Saint-Petersburg Russia

Microsoft Technologies in Theory and Practice of Programming

Contest – Conference

High-Level Design Description

Project:	Web-SynDic
Customer:	Department of Computer Science, Head of the Department Yury A. Bogoyavlensky Petrozavodsk State University
Developers:	Dmitry G. Korzun (senior manager, PhD) Kirill A. Kulakov (technical manager, master student, BSc) Andrey Y. Salo (senior student) Andrey A. Ananin (junior student) Mikhail A. Kryshen (junior student)
Product:	Web System for Demonstrating and Testing the Syntactic Algorithms for Solving Linear Diophantine Equations in Nonnegative Integers

---

Contents

1 Introduction

The Web-SynDic project[1] is a student software engineering (SE) project of the Petrozavodsk State University (PetrSU), Department of Computer Science (CSDept).

The project is related to the research done at CSDept of PetrSU in development of a new type of algorithms for efficient solving some classes of nonnegative linear Diophantine equations (NLDE) by syntactic (parsing) methods [10, 11, 12]. These syntactic algorithms seem to be promising tool for solving some classes of NLDE system; more exactly a class of NLDE system, associated with formal grammars (ANLDE systems). For this class the syntactic algorithms allow efficient (polynomial and pseudo-polynomial) computations comparing with the general NLDE case when the same computational problems are NP-complete or even overNP [13].

The general goal of the project is to develop a full function web system[2] for visual demonstrating and testing the syntactic algorithms via the Internet. This allows researchers to input ANLDE systems (manually or automatically generated), search their Hilbert bases, test the correctness of the found solution, estimate the resource consumption, and compare the efficiency with available solvers, different from syntactic.

1.1 Purpose

This document defines the developed decomposition of the Web-SynDic system into principal subsystems. Interrelations and interface between these subsystems are stated as well.

The goal is to have a base for further, low-level design: subsystems, user interface, data models, algorithms, configuration, and installation. The objectives are in constructing a high-level model of how to implement sufficiently the requirements, stated in the Requirement Specification document.

The document is intended mainly for the project development team. Experts from customer's side may analyze this document to be sure that the requirements are going to be implemented sufficiently and efficiently.

This specification is frozen by the end of the design phase. Any changes to the design are described in a separate document—The Implementation Document[3].

Complete, both high- and low-level design can be found in the original project document "Design Specification" at `http://zeta.cs.karelia.ru/Web-SynDic/doc/eng/design`.

[1] The original, more detailed document set of Web-SynDic Project can be found at [1].

[2] The Web-SynDic Server is published at [2].

[3] This document can be found at `http://zeta.cs.karelia.ru/Web-SynDic/doc/eng/implementation/`

1.2 References

[1] Web-SynDic Project development and maintenance site.
`http://zeta.cs.karelia.ru/Web-SynDic/doc/eng/` (English)
`http://zeta.cs.karelia.ru/Web-SynDic/doc/rus/` (Russian)

[2] The primary Web-SynDic Server is installed at
`http://zeta.cs.karelia.ru:8080/Web-SynDic/main.jsp`

[3] Web-SynDic: Requirements Specification. The original version.
`http://zeta.cs.karelia.ru/Web-SynDic/doc/eng/requirements/`

[4] Web-SynDic: Design Specification. The original version.
`http://zeta.cs.karelia.ru/Web-SynDic/doc/eng/design/`

[5] Web-SynDic: Software Requirements Specification. Version for the Microsoft Conference, March 2004.

[6] Roger S. Pressman. *Software Engineering. A Practitioner's Approach.* European adapt., 5th ed. McGraw-Hill, 2000. 915 p.

[7] Ian Sommerville. *Software Engineering.* 6th ed. Addison-Wesley, 2000.

[8] Graig Larman. *Applying UML and Patterns.* Prentice Hall, 2000.

[9] Jim Conallen. *Building Web Applications with UML.* Addison-Wesley, 2000.

[10] Yury A. Bogoyavlensky, Dmitry G. Korzun, *Obshchiy vid resheniya sistemy lineynih diophantovih uravneniy, associirovannoy s kontextno-svobodnoy grammatikoy.* Trudy Petrozavodskogo gosudarstvennogo universiteta. Ser. "Prikladnaya matematika i informatika". Vol. 6. Petrozavodsk, 1998. pp. 79–94. (in Russian)

[11] Dmitry G. Korzun. *Syntactic Algorithms for Solving Nonnegative Linear Diophantine Equations and their Application for Modelling of Internet Link Workload Structure.* PhD Thesis, Department of Computer Science, University of Petrozavodsk, 2002. 185 p. (in Russian)

[12] Dmitry G. Korzun. *Grammar-Based Algorithms for Solving Certain Classes of Nonnegative Linear Diophantine Systems.* Proceedings of Annual international Finnish Data Processing Week at the University of Petrozavodsk (FDPW'2000): Advances in Methods of Modern Information Technology. Vol. 3. Petrozavodsk, 2001, pp. 52–67.

[13] Schrijver A. *Theory of linear and integer programming.* Wiley, Chichester, 1986.

[14] Domenjoud E. *Solving Systems of Linear Diophantine Equations: An Algebraic Approach.* In U. Tarlecki (ed.), Proceedings of 16th International Simposium on Mathematical Foundations of Computer Science. Springer–Verlag, 1991. LNCS 520. PP. 141–150.

1.3 Document conventions

In diagrams, different colors are used to emphasize roles of each problem domain object. The convention about colors, used in models, is stated in Figure 1. The Web-SynDic area defines

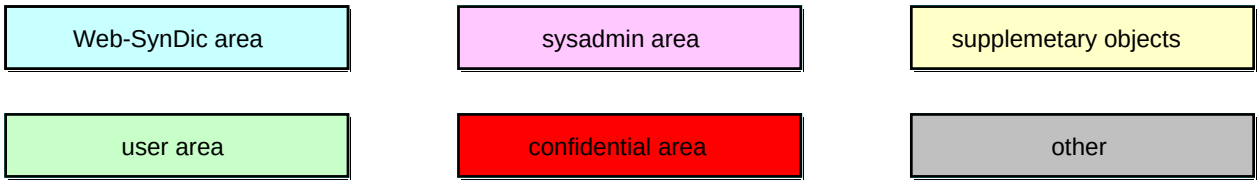

Figure 1: Color convention for models

general bounds of the Web-SynDic system. The user area contains objects that user may have access to. The sysadmin area is for activity of the system administrator. The confidential area must not be accessible by a user (e.g. the external algorithms).

Term	Description
ANLDE system	**A**ssociated with a formal grammar, NLDE system. See [10, 11].
Browser	Standard Internet browser, e.g. Microsoft IE 6.0.
Client	Web-SynDic—access point for a user. Available with standard Internet browser.
CS	Computer Science
CSDept	Computer Science Department. The PetrSU CSDept web-site is `http://www.cs.karelia.ru`
Data store	Storage of users data, limits and other supplementary data for Web-SynDic.
Hilbert basis	A set of all indecomposable (minimal) solutions of a homogeneous NLDE system.
HTTP	HyperText Transmission Protocol.
Generator	Executable program that implements a generating algorithm.
ILP	Integer linear programming
Indecomposable solution	A particular solution that is not a sum of two particular solutions.

lp_solve	The non-commercial linear programming code, written in ANSI C by Michel Berkelaar. Also it supports ILP problems. Available on `http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml`
NLDE	Nonnegative linear Diophantine equations, i.e. their solutions are in nonnegative integers and coefficients are integer. See for example [13, 14].
Particular solution	Any non-trivial solution of a homogenous NLDE system.
PetrSU	Petrozavodsk State University, `http://petrsu.karelia.ru`
SE	Software Engineering, standard course books are [6, 7].
Server	Web-SynDic server part. Performs all processing.
Slopes	Algorithm of M.Filgueiras and A.-P.Tomás for searching Hilbert basis of a homogenous NLDE system, available on `http://www.ncc.up.pt/~apt/dioph/`
Solver	Executable program that implements a solving algorithm.
Syntactic Algorithms	The algorithms that solve ANLDE system by constructing some derivations in the corresponding formal grammar, see [12, 11]. Web-SynDic is intended to demonstrate and test such algorithms.
Trivial solution	All-zero solution $\mathbb{0} = (0, \ldots, 0)$ of a homogeneous NLDE system.
UML	Unified Modelling Language. See for example [8, 9].
Web-SynDic	It stands for **Web**–based demonstrating and testing the **Syn**tact**ic** algorithms for solving nonnegative linear **D**iophantine equations.

2 System Architecture

In this section the architecture of the Web-SynDic system is presented. Key subsystems are identified, for each of them their purpose and specifics are described. The section results with the integral UML-based model of the architecture [8, 9].

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

2.1 Integral high-level architecture

The Web-SynDic system is divided into three key subsystems (Fig. 2). *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. The internal high-level static structure of the subsystems is described in subsections 2.2, 2.3, and 2.4. Detailed design for collaboration of these subsystems is in section 3.

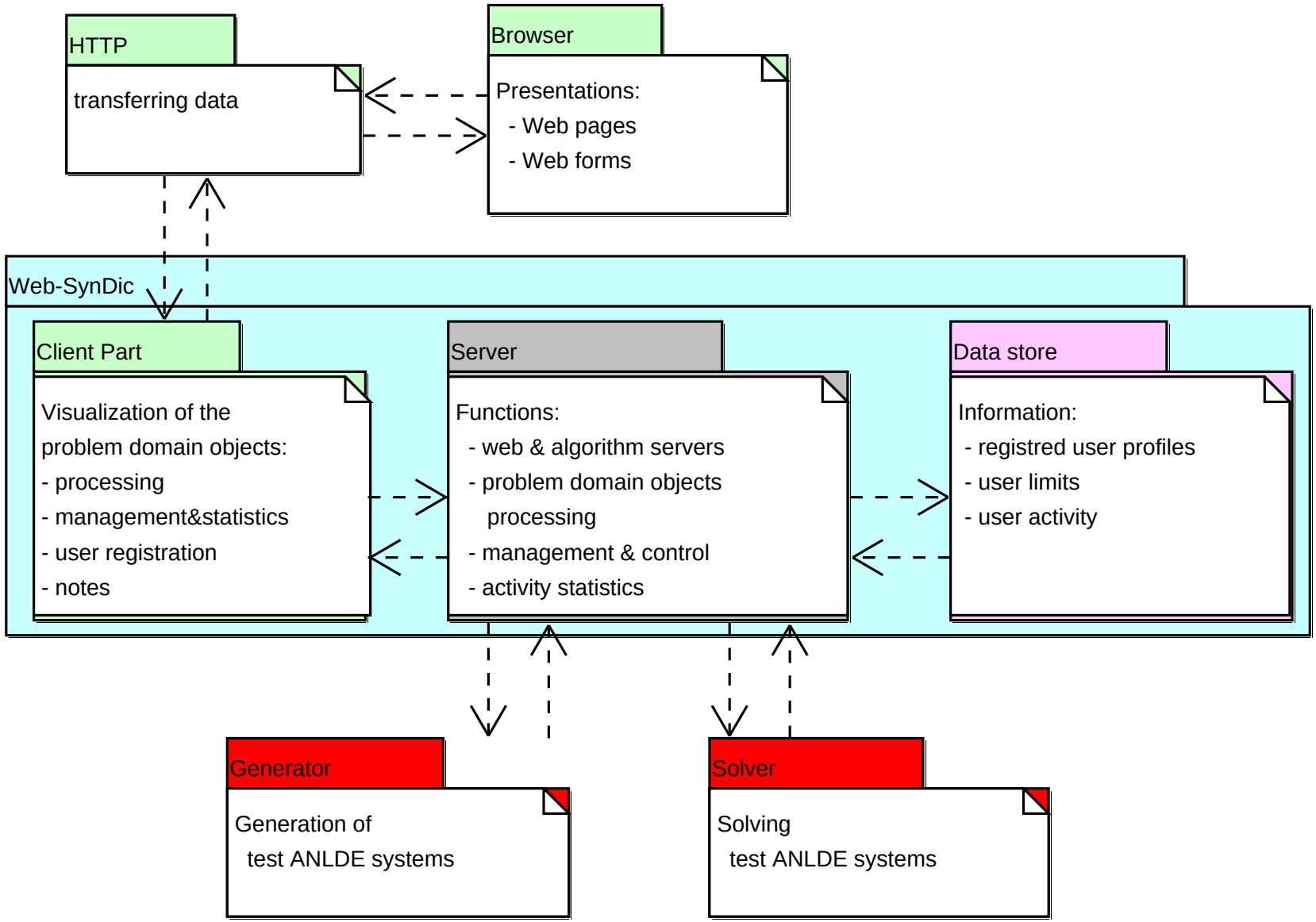

Figure 2: Integral architecture of the Web-SynDic system

Each subsystem has its interface functions. They provide methods to access the data and to interact with the subsystem. In this section only high-level interface functions are presented; for the detailed interface design see section 4.

The team decided to use:

1. Thin web client technology for the software architecture according to Req. AD1.

2. Java was used as basic programming language according to Req. AD2 and AD3.

The satisfaction with AD1 is clear. Req. AD2 is satisfied because Java is cross-platform language. In principle, the satisfaction with Req. AD3 can be easily made via existing CASE tools for Java $\rightarrow$ J# conversion. At this moment these tools are not available for the team.

External algorithms are available as C and C++ code. The team will use `cygwin` package to port them under OS Windows family and `gcc` compiler under Linux.

Testing of the implementation will be executed under OS Windows and Linux.

2.2 Server subsystem

The *Server* subsystem is divided into five modules, see Figure 3.

The *Web server* module is an interface service point between a user and the Web-SynDic system. The web server receives requests and input data from a client part, converts them from an input format to internal one, 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 *Session processing* module manages all established 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 this high-level architecture it is presented as a separate module due to its key role in the coordination.

The *Algorithm server* module performs processing of problem domain objects according with the required demonstrating and testing rules. The server executes external algorithms whenever it is required by a user during her/his session, and passes the outcome to the *Session processing* module.

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

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

Figure 3: Architecture of the *Server* subsystem

2.3 Client Part subsystem

The *Client part* subsystem is an interface between a browser (external entity) and the *Web server* subsystem. This is a starting point for a user to access available functions. This subsystem does not require explicit implementation: standard mechanisms of a browser and the 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 4 for the classification and the Requirements Specification for the use case description.

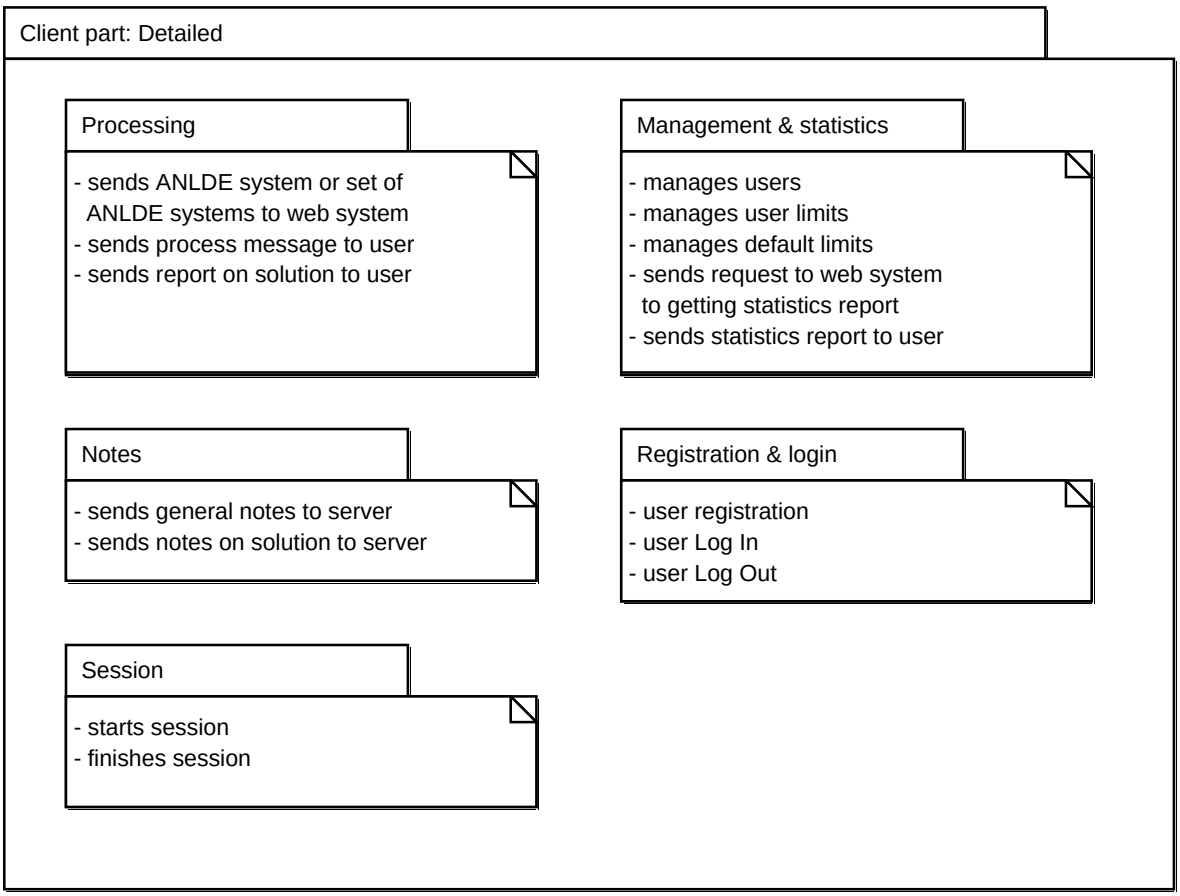

Figure 4: Architecture of the *Client Part* subsystem

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.

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, servlet or java class/package), see section 3 for details.

2.4 Data Store subsystem

The *Data store* subsystem is divided into three modules, see Figure 5.

User profile A user profile is a set of data for 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:

1. version of user profile format,
2. password,
3. email,
4. user limits,
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 are 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 the time stamp of an atomic usage event, user ID (nickname for a registered user, IP address for a regular user) and an activity parameter (type of user activity and particular attributes of this activity).

Figure 5: Detailed architecture of the Data Store

Activity data are implemented as a set of log files. Each log file is list of rows, each row is a record with detailed information about an atomic usage of the Web-system:

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

It is assumed that one 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 during lifetime of the Web-system. Default value of log file period is one month.

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

When a user registers in the Web system, corresponding user profile is created and added to the set of user profiles; at the beginning the user limits are just a copy of limits of anonymous user. The profile contains all data of the user including his/her limits.

A registered user may change the user limits. User 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 anonymous user) and the changes are valid for this session only.

3 Subsystems Collaboration

3.1 Work with Web-SynDic

Working with Web-SynDic requires a user to start a session; she/he may work only within a session. Figure 6 shows the behavior of session starting and finishing.

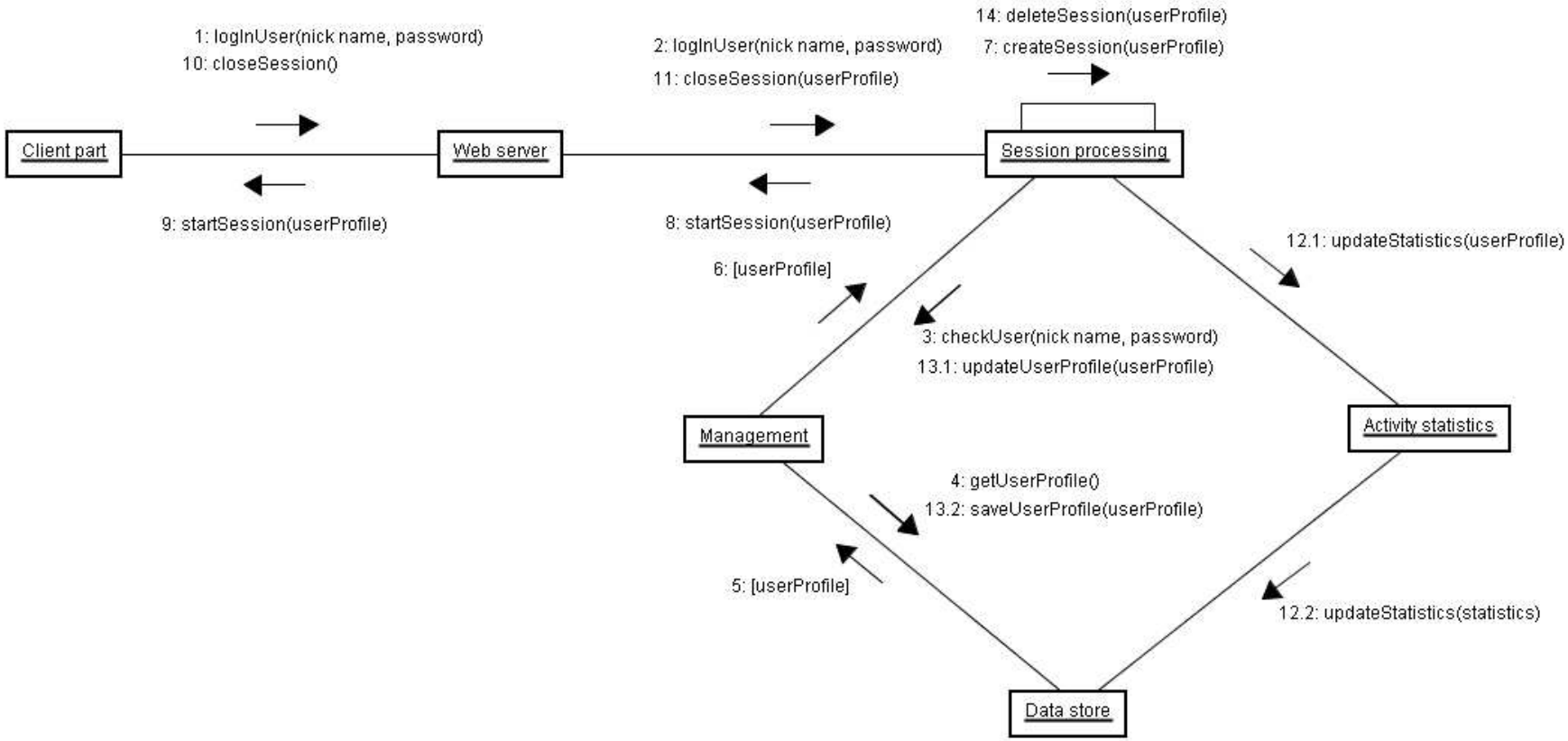

Figure 6: Work with Web-SynDic Collaboration Diagram

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 is returned,

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

7: createSession(userProfile) creates session, identified with the user nickname, or creates a session for anonymous (non-registered) 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.

3.2 Log In

Log In behavior is shown in Figure 7. A user inputs her/his nickname and password. If they are correct, then the login is successful and the user may work with Web-SynDic as a registered one. Otherwise, the invalid login message is displayed in the "Content" part and the user may work only as anonymous (non-registered) 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.

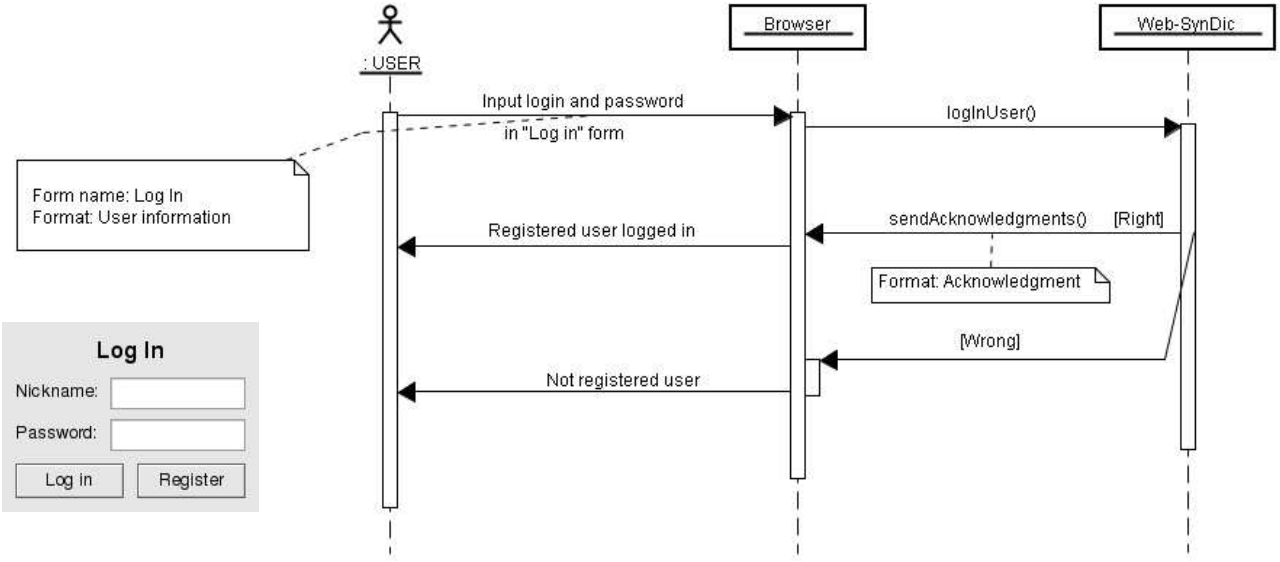

Figure 7: Log In sequence diagram

3.3 Process an ANLDE system

For processing a test ANLDE system, a user may initialize four flows, see Figure 8.

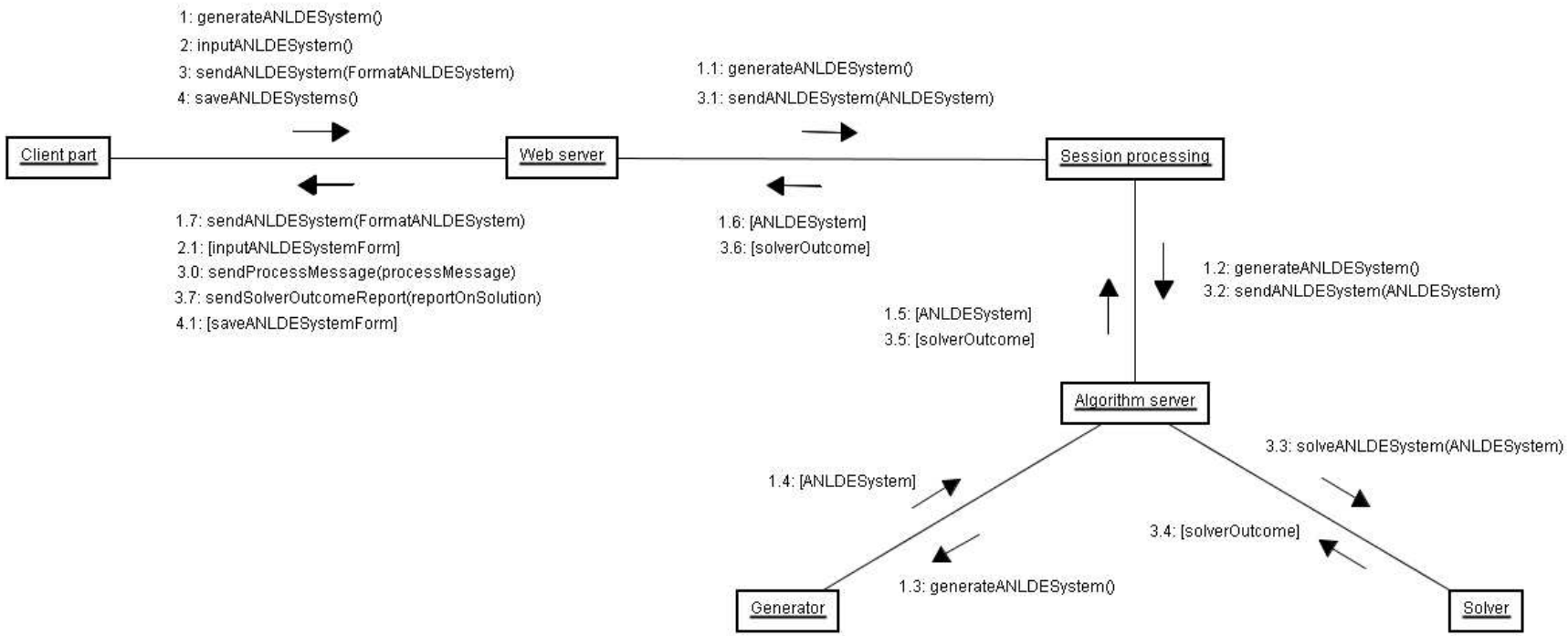

Figure 8: Process an ANLDE system Collaboration Diagram

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. For generation process the user may choose a generator (corresponding item "Generator"). If the user has not chosen 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"; 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. If the set of ANLDE systems does not satisfy the user limits, then the error message is also displayed in the "Content" part.

If the entered ANDLE system is valid, then the process message is displayed in the "Content" part. After solving the system, a report on solution is displayed in "Content".

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.

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. 9.

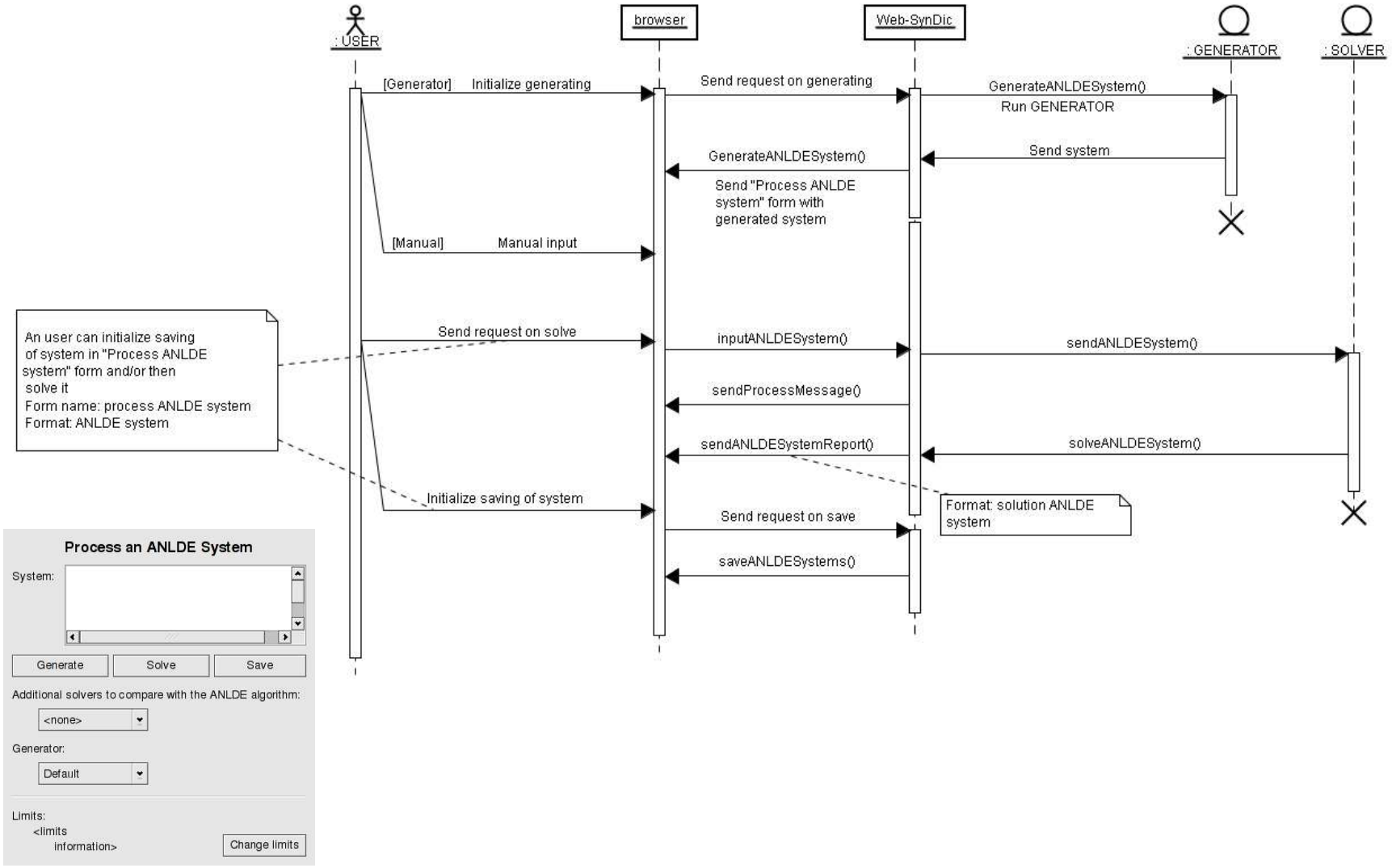

Figure 9: Process an ANLDE System sequence diagram

3.4 Process a set of ANLDE systems

For processing a set of test ANLDE systems, a user may initialize four flows, see Figure 10.

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,

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.

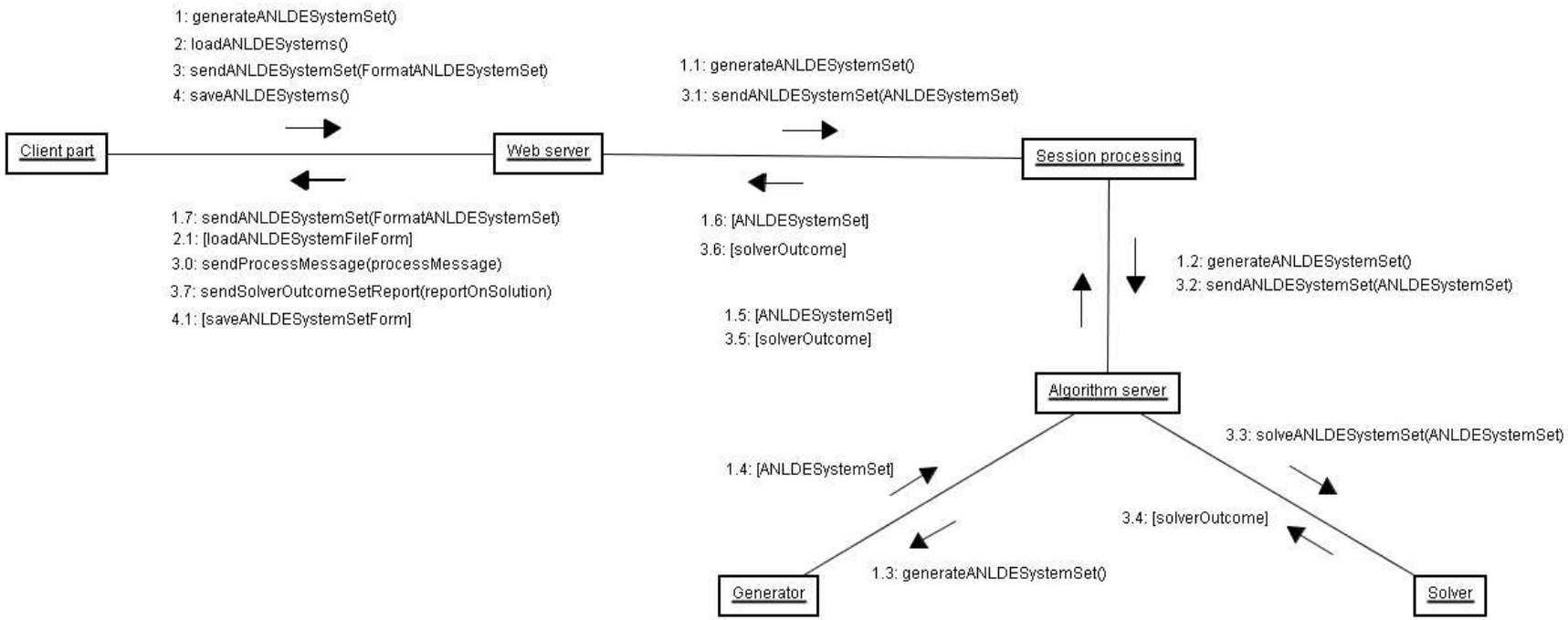

Figure 10: Process a set of ANLDE systems Collaboration Diagram

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. Then the web system checks the given set of ANLDE systems. If the set is incorrect, then an error message is displayed in “Content”. If the set does not satisfy user limits, then an error message is also displayed in “Content”.

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.

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

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. 11

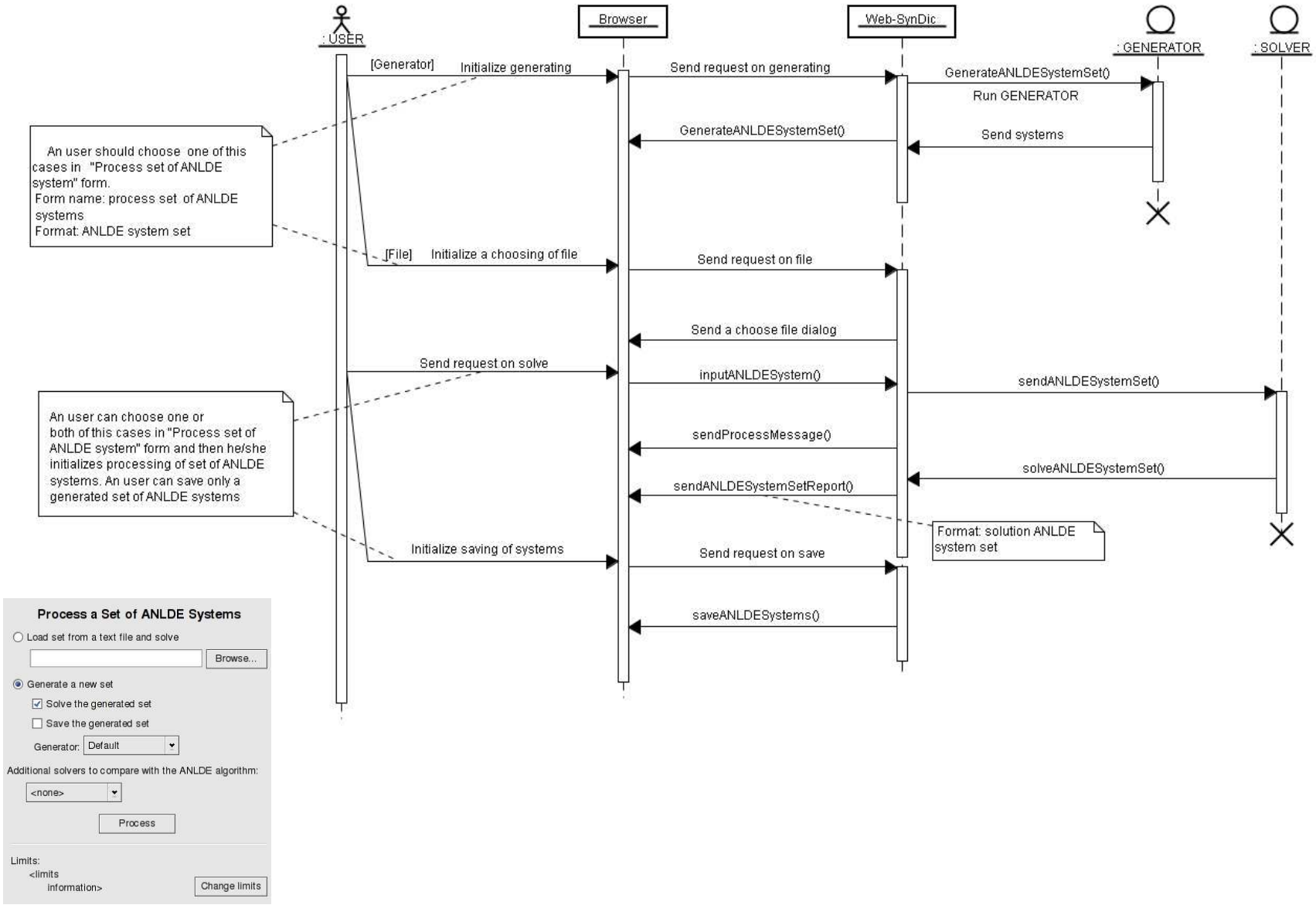

Figure 11: Process a Set of ANLDE Systems sequence diagram

3.5 Register a user

The web system allows a non-registered user to register when she/he wishes. The registration process has the following behavior, see Figure 12.

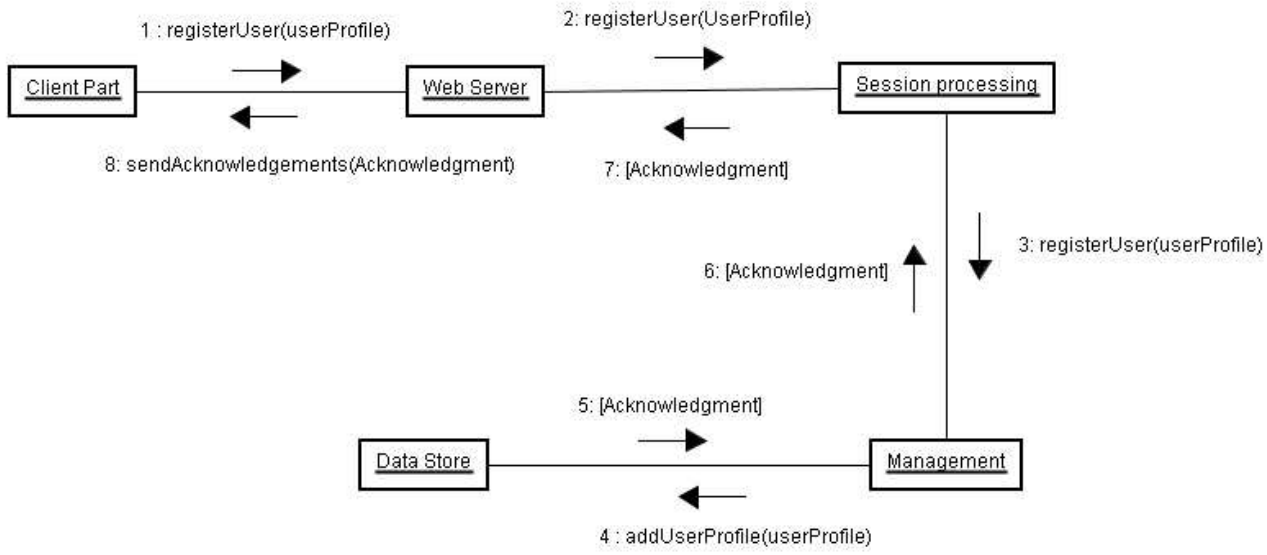

Figure 12: Register a user Collaboration Diagram

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;

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 registration 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 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. If the values are correct, then the acknowledgment message is displayed in the ”Content” part.

This form corresponds to the ”Register a user” use case, see Figure 13

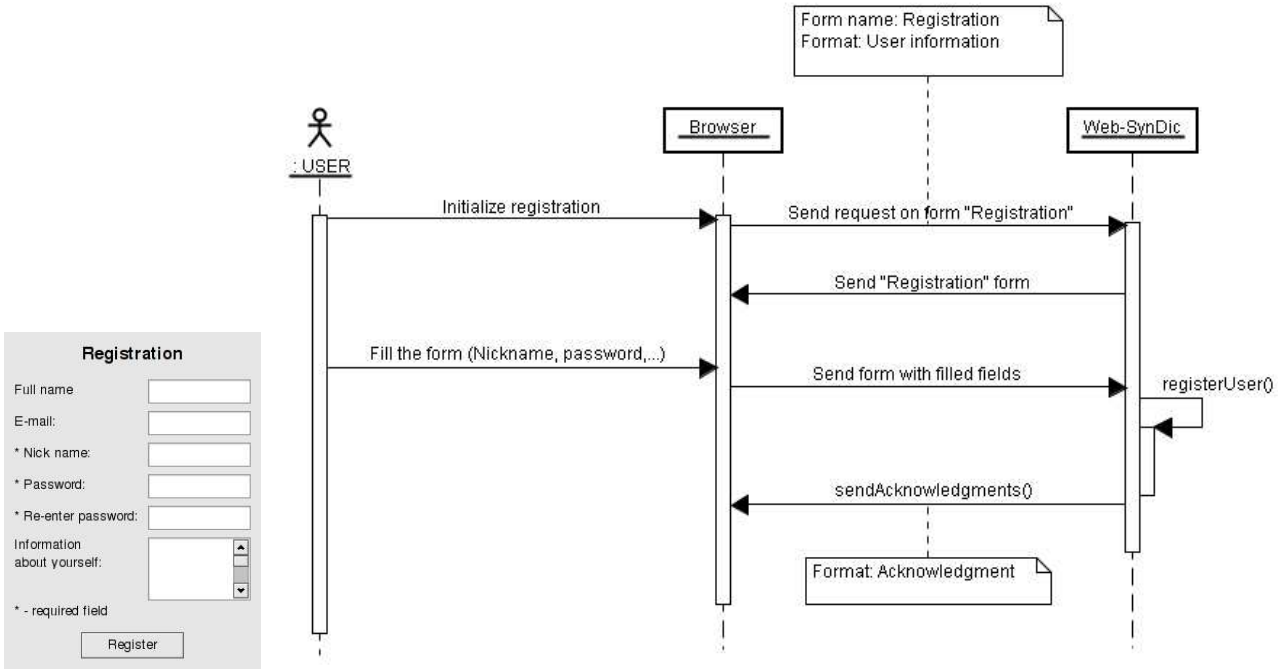

Figure 13: Register a User sequence diagram

3.6 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 14.

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

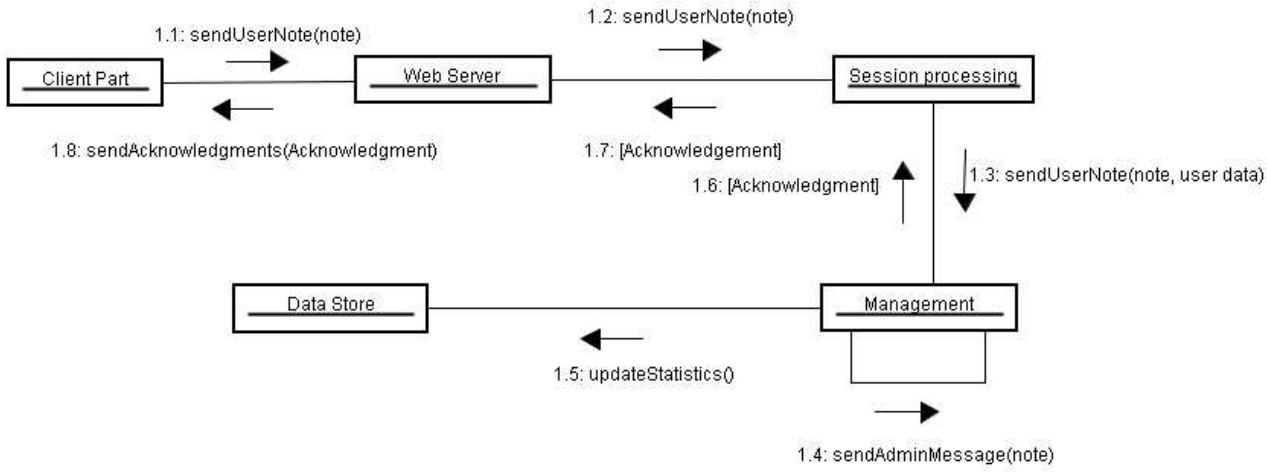

Figure 14: Send user notes Collaboration Diagram

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 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. If not, then the acknowledgment message is displayed in the "Content" part.

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 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. If not, then the acknowledgment message is displayed in the "Content" part.

This form corresponds to the "Send a note" use case, see Fig. 15.

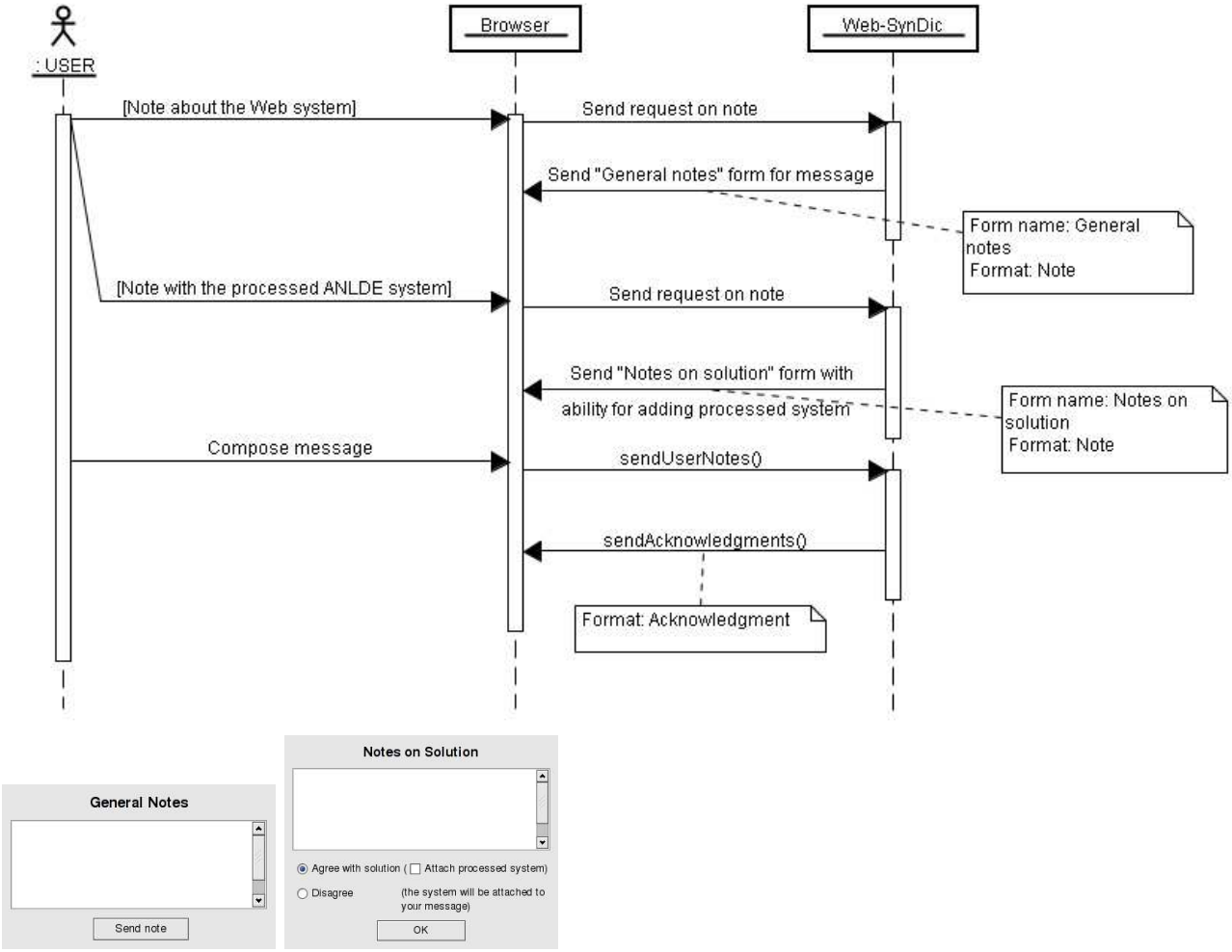

Figure 15: Send a Note sequence diagram

3.7 Manage user limits

For management of her/his limits on generation and solution processes, a user may initialize two flows, see Fig. 16.

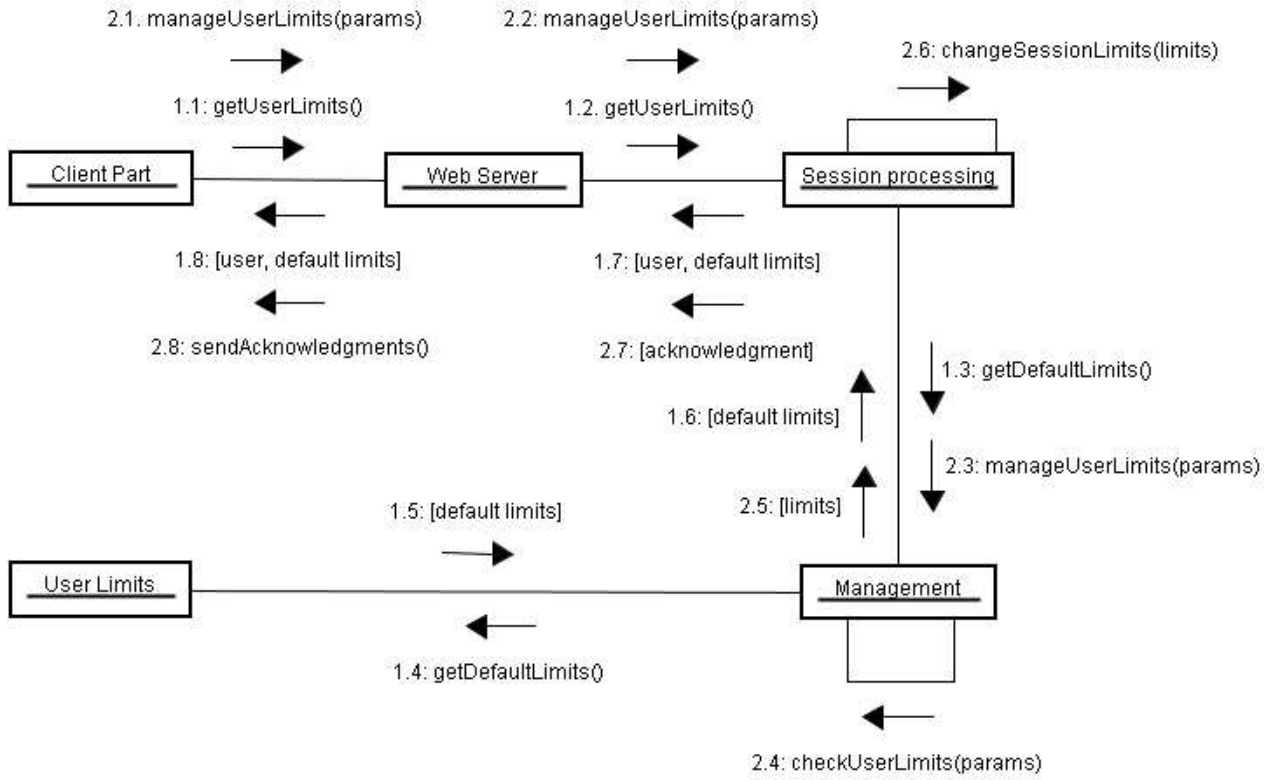

Figure 16: Manage User Limits Collaboration Diagram

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;

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. The user changes values of the current user limits. If the new values are correct, then acknowledgment message 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.

This form corresponds to the "Manage user limits" use case, see Fig. 17.

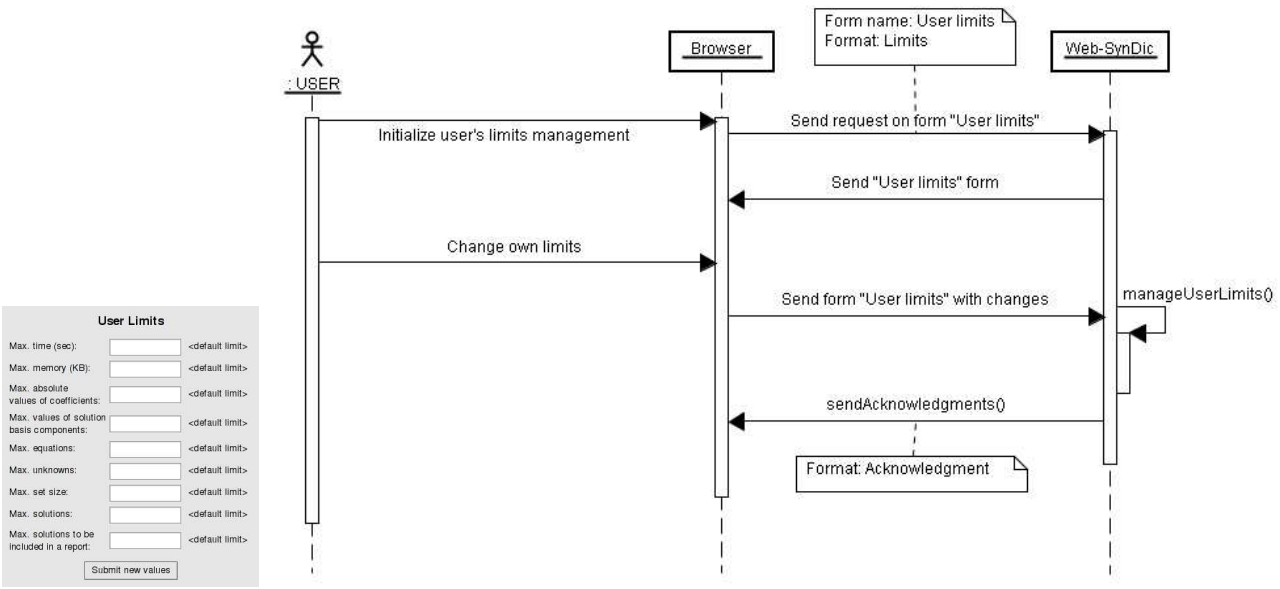

Figure 17: Manage User Limits sequence diagram

3.8 Manage default limits

The system administrator may initialize two flows as it is shown in Figure 18.

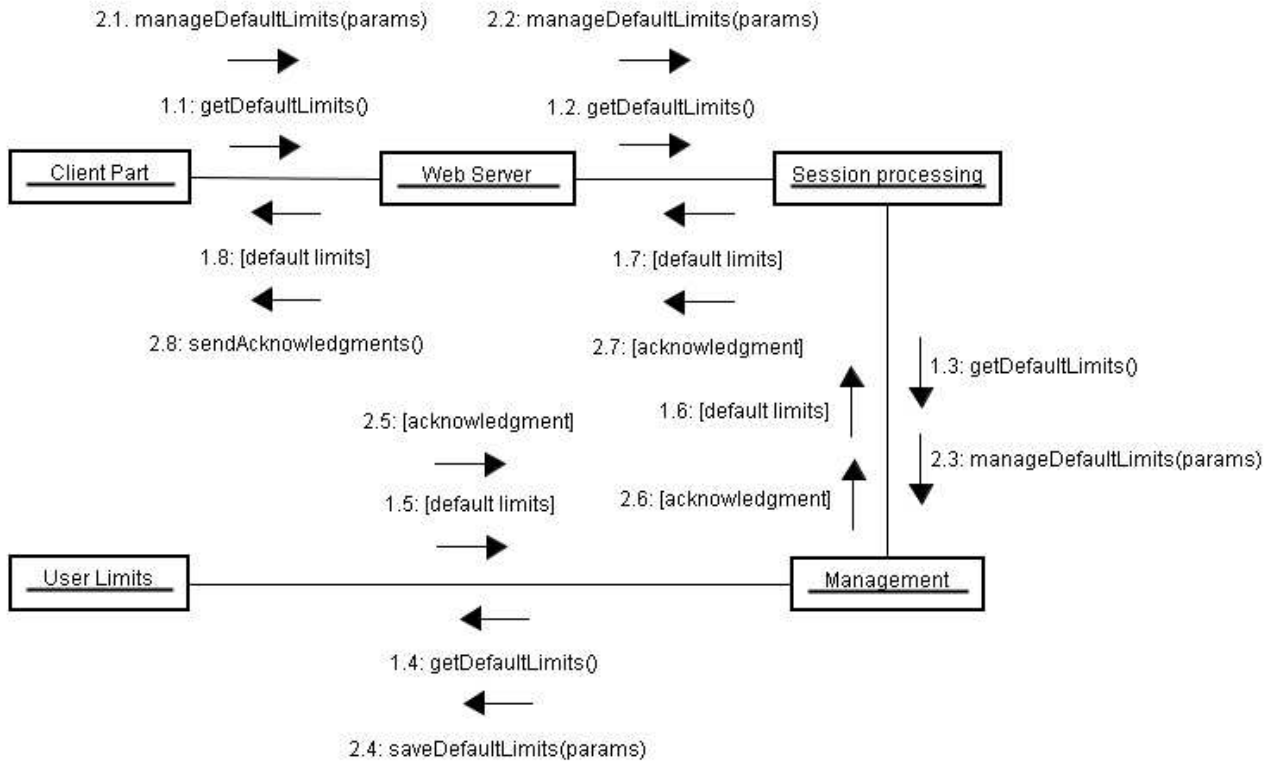

Figure 18: Manage Default Limits Collaboration Diagram

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

1.1: getDefaultLimits() requests for current default limits;

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. The system administrator changes values of default limits and presses "submit new values" button. If the new values are correct, then the acknowledgment message is displayed in the "Content". Otherwise, "Default limits" form will be reloaded in the "Content" part and the error is indicated.

This form corresponds to the "Manage default limits" use case, see Fig. 19.

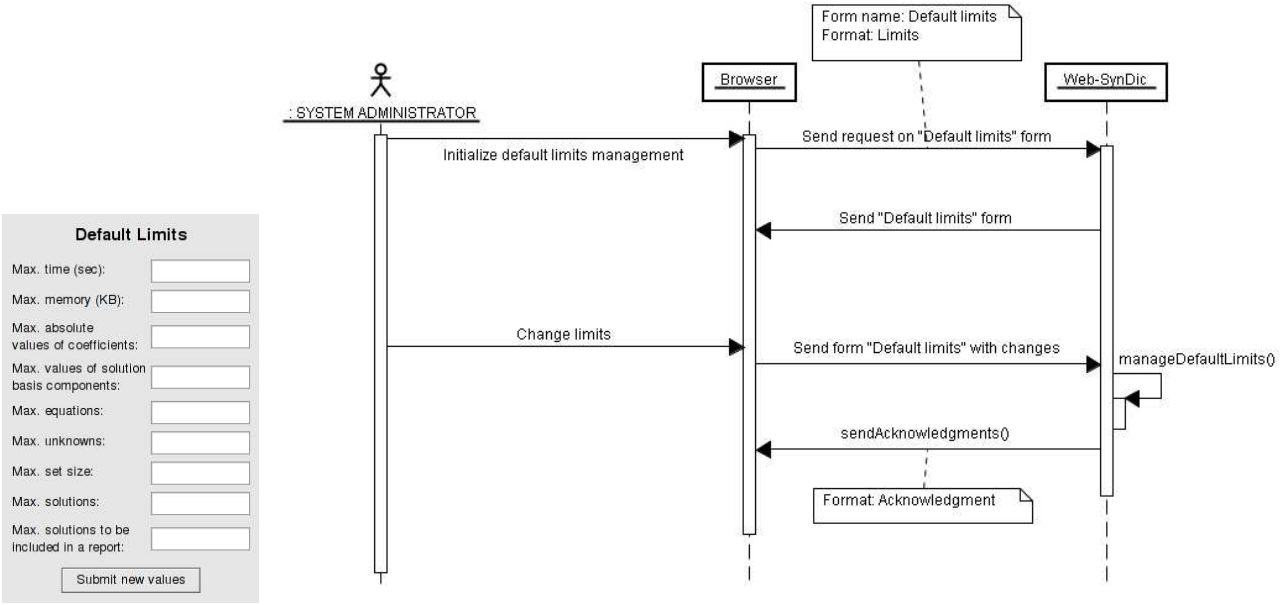

Figure 19: Manage Default Limits sequence diagram

3.9 Get statistics

The system administrator may initialize the only flow to request statistics with chosen metrics, see Fig. 20.

1.1: sendRequestStatActivity(params) sends parameters "params" and requests to produce the activity statistics report;

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. 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 is displayed in the "Content" part. After choosing activity domain and activity metrics, she/he presses "Get report" button and the report is loaded to the "Content" part.

This form corresponds to the "Get statistics" use case, see Fig. 21.

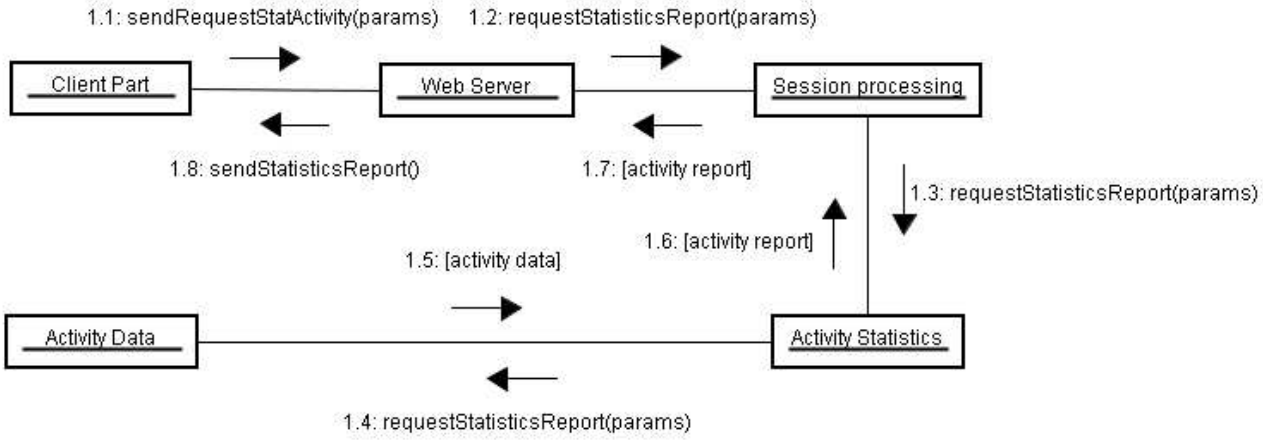

Figure 20: Get Statistics Collaboration Diagram

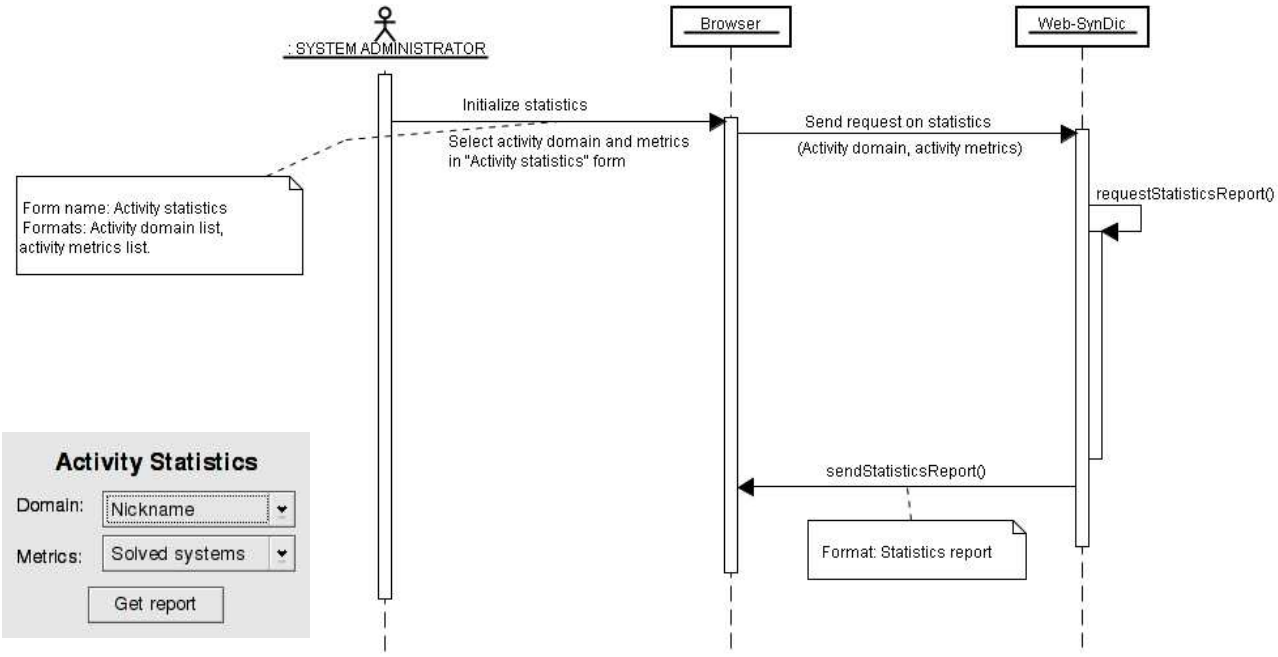

Figure 21: Get Statistics sequence diagram

3.10 Manage users

For user management, the system administrator may initialize two flows, see Figure 22.

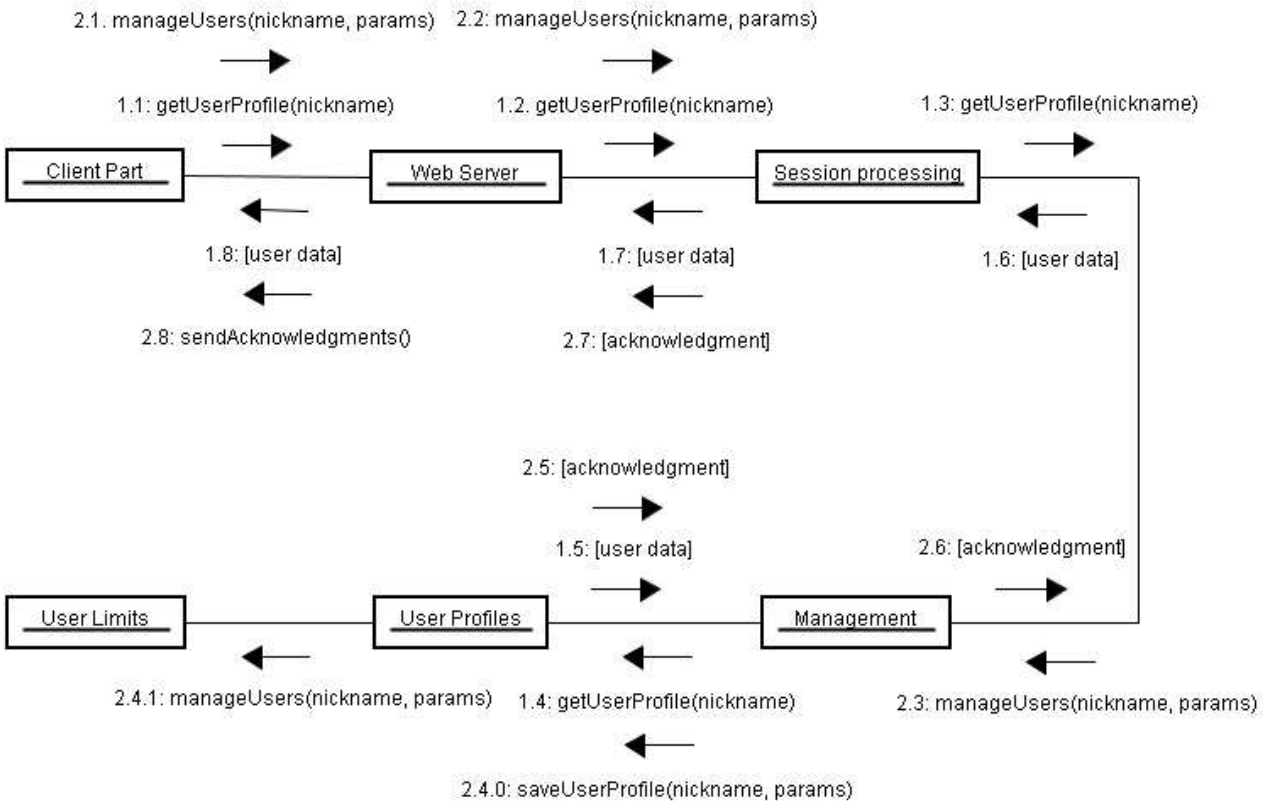

Figure 22: Manage Users Collaboration Diagram

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

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 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 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.

These forms correspond to the "Manage users" use case, see Fig. 23.

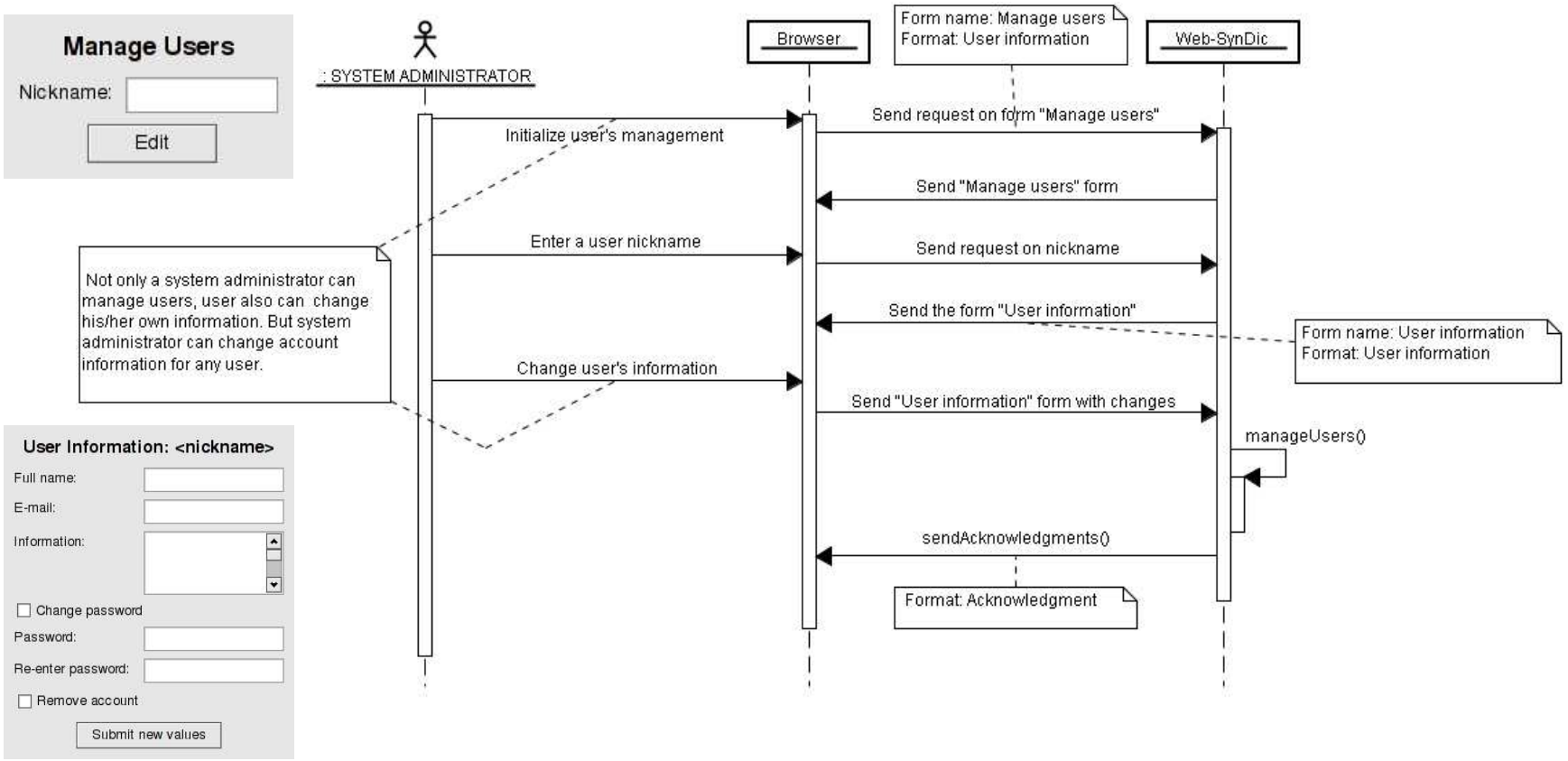

Figure 23: Manage Users sequence diagram

4 Subsystems Interface

The composition model for subsystems interface is shown in Figure 24.

4.1 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. Interface "IBrowser-Client" consists of the following functions:

getIPAdress() returns user IP address.

4.2 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,

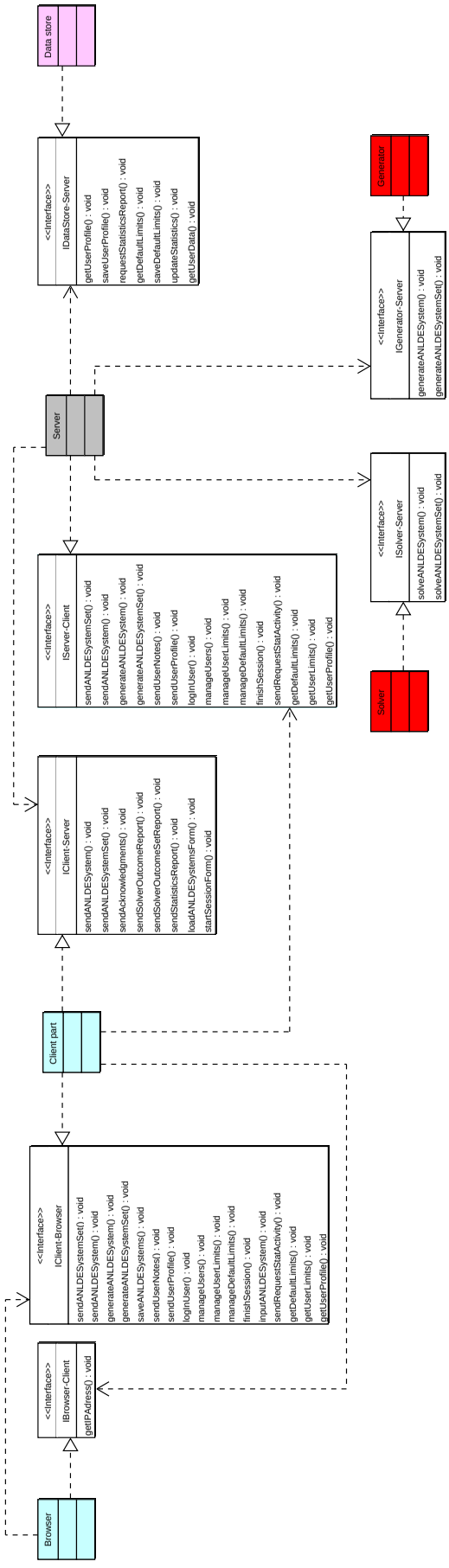

Figure 24: Detailed subsystems interface

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.

4.3 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.

The composition model for server modules interface is shown in Figure 25.

4.3.1 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.

4.3.2 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,

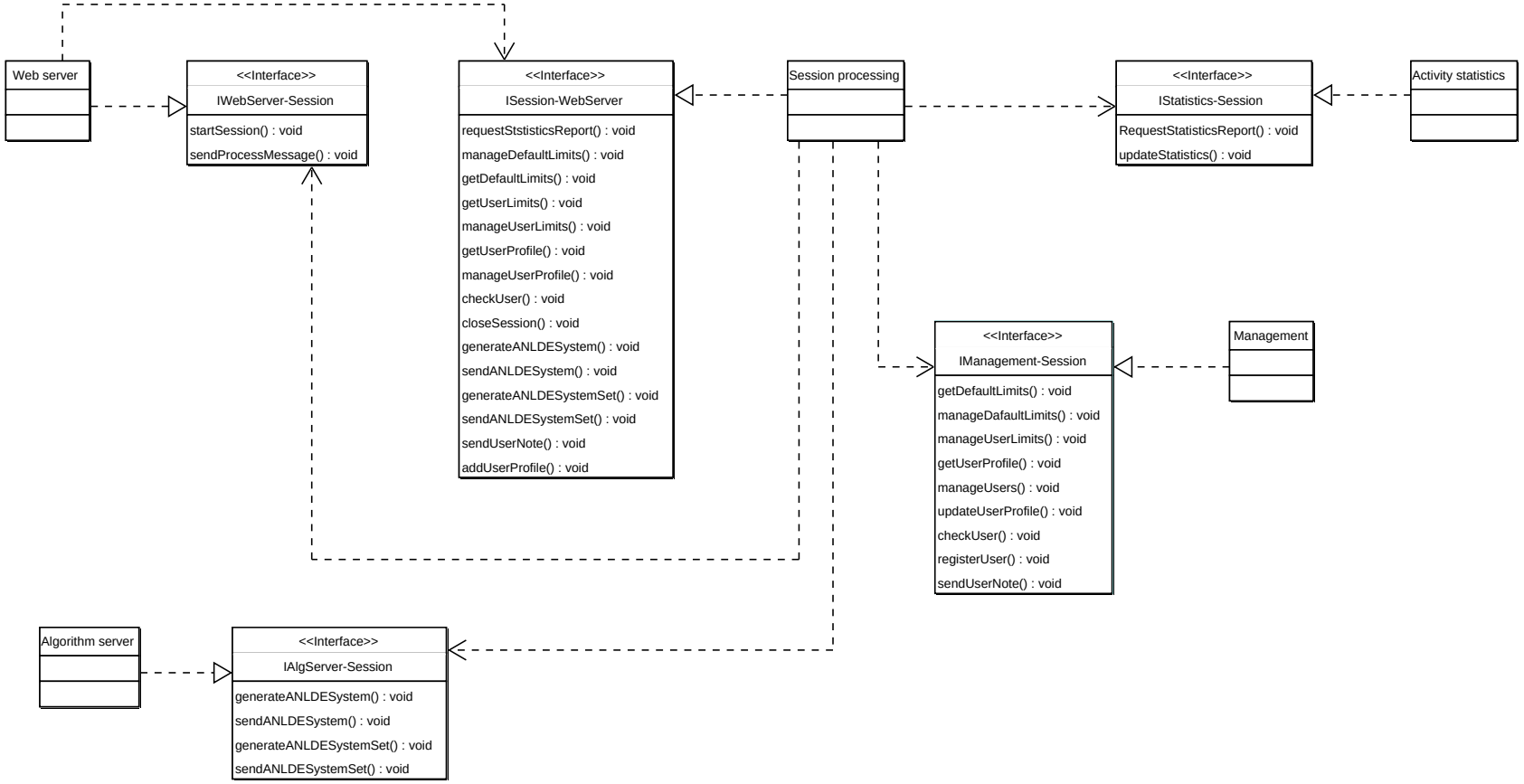

Figure 25: Detailed server subsystems interface

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.

4.3.3 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,

4.3.4 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.

4.3.5 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.

4.4 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.

4.5 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.

4.6 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.