Building an E-Contract Management System Using Google Docs

Gabriela Visinari
Computer Science Department
Technical University of Cluj Napoca, Romania
Email: visinari.gabriela@bavaria.utcluj.ro

Adrian Groza
Computer Science Department
Technical University of Cluj Napoca, Romania
Email: adrian.groza@cs.utcluj.ro

***Abstract*—This paper introduces a way for approaching electronic contracts management involving two parties. Contract monitoring automation is achieved by giving legal clauses a formal representation. Inspired from the file system properties, a mechanism for dynamic reorganization of contracts on Google Docs is proposed.**

***Keywords*-e-contracts; management system;**

I. Introduction

New technologies have changed the business environment and provided the trading process in e-business more efficiently [1]. Consider a lawyer that designs a contract template for a business entity. He uses that template because many of the agreements with company's clients are related: the company may offer the same service to more clients. The lawyer contacts the administrator of the e-contract management system and sends him this template. An expert in Jess formalizes the clauses in the contract as Jess rules. Even before the coding begins, if the formalization of clauses is terminated, the lawyer can give his approval with respect to contract execution, since a Jess rule file can be ran even from Jess console, independent of the system. For business related reasons, once a lawyer agrees to have a contract template on the application's server, he also agrees that other users of the system may use that template for the creation of their contracts. So, even if a new company with a company representative lawyer (CRL) that does not have an account in the system supports the cost for adding a new contract template to the application, it has the advantage of using contract templates added by other companies.

At this stage, the e-contract management system comes into play. The CRLs are able to create contracts using the existing templates, make them available to everyone who has a gmail account and has shown interest in the application by logging in and providing their own contractual clauses. The system also manages contract monitoring and execution. If by monitoring, one should understand contract observation with no intervention from the user (e.g. finding out which contracts expired), by execution one should understand that there is the need to specify the actions that were taken in order to obtain feedback, based on rules defined in the agreement.

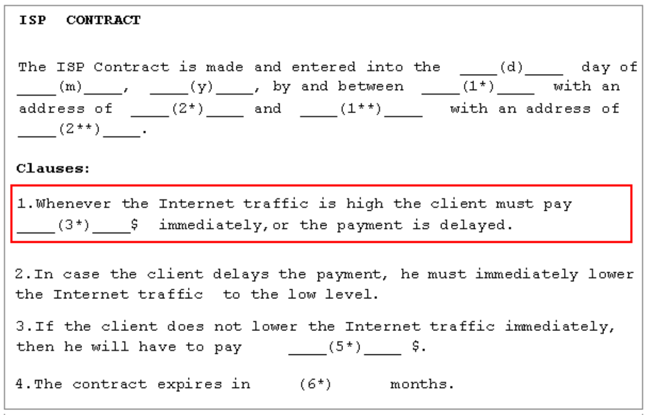
By making possible to upload a contract on Google Docs, the user benefits of Google's services features that are more and more complex. Moreover, the application offers a way to manage contracts on Google Docs such that the task of finding contracts in an archive based on user preferences reduces the time cost whether the companies are interested in all the contracts signed with a customer that proved himself untrusted or wanting to know which contracts expired.

For the validity of contracts it is not required the prior consent of parties on the electronic means. This enforces the use of e-contracts, mostly when it has the form of an agreement, where the contract is not necessarily consulted on establishing its structure which has an impact on time efficiency, both parties being assumed to accept only contracts that implement their interest [2].

II. Technical instrumentation

Google Docs: The Google Documents List Data API allows client applications to manipulate general operating system file management tasks including creation, deletion, download, upload, modifying permissions, via HTTP requests. For this project, Java client library was used to ensure that compatibility with the other technologies is achieved. Various authentication methods were available but the implemented system employs ClientLogin as the authentication method which was chosen for simplicity, since the main focus when using Google Docs API was having access to the web services.

Rule-Based Reasoning: A possible solution for representing contracts was to use deontic logic, with the clauses expressed as obligations and permissions. Another option was to specify the logic behind a contract with procedural coding. Contract clauses are mainly "if" statements, that involve significant conditional branching or decision-making. An approach which uses procedural programming may be time inefficient. However, none of these options were selected. Instead, a fast rule based engine, called Jess, which comes with an API for integration with Java, was chosen to represent the contract clauses as rules. Jess uses the Rete algorithm



ISP CONTRACT

The ISP Contract is made and entered into the ____(d)____ day of ____(m)____, ____(y)____, by and between ____(1*)____ with an address of ____(2*)____ and ____(1**)____ with an address of ____(2**)____.

Clauses:

1.Whenever the Internet traffic is high the client must pay ____(3*)____$ immediately,or the payment is delayed.

2.In case the client delays the payment, he must immediately lower the Internet traffic to the low level.

3.If the client does not lower the Internet traffic immediately, then he will have to pay ____(5*)____ $.

4.The contract expires in (6*) months.

Figure 1. Contract natural language representation.

for deciding which rule to execute, thus its impact on efficiency is more visible as the number of rules increases.

III. Contract Representation

A. Natural Language Representation

Figure 1 shows the way a contract is represented in natural language, so that it can be parsed and formalized. The templates are loaded from the server file system, where they are kept in word processor format. As stated from the beginning, every template only involves two parties. Looking at figure 1, the fields to be completed are of three types: the one to be completed by the first party marked by one asterisk, the ones to be completed by the second party marhed by two asterisks and the ones to be completed automatically by the system(e.g. the date the contract was signed; expiration date, as sum of sign date and months in which the contract will expire). The clauses of the contract are rules, making it easy to represent them in Jess.

B. Formal Language Representation

Figure 2 shows a formal representation of the first statement in figure 1. In order to obtain the desired behaviour and to have a structured view of the Jess code, templates, rules and functions were used. The contract represents an internet service provider agreement and it is characterized by two variables, namely the price the client must pay when the internet traffic is high and the price the client must pay when he delays the payment and does not lower the internet traffic. The expiration date variable is common to all contracts. Thus, an isp contract is defined by these two variables, namely traffic-high-price and delay-no-low-level-price which have to be instantiated so that the execution of the contract can be started. As a result of these observations, two templates have been defined in Jess: isp-contract and start-contract. Once a contract has been defined, the

```
(deftemplate isp_contract
   (slot traffic-high-price)
   (slot delay-no-low-level-price))

(deftemplate start-contract
  (slot traffic-high-price)
  (slot delay-no-low-level-price))

(defrule define-isp-contract
  (isp_contract
    (traffic-high-price ?t)
    (delay-no-low-level-price ?d))
  =>
  (assert (start-contract (traffic-high-price ?t)
  (delay-no-low-level-price ?d))))

(defrule start-contract
  (traffic-high-price ?t)
  (delay-no-low-level-price ?d)
  =>
  (assert (first-question ?t ?d)))

(defrule ask-status-traffic
  (first-question ?t ?d)
  =>
  (printout t "<br>Status of the Internet traffic
    <br>1.high
    <br>2.low"))

(defrule read-answer
  ?fact-id <- (read-answer ?a ?t ?d)
  =>
  (read-last-answer (read t) ?a ?t ?d)
  (retract ?fact-id))
```

Figure 2. Contract formal language representation.

contract can be started as suggested by the rules define-isp-contract and start-contract.

A contract is executed in the form of an interview between the system and the creator of the contract, in order for the automatic engine to obtain information that has an impact on the conclusion, based on the acceptor's actions. Thus, every contract begins with a question addressed by the system to the creator of the agreement, which explains the assertion of the fact first-question. Based on the last answered question and on the given answer, the user obtains feedback according to the rules of the contract through the function read-last-answer and the corresponding feedback function.

Another consequence is the assertion of a fact that makes the rule defined for the next question to fire. (e.g. (assert paid-for-traffic-high-price ?traffic-price ?delay)). Although the answer is read from the Jess terminal, the Jess API provides a way to add an output router, so that the answer can be read from any storage which in this case is a file. This allows the interaction of the user with the rule based system through a graphical interface. The process is automated by storing the question id (e.g. traffic-is-question) in the session object, in the moment it was addressed, for further use. The series of steps: ask question; read answer; give feedback based on the asked

```
(deffuntion read-answered (?answer ?answeredquestion
                           ?traffic-price ?delay)
 (if (?answer equals "traffic-is-question") then
   (traffic-is-feedback ?answer ?traffic-price ?delay))
 (if (?answer equals "paid-for-traffic-is-question")
   then (paid-for-traffic-high-feedback ?answer
    ?traffic-price ?delay))
 (if (?answer equals "lowered_traffic-question") then
   (lowered-traffic-feedback ?answer ?traffic-price ?delay))
 (if (?answer equals "paid-no-lowered-traffic-is-question")
   then (paid-no-lowered-traffic-feedback ?answer
          ?traffic-price ?delay)))

(deffunction traffic-is-feedback
 (?answer ?traffic-price ?delay)
  (if (eq ?answer 1)
    then
     (printout t "<br>The Internet traffic is high.
      The client must pay $" ?traffic-price)
     (assert (paid-for-traffic-high-question
?traffic-price ?delay))
    else (printout t "<br>The Internet traffic is low.
  Nothing to be done")))
```

Figure 3. Contract representation functions.

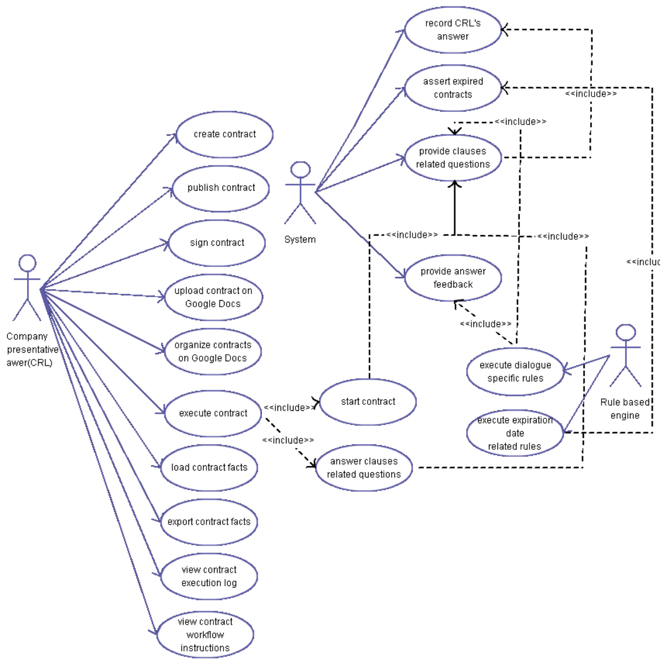


Figure 4. Use case model.

question and the given answer; ask next question are repeated until the contract reaches an execution stage where no more actions have impact on the final state.

IV. System Design

A. Use case model

Figure 4 presents a use case diagram in order to capture system requirements, having the role to give an idea on what architecture design option to choose. The CRL represents the end user of the system. Before managing the contract, the CRL needs to login in the application. He may take one of the two roles: first party or second party. This does not imply that for every contract he represents the first party or the second one, it means that he will be assigned a role for each created contract. While the first party is responsible for creating the contract and publishing it, the second party is responsible for signing an available contract.

After the contract has been signed, only the creator of the contract can begin execution of the contract, which consists of a dialogue between the user and the system, aiming at collecting the actions performed by the contractees. So contract execution, at this step, is done only from first party's perspective.

When execution is finished, the contract creator has the chance to export the facts that result from the dialogue between him and the system. However, both parties have the chance to evaluate the results of second party's actions, by uploading a set of facts specific to the second party. At each step, the user is guided by information present in the application about what needs to be done in that phase of contract management. Signing is a precondition for uploading to Google Docs. The contracts are uploaded in a predefined contracts folder, meant to be used as storage for contracts. After the contracts have been uploaded, the CRL has the chance to organize the documents in that folder, by contract type, parties involved or expiration date.

The System actor, represents the application as an independent component with respective to the rule based engine. It has been separated from the rule based engine actor, since contract rules should be executed from the application or independent of it,making use of rule based engine services. The actor is responsible for monitoring the expiration date of signed contracts. Another responsibility is to provide the creator of the contract with questions that have the form "Did the client pay for high internet traffic[y/n]". After the CRL provides the answer, the system records it, for logging information.

When a contract is exported, the facts contained in it, are based on these answers. The next question asked is based on the answer of the previous question. Moreover, the system provides feedback as a result of taking the previous action. The rule based engine is responsible for executing contract related rules. Based on the facts that result from CRL-system dialogue, or from the ones imported by CRL, the rule based engine decides what actions need to be taken further. Another use case assigned to this actor, is to find out what contracts have expired. This is done based on contract content.

B. *Conceptual architecture*

Figure 5 shows the conceptual architecture diagram of the system that was built based on use cases specification. The user must be authenticated din order to perform any action related to contract management so a login page is necessary. After the user is authenticated, he should be redirected to his personal page, named company representative lawyer's page. For the system to link the events generated by the user when interacting with the graphical user interface and the data related to individual contracts, kept either in databases or in files, a controller is used as an intermediate component between front end and data model. The controller calls services from the business logic layer which encompasses all the processing needed for contract management from a functional point of view (e.g.: create contract, publish contract, sign contract).

The business logic layer includes functionality offered by two major components: the Google Docs connector and the rule based engine. The Google Docs connector is needed for "upload contract on Google Docs" and "organize contracts on Google Docs" use cases. The information related to existing contracts collected when the user creates or signs a contract and other data useful for contract management is kept in a database. For performing records updates, inserts, deletions or retrieval, a database access layer is used. As stated by the "create contract" use case, an agreement will be created based on a template, so there should be a place where the templates are kept. This motivates the use of a contract template storage. Any rule based engine has as a component a specific type of knowledge base, namely a list of rules. This list also should be kept in a contract rules storage.

V. Contracts organization on Google Docs

In [3], the authors described a system for retrieving files in an efficient way, by using organized relationships between the components of the file system. The solution presented is based on user-defined metadata for both files and folders stored in a relational database in the form of key-value pairs, that can be used for querying the file system. Apart from efficiency, it improves the practical aspect of the file systems by offering advanced search criteria, since the operating systems offers only basic operations for data organization at user level.

Starting form this idea we propose here to organize the uploaded documents on Google Docs by each individual user of the electronic contract system. Some file system concepts are redefined in [3]. The only physical containers in the system are defined as *categories*. A property is associated to a category, while a value is associated to a file or a subcategory. Each category contains several properties and each file or subcategory belonging to a

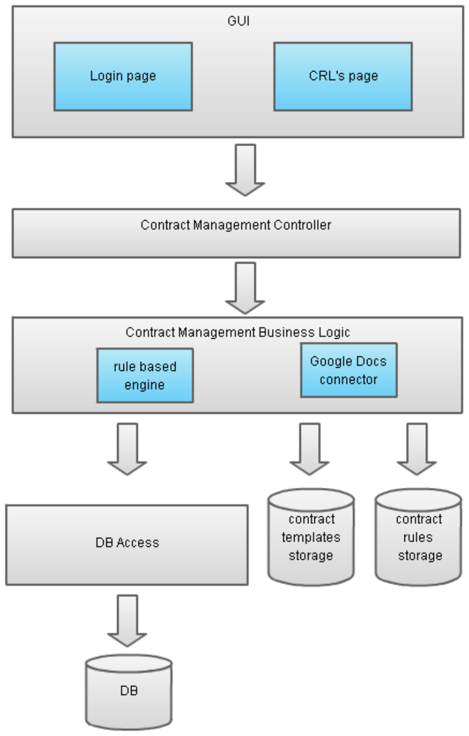


Figure 5. System architecture.

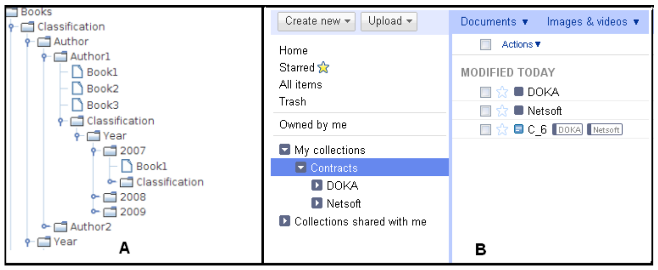


Figure 6. Google Docs organization of contracts.

category, may or may not assign a value to any property associated with that category. Another term that was introduced is the *classification directory* which separates a folder into classified and non-classified view.

Classification can be done at any level of the file hierarchy, without the components from a higher level being affected by the classification. At the higher level, the classification folder contains all the properties that were defined for that category and all the values that were not associated to any of those properties, files that will not appear in a further classification. At the next level, each property contains directories corresponding to the values each of them may take. A value folder contains all files with which it is associated and also, another classification directory.

Figure 6 shows the folder organization after classification was applied. In the left part of the figure, it can be seen that the category on which the search is done is called "Books". A classification directory is created once the search by property "Author" is done. In the next step, virtual containers corresponding to the values associated to properties are created, namely "Author1" and "Author2". Finally, a classification is applied again at the level of directory "Author1". All this process is done using the information in the metadata of the files.

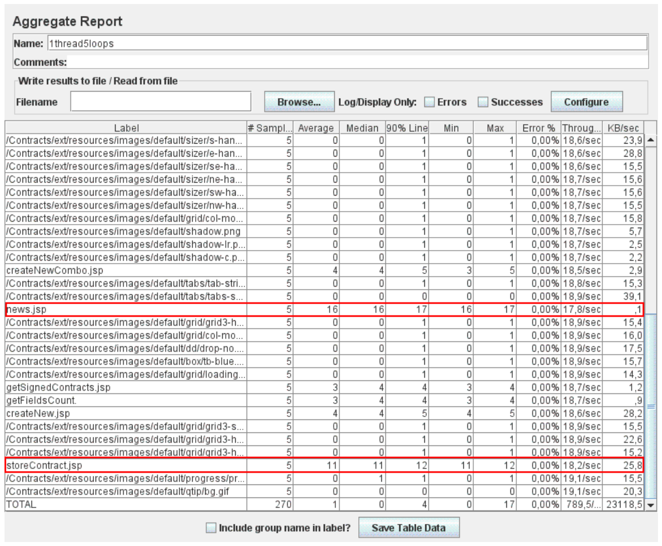
In the right part of figure 6, there is also a single category called "Contracts". It can be seen that no classification directory is created, the reason being the fact that classification can be applied only at the level of the main category. The creation of the folders corresponding to properties is also omitted, since this involves unnecessary calls to Google Docs web services, resulting in time costs. Instead, only the folders corresponding to the values are built. Thus, when applying from the application a "By Parties" filter, all the contracts in the category are moved into folders that have as name one of the corresponding party name. In case of using "By Parties" filter, you would expect duplication of contracts: for example, contract "CX" should be placed both in a folder called "Party1" and in a folder called "Party2". This is true but only from a visual perspective. From a storage perspective, this does not represent a problem, since for every file, a list of pointers to folders it should be visible in is maintained.

VI. Testing the Application

Testing the application and finding bugs as soon as possible is an important factor in avoiding issues that may become more complex in the future. Finding problems in early steps are easier to track so this brings benefits in terms of time resources as well. This is the main role of the continuous integration environment: find what was broken after every commit by running automated tests. A continuous integration environment, based on the interaction between Jenkins, Ant, Jmeter, Selenium, Glassfish web server and SVN server had the mission to trigger a build every night and run the existing scripts, removing the burden to run tests manually. Finally it published the test reports generated by Ant.

A test plan was designed for creating a contract. For each instance of the test plan, the database stored 10 private contracts, 10 available contracts and 10 public contracts. Each time a new instance of the test plan ran, the database was first restored. By looking at the aggregate report in 7 generated by Jmeter, it can be seen that storing the contract takes the longest time, and has the lowest throughput. Here is where errors first appear when increasing the number of users or the loop count.

The application allows contracts to be created by 10 users and 100 loops, without error. Figure 8 shows the



Name: 1thread5loops

Comments:

Label	# Sampl...	Average	Median	90% Line	Min	Max	Error %	Throug...	KB/sec
/Contracts/ext/resources/images/default/sizer/s-han...	5	0	0	1	0	1	0,00%	18,6/sec	23,9
/Contracts/ext/resources/images/default/sizer/e-han...	5	0	0	1	0	1	0,00%	18,6/sec	28,8
/Contracts/ext/resources/images/default/sizer/se-ha...	5	0	0	1	0	1	0,00%	18,6/sec	15,5
/Contracts/ext/resources/images/default/sizer/ne-ha...	5	0	0	1	0	1	0,00%	18,7/sec	15,6
/Contracts/ext/resources/images/default/sizer/sw-ha...	5	0	0	1	0	1	0,00%	18,7/sec	15,6
/Contracts/ext/resources/images/default/sizer/nw-ha...	5	0	0	1	0	1	0,00%	18,7/sec	15,5
/Contracts/ext/resources/images/default/grid/col-mo...	5	0	0	1	0	1	0,00%	18,7/sec	15,8
/Contracts/ext/resources/images/default/shadow.png	5	0	0	1	0	1	0,00%	18,7/sec	5,7
/Contracts/ext/resources/images/default/shadow-lr.p...	5	0	0	1	0	1	0,00%	18,7/sec	2,5
/Contracts/ext/resources/images/default/shadow-c.p...	5	0	0	1	0	1	0,00%	18,7/sec	2,2
createNewCombo.jsp	5	4	4	5	3	5	0,00%	18,5/sec	2,9
/Contracts/ext/resources/images/default/tabs/tab-stri...	5	0	0	1	0	1	0,00%	18,8/sec	15,3
/Contracts/ext/resources/images/default/tabs/tabs-s...	5	0	0	0	0	0	0,00%	18,9/sec	39,1
news.jsp	5	16	16	17	16	17	0,00%	17,8/sec	,1
/Contracts/ext/resources/images/default/grid/grid3-h...	5	0	0	1	0	1	0,00%	18,9/sec	15,4
/Contracts/ext/resources/images/default/grid/col-mo...	5	0	0	1	0	1	0,00%	18,9/sec	16,0
/Contracts/ext/resources/images/default/dd/drop-no....	5	0	0	1	0	1	0,00%	18,9/sec	17,5
/Contracts/ext/resources/images/default/box/tb-blue....	5	0	0	1	0	1	0,00%	18,9/sec	15,7
/Contracts/ext/resources/images/default/grid/loading...	5	0	0	1	0	1	0,00%	18,9/sec	14,3
getSignedContracts.jsp	5	3	4	4	3	4	0,00%	18,7/sec	1,2
getFieldsCount.	5	3	4	4	3	4	0,00%	18,7/sec	,9
createNew.jsp	5	4	4	5	4	5	0,00%	18,6/sec	28,2
/Contracts/ext/resources/images/default/grid/grid3-s...	5	0	0	1	0	1	0,00%	18,9/sec	15,5
/Contracts/ext/resources/images/default/grid/grid3-h...	5	0	0	1	0	1	0,00%	18,9/sec	22,6
/Contracts/ext/resources/images/default/grid/grid3-h...	5	0	0	1	0	1	0,00%	18,9/sec	15,2
storeContract.jsp	5	11	11	12	11	12	0,00%	18,2/sec	25,8
/Contracts/ext/resources/images/default/progress/pr...	5	0	1	1	0	1	0,00%	19,1/sec	15,5
/Contracts/ext/resources/images/default/qtip/bg.gif	5	0	0	0	0	0	0,00%	19,1/sec	20,3
TOTAL	270	1	0	4	0	17	0,00%	789,5/...	23118,5

Figure 7. Jmeter aggregate report.

way the average, median, min, max,error and throughput change when the number of users increases, while the number of loops remains constant: 100, meaning that 10 users create concurrently 100 contracts with a ramp-up period of 1 second, so that in an interval of 1 second, all users are active.

VII. Discussion and Related Work

While in the approach described in [3] file metadata is used for querying the system, the solution for file management used by the current application uses file content. The second approach enforces the use of conventions for the structure of a contract. Once a contract is uploaded on Google Docs, a header containg key-value pairs is added to the document(E.g. Contract type: Confidentiality, Expiration date: 01.04.2012). These two approaches cannot be compared in terms of efficiency, since for the developed contract management tool each operation requires a call to a Google Docs web service, but they have similar goals of improving data organisation at user level.

Formalising diferent types of real contracts also appears in [4]. The representation language is given by the semantics and expressivity of higher order social commitments. By composing these social commmitments contracts like: gratuitous promises, unilateral contracts, bilateral contracts, and forward contracts. In our case, we formalise a set common patterns of contractual clauses. By activating a subset of these clauses we have formalised known types of contracts like ISP. The user has the possibility to create his new type of agreement, by instantiating different clauses.

DR-CONTRACT management system [5] exploits defeasible reasoning to handle new contradictory infor-

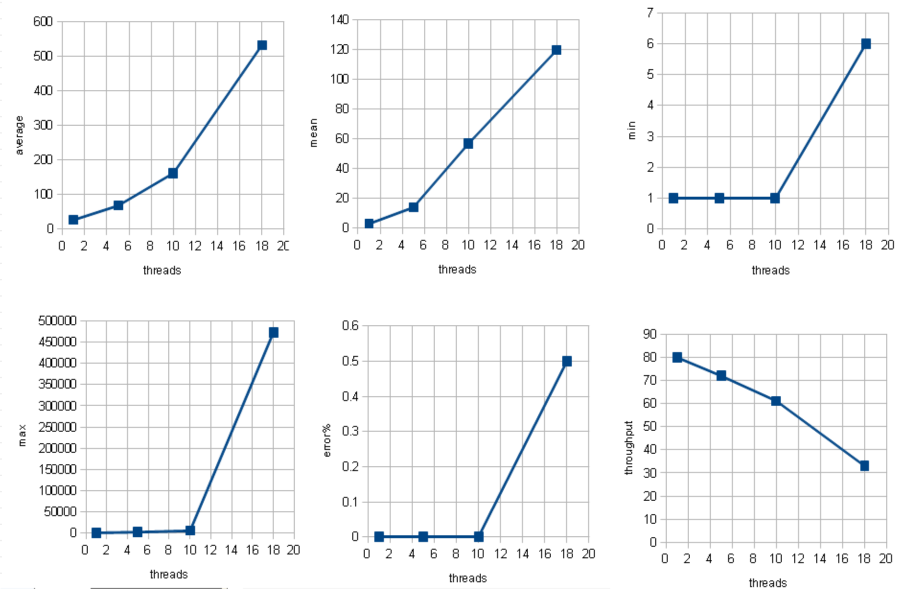


Figure 8. Jmeter performance report.

mation that can occur durig a contract lifecycle. The RuleML [6] is extended to deal with the monitoring the contracts. We focus here on facilitating the contract organisation and online accesibility based on user preference, in the context of WebOS [7].

VIII. Conclusion

This paper introduces a contract management system able to handle the operations needed for the life-cycle of an agreement: creation, making the contract available to the customers, signing the agreement, monitoring and execution. The method that enables formal representation of clauses and the solution came through Jess rules. The contract terms were automatically identified by employing the Jess rule based system.

The main contribution of this paper is the mechanism for dynamic re-organization of contracts on Google Docs based on user preferences, by exploiting the existing file system properties. The practical use and the quality of the e-contract management system was ensured by testing its functionality and performance in a continuous integration environment. On going work regards the integration of legal ontologies aiming to facilitate the understanding of contractual clauses, when designing or signing new contracts. This would be a first step towards an online dispute resolution system which applies legal reasoning on formal representation of contracts.

Acknowledgement

We are grateful to the anonymous reviewers for their useful comments. The work has been co-funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557 and PN-II-Ideas-170.

References

[1] R. F. Lusch, S. L. Vargo, and G. Wessels, “Toward a conceptual foundation for service science: contributions from service-dominant logic,” *IBM Syst. J.*, vol. 47, pp. 5–13, January 2008.

[2] S. Becher and T. Zarsky, “E-contract doctrine 2.0: Standard form contracting in the age of online user participation,” *Mich. Telecomm. Tech. L. Rev*, 2008.

[3] A. Coldea, A. Colesa, and I. Igant, “Orcfs: Organized relationships between components of the file system for efficient file retrieval,” in *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2010*, 2010, pp. 434–441.

[4] I. A. Letia and A. Groza, “Agreeing on defeasible commitments,” in *DALT*, ser. LNCS, M. Baldoni and U. Endriss, Eds., vol. 4327. Springer, 2006, pp. 156–173.

[5] G. Governatori and D. Hoang, “A semantic web based architecture for e-contracts in defeasible logic,” in *Rules and Rule Markup Languages for the Semantic Web*, ser. LNCS, A. Adi, S. Stoutenburg, and S. Tabet, Eds. Springer Berlin Heidelberg, 2005, vol. 3791, pp. 145–159.

[6] A. Rotolo, “Rule-based agents, compliance, and intention reconsideration in defeasible logic,” in *RuleML Europe*, ser. LNCS, N. Bassiliades, G. Governatori, and A. Paschke, Eds., vol. 6826. Springer, 2011, pp. 67–82.

[7] A. Weiss, “Webos: say goodbye to desktop applications,” *netWorker*, vol. 9, pp. 18–26, December 2005.