Description: class for storing expanded gordano generator description, like location, conversion functions, etc.

Fields:

private String name="expandedGordano"; — generator's name
private String path="./exgordanogen"; — generator's location
private String description="ANLDE generator. Generate ANLDE system and corresponding Hilbert basis"; — generator's description

Constructors:

public Generator-ExpandedGordano(); — creates storing for generator

Methods:

public String getName(); — gets name
public String getPath(); — gets path
public String getDescription(); — gets description
public ANLDE generate(GenParams,Limits); — converts parameters and limits into internal format and generates ANLDE system or a set of ANLDE systems

5.6 Data store

The class diagram used in data store is shown in Fig. 40.

The data store has three interface classes for user information, default limits information and statistics information from the data store. It realizes writing and reading data from the data store from the hard disk. Information about users, default limits and statistics is storing in log files (Statistics log file, default limits log file, user profile log files). First string in each files is reserved for a format version number.

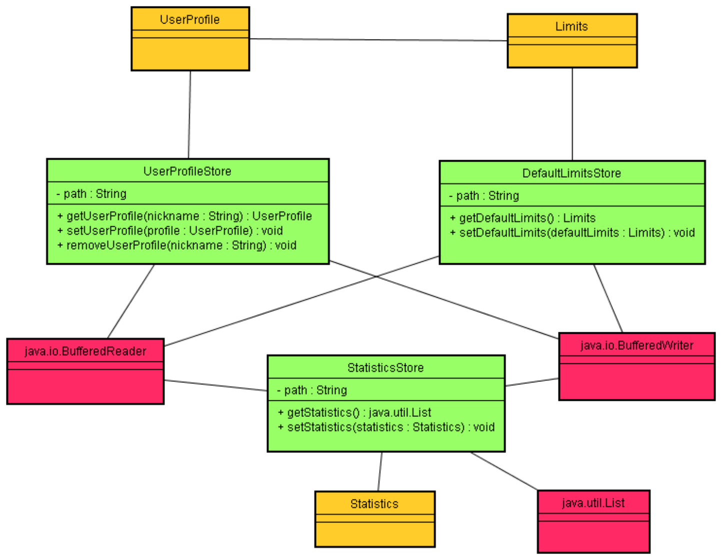

UserProfile
Limits
UserProfileStore
- path : String
+ getUserProfile(nickname : String) : UserProfile
+ setUserProfile(profile : UserProfile) : void
+ removeUserProfile(nickname : String) : void
DefaultLimitsStore
- path : String
+ getDefaultLimits() : Limits
+ setDefaultLimits(defaultLimits : Limits) : void
java.io.BufferedReader
java.io.BufferedWriter
StatisticsStore
- path : String
+ getStatistics() : java.util.List
+ setStatistics(statistics : Statistics) : void
Statistics
java.util.List

Figure 40: Data store architecture

5.6.1 User profile log file

Description: This section describes user profile file format. The user nickname and the name of corresponding user profile file are the same.

Format: A user profile log file is a set of strings. First string after format version number is a user's password. Next string is email. The next nine strings are the user limits (Limits format). And all remaining strings are information about a user.

Example:

#version: 1
qwerty
qwerty@localhost.localdomain
100
3000
100
100
20
20

20
1000
50
I live in Petrozavodsk

5.6.2 Class UserProfileStore

Description: Class for writing, reading and/or deleting user profiles.

Fields:

private String path; — path to the user profiles store.

Constructors :

public void UserProfileStore(String path); — initializes object with path to the user profile store.

Methods:

public UserProfile getUserProfile(String nickname);— gets user profile from the hard disk with corresponding nickname
public void setUserProfile(UserProfile profile); — writes user profile on the hard disk
public removeUserProfile(String nickname); — deletes user profile

5.6.3 Class DefaultLimitsStore

Description: Class for writing and reading default limits.

Fields:

private String path; — path to the default limits file

Constructors :

public void DefaultLimitsStore(String path); — initializes object with path to the default limits file

Methods:

public Limits getDefaultLimits(); — gets default limits from the data store public void setDefaultLimits(Limits defaultLimits); — writes default limits on the data store

5.6.4 Default limits file

Description: This section describes default limits file format.

Format: A default limits file consists of a set of strings. Each string is a one default limit (Limits format).

Example:

```
#version: 1
100
3000
100
100
20
20
20
20
1000
50
```

5.6.5 Statistics log file

Description: This section describes statistics file format. Each file contents monthly activity statistics.

Format: A statistics log file is a set of strings. Each string is a statistics for one user. String has a set of fields separated by white spaces. Format described in Statistics class. Fields in the string are:

1. user nickname
2. user IP address
3. number of generated systems
4. number of input systems
5. number of solved systems
6. number of acknowledgment systems
7. number of systems with solutions, which are discrepances solving
8. summary system time(sec)
9. summary user work time(sec)

10. summary used memory(Kb)
11. start session time mark(ms)
12. end session time mark(ms)

Example:

```
#version: 1
guest 123.123.123.123 15 13 14 9 1 0.1 9.34 2192 1068376393 1068377093
```

5.6.6 Class StatisticsStore

Description: Class for writing and reading activity statistics. Statistics is collected in a buffer. Records of statistics are writed on the data store per month. Each file of statistics is a monthly statistics. Activity statistics is accessible via web-interface only for a current month. Statistics for other previous months is stored on the hard disk of the server.

Fields:

`private String path;` — path to the statistics files

Constructors:

`public void statisticsStore(path);` — initialize object with path to the statistics files

Methods:

`public java.util.List getStatistics();` — get statistics from the data store, returns list of Statistics objects

`public void setStatistics(Statistics statistics);` — write monthly statistics to the buffer and data store from the buffer.

5.7 Management

The class diagram used in management subsystem is shown in Fig. 41.

Management has one interface class. It realizes different management functions.

5.7.1 Class UserProfile

Description: Class for storing user profile.

Fields:

`private String nickname;` — user's nickname
`private String fullname;` — user's fullname
`private String password;` — user's password

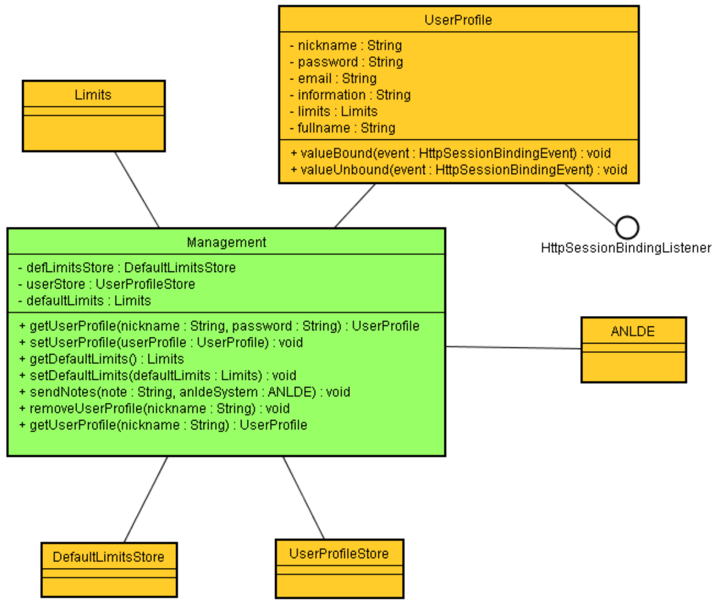

UserProfile
- nickname : String
- password : String
- email : String
- information : String
- limits : Limits
- fullname : String
+ valueBound(event : HttpSessionBindingEvent) : void
+ valueUnbound(event : HttpSessionBindingEvent) : void
Limits
HttpSessionBindingListener
Management
- defLimitsStore : DefaultLimitsStore
- userStore : UserProfileStore
- defaultLimits : Limits
+ getUserProfile(nickname : String, password : String) : UserProfile
+ setUserProfile(userProfile : UserProfile) : void
+ getDefaultLimits() : Limits
+ setDefaultLimits(defaultLimits : Limits) : void
+ sendNotes(note : String, anldeSystem : ANLDE) : void
+ removeUserProfile(nickname : String) : void
+ getUserProfile(nickname : String) : UserProfile
ANLDE
DefaultLimitsStore
UserProfileStore

Figure 41: Management architecture

private String email; — user's email
private String information; — user's information
private Limits limits; — user's limits

Methods:

public String getNickname(); — gets user's nickname
public void setNickname(String nickname); — sets user's nickname
public String getFullname(); — gets user's full name
public void setFullname(String nickname); — sets user's fullname
public String getpassword(); — gets user's password
public void setpassword(String password); — sets user's passwords
public String getEmail(); — gets user's email
public void setEmail(String email); — sets user's email
public String getInformation(); — gets user's information
public void setInformation(String information); — sets user's information
public Limits getLimits(); — gets user's limits

public void setLimits(Limits limits); — sets user's limits
public void valueBound(HttpSessionBindingEvent event); — empty finction for interface realization
public void valueUnbound(HttpSessionBindingEvent event); — notifies user profile to save itself

5.7.2 Class Management

Description: Class for management. Functions in this classes manage default limits and users profiles. It is not necessary to use separate functions for managing user limits, because limits are stored in UserProfile class.

Fields:

private DeafultLimitsStore defLimitsStore; — object for default limits
private UserProfileStore userStore; — object for user profile
private Limits deafaultLimits; — default limits

Constructors :

public void UserProfileStore(); — initializes object with defLimitsStore and userStore objects

Methods:

public UserProfile getUserProfile(String nickname, String password); — checks accordance of nickname to the corresponding user profile log file; checks password; gets user profile; constrain user limits by default limits
public UserProfile getUserProfile(String nickname); — gets user profile
public void setUserProfile(UserProfile profile); — save userProfile
public delUserProfile(String nickname); — deletes user profile
public Limits getDefaultLimits(); — gets default limits
public void setDefaultLimits(Limits defaultLimits); — writes default limits
public void sendNotes(String note, ANLDE anldeSystem); — managing notes, send notes to the system administrator e-mail

5.8 Activity statistics

The class diagram used in activity statistics subsystem is shown in Fig. 42.

5.8.1 Class Statistics

Class Statistics (see Fig. 42) is used to transfer statistics data between ActivityStatistics and StatisticsStore and to store statistics data in SessionManager. Methods set*, add*, and get* are

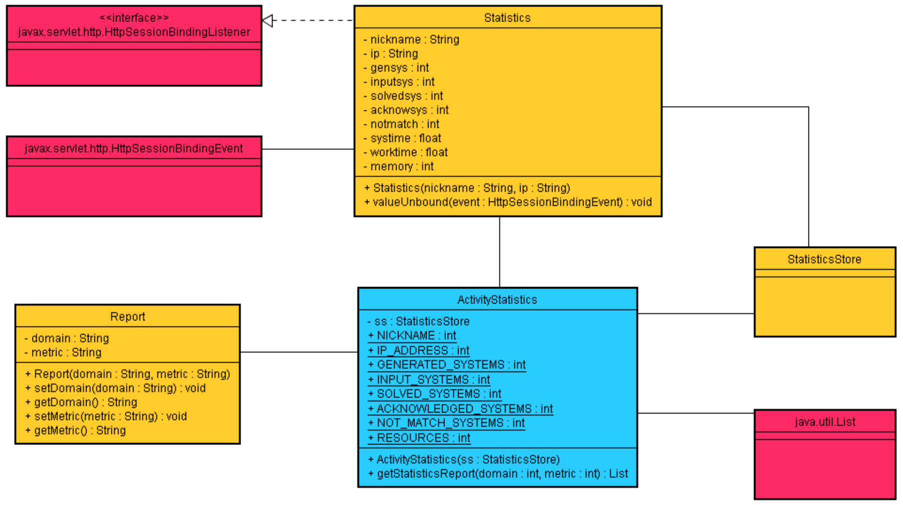

<<interface>>
javax.servlet.http.HttpSessionBindingListener
Statistics
- nickname : String
- ip : String
- gensys : int
- inputsys : int
- solvedsys : int
- acknowsys : int
- notmatch : int
- systime : float
- worktime : float
- memory : int
+ Statistics(nickname : String, ip : String)
+ valueUnbound(event : HttpSessionBindingEvent) : void
javax.servlet.http.HttpSessionBindingEvent
StatisticsStore
Report
- domain : String
- metric : String
+ Report(domain : String, metric : String)
+ setDomain(domain : String) : void
+ getDomain() : String
+ setMetric(metric : String) : void
+ getMetric() : String
ActivityStatistics
- ss : StatisticsStore
+ NICKNAME : int
+ IP_ADDRESS : int
+ GENERATED_SYSTEMS : int
+ INPUT_SYSTEMS : int
+ SOLVED_SYSTEMS : int
+ ACKNOWLEDGED_SYSTEMS : int
+ NOT_MATCH_SYSTEMS : int
+ RESOURCES : int
+ ActivityStatistics(ss : StatisticsStore)
+ getStatisticsReport(domain : int, metric : int) : List
java.util.List

Figure 42: Activity statistics

not shown on diagram.

Fields:

- private String nickname;
  user nickname.
- private String ip;
  user IP address.
- private int gensys;
  number of generated systems.
- private int inputsys;
  number of input systems.
- private int solvedsys;
  number of solved systems.
- private int acknowsys;
  number of acknowledged systems.
- private int notmatch;
  number of ANLDE systems which are discrepances solving.

- private float systime;
  total system time(sec).
- private float worktime;
  total work time(sec).
- private int memory;
  total memory used(Kb).
- private long startSession;
  start session time mark(ms).
- private long endSession;
  end session time mark(ms).

Constructors:

- Statistics(String nickname, String ip);
  constructs new Statistics object with corresponding *nickname* and *ip* and sets other attributes to zero.

Methods:

- public void valueUnbound(HttpSessionBindingEvent event); this function is called when Statistics object is unbound from session. The function extracts StatisticsStore object from servlet context and saves statistics data.
- public String getNickname();
  returns nickname.
- public String getIp();
  returns ip.
- public void setGeneratedSystems(int gensys);
  sets number of generated systems.
- public void addGeneratedSystems(int gensys);
  increases number of generated systems by *gensys*.
- public int getGeneratedSystems();
  returns number of generated systems.
- public void setInputSystems(int inputsys);
  sets number of input systems.
- public void addInputSystems(int inputsys);
  increases number of input systems by *inputsys*.

- public int getInputSystems();
  returns number of input systems.
- public void setSolvedSystems(int solvedsys);
  sets number of solved systems.
- public void addSolvedSystems(int solvedsys);
  increases number of solved systems by *solvedsys*.
- public int getSolvedSystems();
  returns number of solved systems.
- public void setAcknowledgedSystems(int acknowsys);
  sets number of acknowledged systems.
- public void addAcknowledgedSystems(int acknowsys);
  increases number of acknowledged systems by *acknowsys*.
- public int getAcknowledgedSystems();
  returns number of acknowledged systems.
- public void setNotMatchSystems(int notmatch);
  sets number of systems with solutions, which don't match.
- public void addNotMatchSystems(int notmatch);
  increases number of systems with solutions, which don't match, by *notmatch*.
- public int getNotMatchSystems();
  returns number of systems with solutions, which don't match.
- public void setSystemTime(float systime);
  sets system time.
- public void addSystemTime(float systime);
  increases system time by *systime*.
- public float getSystemTime();
  returns system time.
- public void setWorkTime(float worktime);
  sets work time.
- public void addWorkTime(float worktime);
  increases work time by *worktime*.
- public float getWorkTime();
  returns work time.
- public void setMemory(int memory);
  sets memory.

- public void addMemory(int memory);
  increases memory by *memory*.
- public int getMemory();
  returns memory.
- public void setStartSession(long startSession);
  sets start session time mark.
- public long getStartSession();
  returns start session time mark.
- public void setEndSession(long endSession);
  sets end session time mark.
- public long getEndSession();
  returns end session time mark.

5.8.2 Class Report

Class Report (see Fig. 42) is used to transfer processed statistics data (report data) from ActivityStatistics to statisticsreport.jsp.

Fields:

- private String domain;
  requested domain.
- private String metric;
  requested metric.

Constructors:

- Report(String domain, String metric);
  constructs new Report object with corresponding *domain* and *metric* attributes.

Methods:

- public void setDomain(String domain);
  sets domain.
- public String getDomain();
  returns domain.
- public void setMetric(String domain);
  sets metric.
- public String getMetric();
  returns metric.

5.8.3 Class ActivityStatistics

Class ActivityStatistics (see Fig. 42) is used to process statistics data and prepare report according to requested domain and metric.

Fields:

- StatisticsStore ss;
  instance of StatisticsStore object.
- public static final int NICKNAME;
  specifies nickname as report domain.
- public static final int IP_ADDRESS;
  specifies IP address as report domain.
- public static final int GENERATED_SYSTEMS;
  specifies number of generated systems as report metric.
- public static final int INPUT_SYSTEMS;
  specifies number of input systems as report metric.
- public static final int SOLVED_SYSTEMS;
  specifies number of solved systems as report metric.
- public static final int ACKNOWLEDGED_SYSTEMS;
  specifies number of acknowledged systems as report metric.
- public static final int DISCREPANCIES;
  specifies number of ANLDE systems, which discrepances solving as report metric.
- public static final int USED_SYSTEM_TIME;
  specifies summary used system time as report metric.
- public static final int USED_WORK_TIME;
  specifies summary used work time as report metric.
- public static final int SESSION_WORK_TIME;
  specifies summary session work time as report metric.
- public static final int SESSIONS;
  specifies number of sessions as report metric.

Constructors:

- ActivityStatistics(StatisticsStore ss);
  constructs new ActivityStatistics object.

Methods:

- public List getStatisticsReport(int domain, int metric);
  prepares statistics report according to requested *domain* and *metric*:
  1. tries to get list of Statistics objects from StatisticsStore;
  2. processes list of Statistics objects using algorithm:
     (a) get domain value (nickname or IP address) Statistics object (according to *domain* specified);
     (b) find Statistics object with the same domain value;
     (c) add some of found Statistics object metrics values (according to *metric* specified) to initial Statistics object metrics values;
     (d) remove found Statistics object from the list;
     (e) repeat for Statistics objects with other domain values;
  3. prepares sorted list of Report objects;
  4. returns list of Report objects.

6 Configuration and Installation

This section describes configuration design of the Web-SynDic system as well as its installation related issues.

6.1 Deployment Directory Layout

Web system's files are arranged in the following directory hierarchy according to the Java Servlet Specification (Version 2.3), Chapter 9:

/ — the document root of the web system, contains JSP, HTML and CSS files;

/images contains images used in the web-pages;

/WEB-INF contains non-public part of the web system (directory contents are not available to clients);

/WEB-INF/classes contains java class files (organized in packages);

/WEB-INF/algorithms contains algorithm server's configuration, external solvers and generators (web system may be configured to use another directory using "algorithmServerDirectory" initial ServletContext parameter);

/WEB-INF/datastore — data store directory, the web system must have write access to this directory (web system may be configured to use another directory using “dataStoreDirectory” initial ServletContext parameter);

/WEB-INF/datastore/statistics contains log files for computing activity statistics;

/WEB-INF/datastore/accounts contains user profiles;

/WEB-INF/datastore/limits-default — file for storing default limits;

6.2 Configuration

Configuration files:

/WEB-INF/web.xml — the Web Application Deployment Descriptor, initial ServletContext parameters, session timeout and other configuration options are set here (see the Java Servlet Specification, Appendix A);

/WEB-INF/algorithms/solvers.properties — list of available solvers with their supporting classes (in the format used by class java.util.Properties, where keys are solvers directory names, values are class names, see the API documentation for the java.util.Properties class, load() method);

/WEB-INF/algorithms/generators.properties — list of available generators with their supporting classes (in the format used by class java.util.Properties, where keys are solvers directory names, values are class names);

6.3 Source Tree

The web system's source code organized in the following hierarchy in the CVS repository:

/src — java sources (organized in packages), all Web-SynDic classes belongs to the subpackages of the ru.petrsu.websyndic package;

/web — static content, the document root of the web system;

/doc — documentation directory;

/build — compilation directory;

/build.xml — targets definitions for the Ant build tool.

6.4 Building and Installing

The following ant targets may be used to build and deploy the web-system:

compile (default): compiles the we-system;

clear: removes compiled class-files;

install: installs the web-system into the servlet container;

remove: removes the web-system form the servlet container;

reload: reloads the web-system installed in the servler container;

dist: creates a binary distribution of the web-system;

javadoc: creates Javadoc API documentation;

test: invokes Test class in the default package (used for testing subsystems).

Use the following command to invoke a target:

```
ant <target>
```