

Automatic Synchronization of Common Parameters in Configuration Files

Moupojou Matango Emmanuel¹, Moukouop Nguena Ibrahim²

¹Department of Computer Science, University of Yaounde 1, Yaounde, Cameroon

²National Advanced School of Engineering, Department of Computer Science, University of Yaounde 1, Yaounde, Cameroon

Email: moupojouemma@yahoo.fr, imoukouo@gmail.com

Received 6 January 2015; accepted 16 April 2015; published 17 April 2015

Copyright © 2015 by authors and Scientific Research Publishing Inc.

This work is licensed under the Creative Commons Attribution International License (CC BY).

<http://creativecommons.org/licenses/by/4.0/>



Open Access

Abstract

In an information system, applications often make use of services that they access using the parameters described in their configuration files. Various applications then use different codes to denote the same parameters. When access parameters of a service are modified, it is necessary to update them in every configuration file using them. These changes are necessary, for example because of security policies involving regular changes of passwords, or departure of some system administrators. The database password could be changed for example. When system administrators can not immediately identify all services affected by a change or when they feel they don't have the skills to edit these files, these parameters remain unchanged, creating critical security flaws. This was observed in more than 80% of the organizations we studied. It then becomes necessary to ensure automatic synchronization of all affected files when changing certain settings. Conventional synchronization solutions are difficult to apply when the relevant applications have already been developed by third parties. In this paper, we propose and implement a solution to automatically update all configuration files affected by a change, respecting their structure and codification. It combines a parameters database, a mapping between the configuration files parameters codes and those of the database, and templates for the generation of files. It achieves the objective for all non-encrypted configuration files.

Keywords

System Administration, Configuration File, Parameter, Update, Synchronization

1. Introduction

A configuration file is a file that contains configuration information used by a computer program to adapt or

How to cite this paper: Emmanuel, M.M. and Ibrahim, M.N. (2015) Automatic Synchronization of Common Parameters in Configuration Files. *Journal of Software Engineering and Applications*, 8, 192-200.

<http://dx.doi.org/10.4236/jsea.2015.84020>

customize its operation. File synchronization (or syncing) in computing is the process of ensuring that computer files in two or more locations are updated via certain rules.

The password change of a critical service (database, email...) that can occur repeatedly (according to security policies), or following the departure of a system administrator is a regular event. Such a change may involve changing parameters of dozens of applications using this service. Although in theory all services affected by a change must be documented, in practice, not only this is not always the case, but in addition to that, many administrators are afraid to make changes in configuration files for applications they do not have control. These manual changes may be associated with errors or false manipulations.

To better understand the problem exposed here, consider the following example which is a usual situation in production environments: Let us consider a set of five services (or applications), each with its particular configuration file that contains, among others, the access parameters to the same database. Those common parameters have different names from one file to the other. If the administrator changes the access parameters to the database, then it will require to also change them in all five configuration files, without forgetting one (if he doesn't use a traceability matrix¹), without mistake, otherwise some services will then cease running: which turns out being a tedious task. It would then be wise to manage those common settings on a unified way. The two-way file synchronization is then required², to ensure that when a parameter in a file undergoes an update, this update is propagated to all other parameters files having a parameter supposed to contain the same value.

The MDAL³ applications have for example the following configuration files, **Figure 1** and **Figure 2**, used by different services:

These two configuration files used by different services, have several common configuration settings that should have the same value at any time for the system remains in a consistent state of configuration (dbuser, dbpassword, applipassword...). Also note that those common parameters might have different names from one file to the other, which could in turn be in different formats as we will see in the implementation example in-Subsection 3.2.

2. Materials and Methods

2.1. State of the Art on File Synchronization

2.1.1. Whole Files Synchronization

This approach considers that files at different locations are identical, and shall have exactly the same content when an update is made upon one of them. Many whole file synchronization techniques and their applications are described [1]-[3], and many tools performing this kind of synchronization are available [4]. Andrew Tridgell proposed an efficient algorithm called rsync for performing this synchronization [5] [6]. Many rsync enhancements and optimizations were also proposed [7] [8]. But rather than synchronizing the whole configuration file, we shall synchronize only some common parameters inside those files, because of the fact that the files are different.

2.1.2. Synchronizing Particular Elements in Files

This is the purpose of this paper. Some elements in configuration files shall be synchronized with other elements in other files in order to keep the consistency of the system. The parameters are semantically equivalent, but can

```
url=jdbc:mysql://emmanuel-PC:3309/evaluation
dbuser=root
dbpassword=manager10
dbdriver=com.mysql.jdbc.Driver
appliuser=system
applipassword=mongui
```

Figure 1. Configuration file mailerdaemonparams.txt of MDAL applications.

¹Document that correlates any two base-lined documents that require a many-to-many relationship to determine the completeness of the relationship.

²Updated files are copied in both directions, usually with the purpose of keeping the two locations identical to each other.

³Megasoft Data Access Library: written in Java Framework, facilitating the development, deployment, operation and maintenance of applications accessing databases.

```

url=jdbc:mysql://emmanuel-PC:3309/bdgo
dbuser=root
dbpassword=manager10
dbdriver=com.mysql.jdbc.Driver
appliuser=system
applipassword=mongui
folder=D:\Program Files\MDAL\srcbackyp
destinationfolder=D:\Program Files\MDAL\srcbackypmail
sizelimit=6000000
backupid=backup_
compte=1000000074
expediteur=imoukouo@gmail.com
destinataire=imoukouo@gmail.com
repeat_interval=120

```

Figure 2. Configuration file folder mailerparams.txt of MDAL applications.

have completely different codes from one file to another.

Existing approaches to do this are:

1. Centralized configuration settings

The principle here is to have a single file containing all the parameters of all applications/services.

The various applications read their functional parameters in this file; and when an update is made, the new value is immediately visible to all other applications. Hyena⁴, for example, allows centralized management of common configuration settings to multiple services by bringing them together in a single file [9].

2. Distributed configuration files with inclusions

This approach is used to preserve the distributed aspect of configuration files; each service having its own configuration file, but containing only its private parameters. Common settings among several configuration files are placed in a common file, and individual files wishing to include them in their settings then have, in their structures, inclusion directives pointing to that or those common files. Thus, when loading the parameters of a file, they are also added, and so recursively, the parameters contained in the files to which it points. Thus, the update of these parameters is only done in the common file and is thus taken into account in the different files including it.

a) The JNLP Protocol⁵ uses this technique when loading .Jnlp files for downloading and running Java applications upon the network. Indeed, a .jnlp file may, in its structure, have a link to another .jnlp file; and when loading the original file resources (different .jar files), the JNLP protocol also loads all the resources listed in the referenced file.

b) The “MS CF Manager”⁶ application, developed during this project, also uses this technique.

Besides ordinary settings, a configuration file can also have the “mdal.include” parameter that contains the different paths, separated by semicolons, of other configuration files whose parameters should be added to those of the original one.

The two approaches mentioned above are valid only if they were taken into account during the development phase of the application. Care is then taken to define how to access configuration files, taking into account their structures. Now considering the case applications/services are already developed, and that the synchronization of different configuration settings shall be done as described above, the problem becomes very different. Anne Jonassen Hass describes the architecture and processes of a configuration manager [10] without addressing our specific problem, which is how to ensure common parameters synchronization among many configuration files. Bob Aiello and Leslie Sachs were able to bridge the language gap between the myriad of communities involved with successful Configuration Management implementations [10]. They describe practical, real world practices that can be implemented by developers, managers, standard makers, and even Classical CM Folk.

2.2. New Configuration files Synchronization Approach and Model

2.2.1. Identification of the Solution

Configuration Management is a multidisciplinary science (computer science, aeronautics, automobile, wea-

⁴Software to simplify and centralize nearly all daily management tasks of Windows network, Active Directory management and adding new capabilities to existing native tools.

⁵Java Network Launching Protocol.

⁶MegaSoft Configuration Files Manager.

pons...) consisting in the management of the technical description of a system and its various components, as well as change management made during the evolution of that system. The diagram showing the general structure and operation of a configurations manager is shown in **Figure 3** [10]:

Used in monitoring different versions of documents in computer science, it allows archive and tracking of all changes that have occurred in these documents. Relying on this science for which a standard exists, it will issue to propose and implement a model of configuration files synchronization for solving the problem. This synchronization should be possible even for existing applications.

Configuration management consists in four main modules:

- **Unique Identification:** It allows determining a configuration item’s metadata, to uniquely identify it, and to specify its relations with the outside world and other configuration elements.
- **Storage:** This module ensures that a configuration item will not disappear, or will not be damaged, but it can be found and made available in the desired state.
- **Change Control:** This module is fully in control of all changes requests to the system, and all implemented changes.

In the case of our problem, this automatically implies an automatic traceability, which is not often done when the administrator directly modifies the configuration files, and this often causes difficulties to return after unfortunate changes.

- **Status Reporting:** It allows providing legible and useful information to ensure effective management of the development and maintenance of a product.

In the context of the problem to solve, change control will be suitable. Indeed, it is usually a modification proposal made by the end user, in the case where configuration management has been applied to the development of an application for example. The problem is then found in a more specific area called “Version Control”. Changes implemented here therefore simply consist in the various updates of different values of the configura-

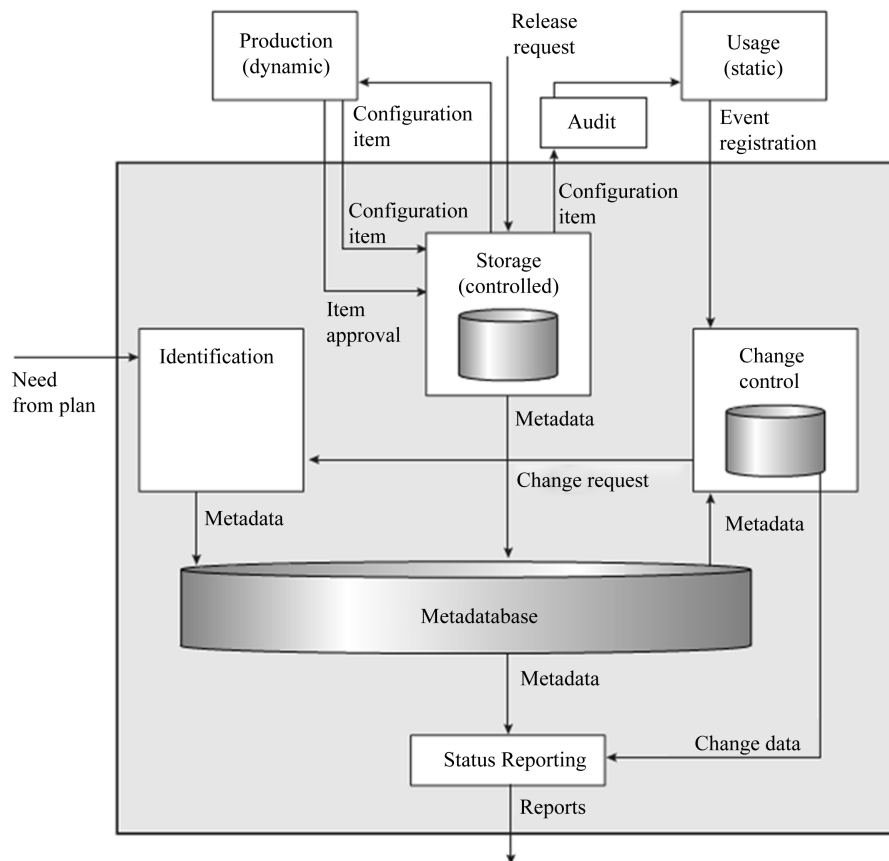


Figure 3. Operation of a configuration manager [10].

tion parameters. The various elements under the control of the configuration manager, which corresponds to the configuration settings in this case, shall in no case be modified, but new versions will be created: this should allow returning to a previous configuration state. It is therefore necessary to make a full scan of the system to establish, in taking into account the specificity of configuration managers architecture, and proceed subsequently to an implementation of this model.

2.2.2. Modeling Solution

Generally, the configuration files synchronization problem may be represented by **Figure 4**.

With:

- A_i : Application i
- A_iS_j : Service j of the application i
- $A_iS_jF_k$: Configuration File k of service j of application i
- P_{ijkl} : Parameter l of the configuration file k of service j of application i
- V_{ijkl} : Value of parameter P_{ijkl}
- F_{ijk} : Type (format) of the configuration file k of the service j of application i
- Rel 1: Parameter P_{111n} corresponds to Parameters P_{1121} and P_{1pm}
- Rel 2: Parameter P_{1pr1} corresponds to Parameter P_{n11n}

The problem is the following: at any time the parameters $(P_{111n}, P_{1121}, P_{1pm})$ a part, and (P_{1pr1}, P_{n11n}) on the other, must have the same value so that settings of applications A_1 and A_n remain consistent. The equalities $V_{111n} = V_{1121} = V_{1pm}$ and $V_{1pr1} = V_{n11n}$ should therefore always be checked. If a parameter is changed, its correspondents must also be changed in cascade.

Let P be the set of parameters under the control of the configuration files synchronization module. We assume that every parameter is inside a group of parameters, and every group has at least one element, the parameter itself. The following functions can be defined:

1. correspondence: $P \times P \rightarrow \text{Boolean}$
 $\text{correspondence}(p,q) \equiv p.\text{group} = q.\text{group}$ ⁷
2. update: $P \times \text{Char} \rightarrow \text{Unit}$
 $\text{update}(p,\text{val}) \equiv \forall q \in P \text{ if } \text{correspondence}(p,q) \text{ then } q.\text{value} := \text{val}$

The realization of this solution is based on the use of templates. A template is a file having the same structure as a configuration file, but wherein parameters values are replaced by their identifiers in the system. Thus, to add

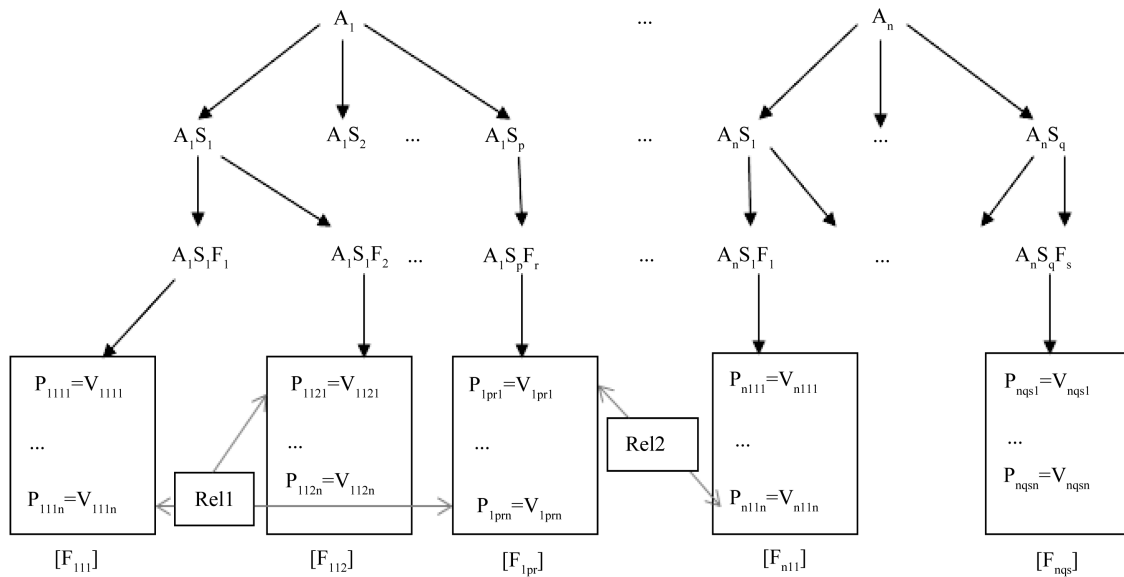


Figure 4. Configuration files synchronization problem.

⁷Two parameters are in correspondence when they contain exactly the same information. So at any time they must have the same value. This is our traceability matrix.

or delete a configuration parameter, this will be from the template. The analysis of the system led to the following relational schema for entities to use to implement the solution:

- application (code, label, description)
- template (code, name, filepath⁸, description)
- application_template (application_code, template_code)
- parametersgroup (code, groupname, description)
- parameter (code, identifier⁹, parametersgroup_code, parameter_type, begin_date, end_date)
- template parameter (template_code, parameter_code)
- value (code, value, creationhour)
- parametersgroup_value (parametersgroup_code, value_code)

When a parameter p is updated, the following operations are performed:

1. A new value is created for the parameters group containing p, and saved.

The following operations are repeated for all configuration parameters q in the same group with p.

2. A copy of q's template content is made.

3. In that copy, all parameters identifiers are replaced by their last saved values in the system.

4. The result then overwrites the template's target file, which is the original configuration file.

The procedure for updating a configuration parameter is as follows:

Algorithm 1 updateParameter (Parameter: p, String: newValue)

*/*p: parameter that needs to be updated; newValue: new value of the parameter */*

1: p.getGroup().setLastValue(Value (newValue));

/ creating a new value and adding it in the list of parameter's group values */*

/ Updating all parameters in the same group with p; p.getGroupedParameters() returning at least {p} */*

2: **for all** par ∈ p.getGroupedParameters()**do**

3: Template template = par.getTemplate();

/ reading the template file containing the parameter */*

4: String file = template.filepath;*/*reading file effectively containing the parameter */*

5: String templateCopy = readFile (template.name);*/* reading template file content */*

6: **for all** param ∈ template.getParameters() **do**

7: templateCopy = templateCopy.replace(param.identifier, param.getGroup().getLastValue());

/ setting values of parameters in the template*/*

8: **end for**

9: writeFile (file, templateCopy) ;*/*updating the target file of the template */*

10: **end for**

3. Results and Example

3.1. Results

As result of the model presented in Subsection 2.2.2, a configuration files synchronization tool was proposed called "Configuration Files Synchronizer" ensuring that correspondent parameters in those files may be synchronized in an easy way. Concretely, the result consists in an application allowing the management of those common settings on a unified way; so that when a parameter undergoes an update, this update is propagated to all other configuration files having a parameter supposed to contain the same value.

To do this, it must be set in the application the configuration files that are to be synchronized, and their respective template files. After this, the different common parameters must be regrouped in appropriate groups. When a parameter has to be updated, the update is not directly done in the configuration file. It is done through

⁸Path to the original configuration file for which this template was created.

⁹Unique identifier of a parameter in the database, which is mapped (same name) to the parameter's value in the template file.

the application, which also updates all the parameters in the same group than the first one, while updating their respective configuration files. An example of this is given in the Subsection 3.2 below. Notice that an implementation of this solution was proposed in Java programming language.

3.2. Implementation Example

We consider two applications A1 and A2, with respective configuration files CF11.txt and CF12.properties for A1, and CF2.xml for A2, as shown on the **Figures 5-7**:

The parameters that are corresponding and that must be synchronized are: appli_name of CF11.txt, and application_name of CF12.properties.

1. Building Configuration Files Templates

Remind that a template has the same structure than its configuration file, the difference being that parameters values are replaced by their identifiers in the system. Proceeding like that, we have the following templates, where **Figures 8-10** are respectively the template files of configuration files represented by **Figures 5-7**:

2. Creating corresponding parameters groups

Now, parameters that are to be synchronized must be placed in the same group, so that, when a parameter will be updated, all parameters in the same group will also be updated, as described in Section 2.2.2. To do this, the following parameter group is created using the equivalent parameters and the template files:

A1 application name (A1_CF11_appli_name, A1_CF12_application_name)

3. Tables content

- application (code, label, description)
 - (1, Application 1, "")
 - (2, Application 2, "")
- template (code, name, filepath , description)
 - (1, CF11_template.txt, {path}\CF11.txt, "")
 - (2, CF12_template.properties, {path}\CF12.properties, "")
 - (3, CF2_template.xml, {path}\CF2.xml, "")
- application_template (application_code, template_code)
 - (1, 1)
 - (1, 2)
 - (2, 3)

```

1 appli_name=A1
2
    
```

Figure 5. CF11.txt.

```

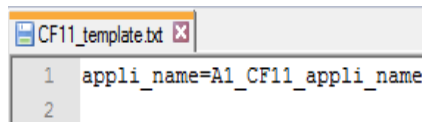
1 application_name=A1
2
    
```

Figure 6. CF12.properties.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/200
3 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persist
4 <persistence-unit name="Application.Persistant.Unit" transaction-type="RESOURCE_LOCAL">
5   <provider>org.hibernate.ejb.HibernatePersistence</provider>
6   <properties>
7     <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
8   </properties>
9 </persistence-unit>
10 </persistence>
11
    
```

Figure 7. CF2.xml.

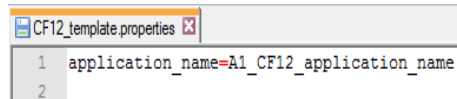


```

1 appli_name=A1_CF11_appli_name
2

```

Figure 8. CF11_template.txt.

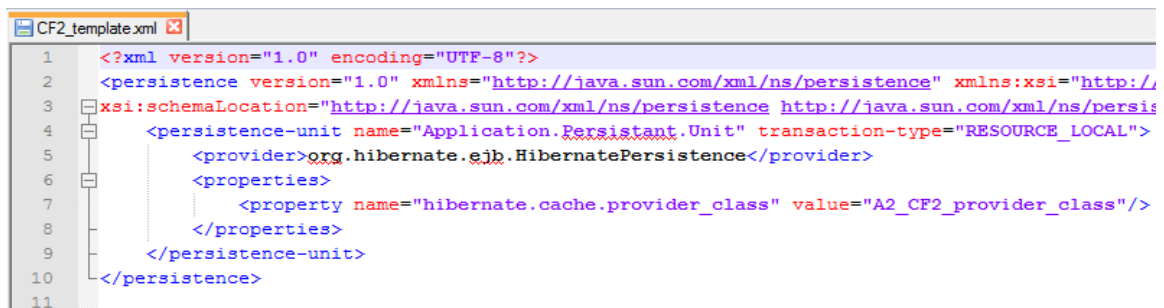


```

1 application_name=A1_CF12_application_name
2

```

Figure 9. CF12_template.properties.



```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://
3 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persi
4 <persistence-unit name="Application.Persistent.Unit" transaction-type="RESOURCE_LOCAL">
5 <provider>org.hibernate.ejb.HibernatePersistence</provider>
6 <properties>
7 <property name="hibernate.cache.provider_class" value="A2_CF2_provider_class"/>
8 </properties>
9 </persistence-unit>
10 </persistence>
11

```

Figure 10. CF2_template.xml.

- parametersgroup (code, groupname, description)
 - (1, A1_application_name, "")
 - (2, A2_CF2_provider_class, "")
- parameter (code, identifier , parametersgroup_code, parametertype, begin_date, end_date)
 - (1, A1_CF11_appli_name, 1, ordinary, 10/02/2015, null)
 - (2, A1_CF12_application_name, 1, ordinary, 10/02/2015, null)
 - (3, A2_CF2_provider_class, 2, ordinary, 10/02/2015, null)
- template_parameter (template_code, parameter_code)
 - (1, 1)
 - (2, 2)
 - (3, 3)
- value (code, value, creationhour)
 - (1, A1, 10/02/2015;07:30)
 - (2, org.hibernate.cache.NoCacheProvider, 10/02/2015;07:30)
- parametersgroup_value (parametersgroup_code, value_code)
 - (1, 1)
 - (2, 2)

4. Conclusion

The multiplicity of configurations files creates a problem of parameters synchronization, when the same parameters are found in different configuration files and with different names. Such parameters are then equivalent and must at all times have the same value, so the update of a parameter must then be propagated to other files in order to maintain the system configuration in a coherent state. Existing approaches to solve this problem, such as centralization or files referencing are limited because they cannot be used for applications already developed. A model for the resolution of this problem (even for existing applications) has been proposed in this paper. Configuration files are represented by their templates, and links are established between equivalent parameters. When a parameter undergoes an update, this update is propagated to all other configuration files having a parameter supposed to contain the same value. This solution results in a very useful tool for system administrators to synchronize configuration files. In perspective, this tool may be improved in order to take into account configura-

tion files security aspects when they are secured by security tools such as mRemote or Secure CRT.

Acknowledgements

We thank the government of Cameroon for premium sought they gave us in funding for our research.

We thank CETIC¹⁰ for funding our research. We thank MEGASOFT SARL Company for allowing us to undertake this work.

References

- [1] Tridgell, A. and MacKerras, P. (2002) The Rsync algorithm. Academic Press, Cambridge.
- [2] Ramsey, N. and Csirmaz, E. (2001) An Algebraic Approach to File Synchronization. *Proceedings of the 9th ACM International Symposium on Foundations of Software Engineering*, 175-185. <http://dx.doi.org/10.1145/503229.503233>
- [3] Balasubramaniam, S. and Pierce, B. (1998) What Is a File Synchronizer? *Proceedings of the ACM/IEEE MOBICOM 98 Conference*, 1998, 98-108.
- [4] Basso, M. and Mann, J. (2013) MarketScope for Enterprise File Synchronization and Sharing. Gartner.
- [5] Tridgell, A. (2000) Efficient Algorithms for Sorting and Synchronization. Ph.D. Dissertation, The Australian National University, Canberra.
- [6] Torsten, S. and Memon, N. (1996) Algorithms for Delta Compression and Remote File Synchronization. Technical Report TR-CS-96-05.
- [7] Rasch, D. and Burns, R. (2003) In-Place Rsync: File Synchronization for Mobile and Wireless Devices. *Proceedings of the USENIX Annual Technical Conference*, 2003.
- [8] Suel, T., Noel, P. and Trendaflov, D. (2004) Improved File Synchronization Techniques for Maintaining Large Replicated Collections over Slow Networks. *Proceedings of the 20th International Conference on Data Engineering*, 30 March-2 April 2004, 153-164. <http://dx.doi.org/10.1109/ICDE.2004.1319992>
- [9] (2014) Hyena. Shared Settings. <http://www.systemtools.com/HyenaHelp/sharedsettings.htm>
- [10] Hass, A.M. (2002) Configuration Management Principles and Practice. DELTA.

¹⁰Centre d'Excellence Africain en Technologies de l'Information et de la Communication.