Spreadsheet Element Sharing through Synchronization

Massimo Maresca
CIPI – Univ. of Genova, Italy and ICSI, Berkeley, CA
massimo.maresca@unige.it

Pierpaolo Baglietto
CIPI – Univ. of Genova, Italy
p.baglietto@cipi.unige.it

***Abstract*—We consider the case of spreadsheet based workgroup collaboration, in which users cooperate through spreadsheets. While spreadsheet exchange through email and shared directories is still the most common way to share data, spreadsheet sharing in the Cloud is rapidly spreading, leveraging the evolution of office tools toward delocalized system over the Cloud. In this paper we observe that spreadsheet sharing through delocalization does not support information confidentiality and introduce the concept of spreadsheet sharing through synchronization, in which both processing and storage maintain their localization in the user systems while a Cloud platform takes care of synchronization. Finally, we focus on Spreadsheet Element Synchronization in which synchronization works at the cell granularity rather than at the file granularity and show some use cases the demonstrate its superiority.**

Keywords—spreadsheets; collaboration; sharing

I. INTRODUCTION

Spreadsheets are probably the best known form of End User Programming [1][2]. The huge number of spreadsheet users and programmers [3] demonstrates the high impact of spreadsheet research and motivates an engineering approach [4]. Recently, spreadsheets have evolved from personal office tools aimed at improving people productivity to enterprise level tools aimed at supporting workgroup collaboration [5] [6] and distributed analytics [7]. In workgroup collaboration, users enter data deriving from local activities, make such data of part of it available to other users, and combine the data obtained from other users with local data. The fact that spreadsheet-based workgroup collaboration has become a dominant paradigm is demonstrated by the common practice of exchanging spreadsheet attachments over email. However, such a practice has not been engineered so far: in particular, users control table exchange/update manually through copy, paste, attach-to-email, extract-from-email operations. Unsupervised data sharing and circulation often leads to errors or, at the very least, to inconsistencies, data losses, and proliferation of multiple copies (the so called "Multiple Versions of the Truth").

To improve spreadsheet-based Workgroup Collaboration, the major IT companies (e.g., Microsoft and Google, but also pbox, Zoho and others) have introduced the sharing paradigm [8][9], which leverages Cloud computing through the Software as a Service paradigm. Unfortunately, such a paradigm does not provide the functionalities that a system supporting workgroup collaboration must provide. In particular, it is not compatible with information confidentiality as processing takes place in Cloud servers, which implies that spreadsheets must reside in such servers in the clear. Enterprises and professional users tend to reject such a solution because they operate on sensitive data.

On the contrary, the Storage as a Service paradigm is compatible with confidentiality, as appropriate encryption systems can insulate the enterprise trusted environment from the external storage service and guarantee that no data in the clear ever exits the trusted environment. However, it does not fully support sharing as the interface exposed by the external Storage Service is typically that of a File Management System and as a consequence only works at the file granularity.

In the paper, we propose a new solution to spreadsheet sharing based on Spreadsheet Synchronization. The proposed solution leaves processing in the user desktop, which is a prerequisite to guarantee confidentiality. It also leaves storage in the user administrative domains and takes advantage of a Cloud server to maintain the user spreadsheets synchronized.The distinctive feature of the proposed solution is that it does not use "file" as an atomic sharing element. On the contrary, it uses a finer grain unit of data called Spreadsheet Element, which corresponds to a cell range, a table, or a worksheet as an atomic storage element.

Spreadsheets interact with each other over a virtual space and share Spreadsheet Elements. Spreadsheet interaction is supported by a spreadsheet platform extension (in particular we have developed a Microsoft Excel Addin) and by a Cloud server.

The paper proceeds as follows. In section II we describe spreadsheet sharing through delocalization, in Section III we introduce Spreadsheet Sharing through Synchronization while in Section IV we focus on Spreadsheet Element Sharing through Synchronization. Then, in Section V we present a structured collaboration framework based on Spreadsheet Element Sharing called Spreadsheet Overlay, in Section VI we present some use cases while in Section VII describe the feedback that we obtained from a set of pilots. Finally, in Section VIII we provide a concluding remark.

II. SPREADSHEET SHARING THROUGH DELOCALIZATION

People traditionally collaborate by exchanging spreadsheets through email or shared directories. The fact that spreadsheet operation, i.e., spreadsheet creation, editing and saving, is decoupled from spreadsheet exchange leads to errors and inconsistencies.

Spreadsheet based collaboration has recently evolved to spreadsheet sharing, which integrates spreadsheet operation and spreadsheet exchange and so reduces errors and inconsistencies. Sharing directly leverages Cloud Computing: to be precise, while the primary goals of Cloud Computing are storage delocalization (Storage as a Service) and processing delocalization (Software as a Service), it is true that a side

benefit of Cloud Computing is the support of shared access to a delocalized resource.

When sharing is based on Storage as a Service (Fig. 1), the data moves to the Cloud while processing remains in the user systems. Such an organization preserves full compatibility with desktop spreadsheet operation, as the Spreadsheet Platform, i.e., the program that operates on spreadsheets (e.g., Microsoft Excel), still runs in the desktop. The Cloud storage exposes a File System interface to the application programs, and in particular to the Spreadsheet Platform, over the network. Sharing leverages the file locking capability typically provided by File Systems.

However, Software as a Service, rather than Storage as a Service, is the most common paradigm for spreadsheet sharing. Its diffusion derives from the fact that both Google Docs and Microsoft Office 365 use it. The evolution from Storage as a Service to Software as a Service can be seen as a two-step process. The first step, shown in Fig. *2*, leads to a system which differs from the Storage as a Service paradigm in the fact that the Spreadsheet Platforms run in the Cloud, rather than in the user desktops and that spreadsheet operation is organized as a Web Application, in which the browser provides the user interface whereas the Spreadsheet Platform provides processing.

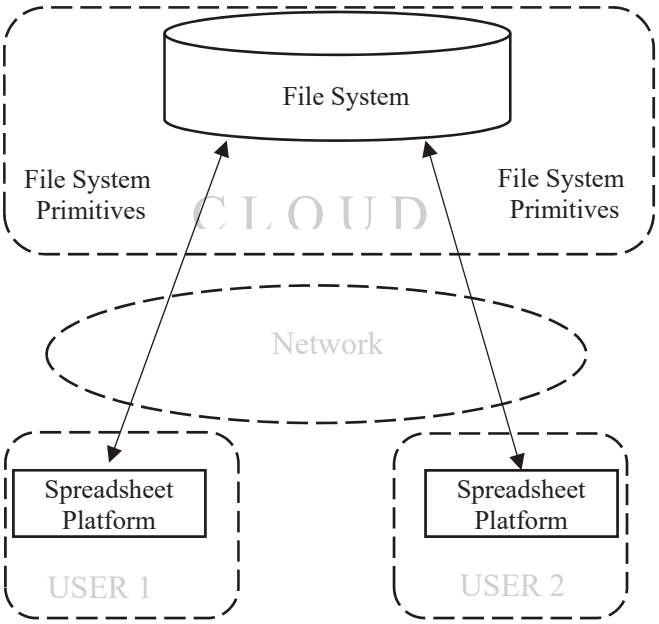

Fig. 1: Spreadsheet Sharing: Storage as a Server

The second step, shown in Fig. 3, leads to a system in which a single Spreadsheet Platform replaces the separate user Spreadsheet Platforms. The single Spreadsheet Platform manages data sharing at the application level rather than at the File System level.

The "Software as a Service"paradigm well matches the business model of those companies, like Google, that provide free services in exchange of user information. The fact that processing takes place in the Cloud is a technical justification of the fact that the data must reach the Cloud in the clear and that, as a consequence, the company that operates the Cloud server can look at the data. As a consequence, a significant weakness of spreadsheet sharing through "Software as a Service" is that such a solution does not support information confidentiality.

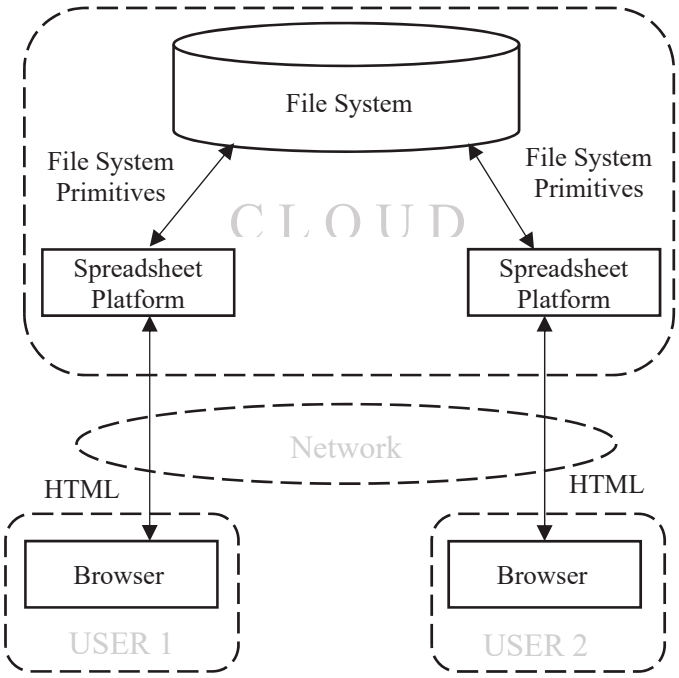

Fig. 2: Spreadsheet Sharing: from Storage as a Service to Software as a Service.

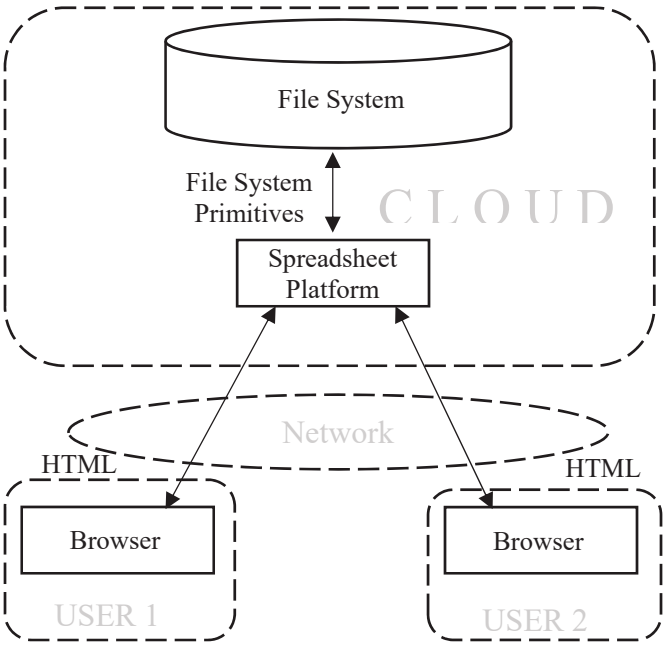

Fig. 3: Spreadsheet sharing: Software as a Service on a single Spreadsheet Platform

A second weakness is that the Graphical User Interface provided by Web Applications cannot be the same as the one

provided by standalone applications, mainly because the latency introduced by the network does not allow obtaining the same look and feel that local processing allows obtaining. In addition, Cloud based spreadsheet platforms do not offer complete backward compatibility with desktop spreadsheet platforms and in particular with the desktop version of Microsoft Excel, which is by far the most pervasive spreadsheet platform in the world. Interestingly, this limited compliance applies not only to Google, as one would expect, but also to Microsoft Office in the Cloud.

III. Spreadsheet Sharing through Synchronization

File Synchronization is a different way to obtain sharing. The idea, in this case, is to have the spreadsheets reside in the user File System and the Spreadsheet Platform run in the user system, as it used to happen in the pre-Cloud days, and to count on an external File Synchronization Platform implemented as a Cloud service to synchronize spreadsheets (Fig. 4).

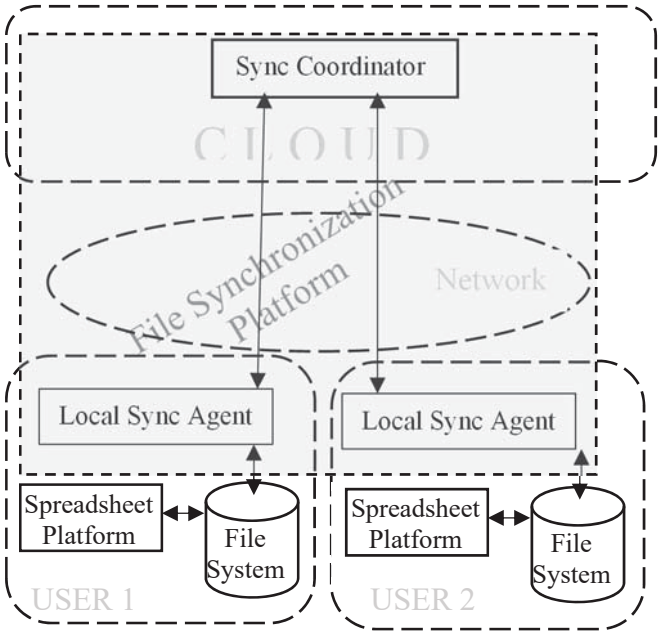

Fig. 4: Spreadsheet Synchronization

Such a File Synchronization Platform is organized as a Client-Server system: the server component, identified as the Sync Coordinator in Fig. 4, runs in the Cloud, whereas the client components, identified as Local Sync Agents in Fig. 4, run in the user systems. The users must configure the File Synchronization Platform to give it the appropriate access rights to operate on the local File System.

Even in the case in which sharing is supported by synchronization a locking mechanism must be provided to handle access conflicts, i.e., the simultaneous attempt of two or more users to update a shared spreadsheet. Before opening a spreadsheet, the Spreadsheet Platform must ask the File Synchronization Platform the appropriate permission to open it. At the reception of such a request, the File Synchronization Platform checks whether the spreadsheet is unlocked and, if it is, it locks it. Then, opening and operation can proceed. Analogously, a spreadsheet close operation first activates synchronization, i.e., the automatic alignment of all the spreadsheet copies with the closing spreadsheet, and then unlocks the spreadsheet.

The advantages of spreadsheet sharing through synchronization over spreadsheet sharing through delocalization are mainly concerned with confidentiality. The Spreadsheet Synchronization Platform may take advantage of end-to-end asymmetric cryptography to guarantee that the shared spreadsheets never leave the private user domain in the clear.

In particular, the Local Sync Agent of the user that has updated a spreadsheet performs the following sequence of operations:

For each remote spreadsheet to be synchronized:

1. it generates a symmetric encryption key;
2. it encrypts the spreadsheet through such a key using symmetric cryptography;
3. it includes the key in a certificate;
4. it encrypts such a certificate using the public key of the target user, using asymmetric cryptography;
5. it creates a data bundle that includes the encrypted certificate and the encrypted spreadsheet;
6. it sends the data bundle to the Synch Coordinator, and through it to the appropriate Local Sync Agents.

It is worth noticing that sharing through synchronization also applies to the Storage as a Service paradigm, with the difference that when using the Storage as a Service paradigm, the spreadsheets reside in the Cloud storage rather than in the user File System.

Considering that to be protected from unauthorized access, both from users and from the Cloud operator, spreadsheets must be stored in the Cloud in encrypted form, to open a spreadsheet, the Spreadsheet Platform, which runs in the user desktop/laptop, must perform the following sequence of operations:

1. it requests exclusive access to the spreadsheet from the Cloud storage,
2. it downloads the spreadsheet, and
3. it decrypts the spreadsheet.

Then, at closing, the Spreadsheet Platform must perform the following sequence of operations:

1. it encrypts the updated spreadsheet,
2. it uploads it to the Cloud storage, and
3. it releases the exclusive access to the spreadsheet.

An end-to-end mechanism for key management must be provided to have the Cloud storage play the role of a passive repository of encrypted spreadsheets. It follows that when confidentiality is a requirement, the Storage as a Server paradigm does not automatically supports sharing, as it does if confidentiality is not a requirement.

As far as Software as a Service is concerned, simply there is no way to guarantee information confidentiality, as spreadsheets must reside in the Cloud server in the clear to be processed by the Spreadsheet Platform running in the Cloud.

IV. Spreadsheet Element Sharing through Synchronization

File Synchronization does not exhibit any spreadsheet-aware functionality. We see that as a limitation considering that very often in workgroup collaboration it happens that users wish to share only part of their spreadsheets. For example, User A may want to share a range with User B and another range with User C while User C is not supposed to see the range shared with User B. The file granularity prevents such a possibility unless User A creates copies of the ranges to be shared in different files and shares such files. This is an unacceptable burden for users, who may end up making mistakes, using obsolete data versions, etc.

The point, therefore, is to migrate from file granularity to Spreadsheet Element granularity, i.e., from File Synchronization to Spreadsheet Element Synchronization, where a Spreadsheet Element is a cell range, a table or a worksheet. We remind that a cell range is a fixed size array of cells, a table is a variable size list of records, while a worksheet is a complete sheet that includes cell ranges, tables and charts.

Spreadsheet Element Synchronization requires that synchronization must be spreadsheet-aware, i.e., managed by the Spreadsheet Platform, rather than by the File System, as it happens in File Synchronization. We then propose that Spreadsheet Platforms must be empowered through the addition of a new software component that takes care of synchronization. Fig. 5 illustrates the main elements of the proposed solution.

The empowered Spreadsheet Platforms participate in a virtual space called the Spreadsheet Space, which supports tabular data exchange and synchronization. Over such a space, spreadsheets connect to each other, synchronize and exchange information.

Synchronization is implemented as follows: a spreadsheet user grants read access rights on a Spreadsheet Element to a set of users, called the target users, and such users display an image of such a Spreadsheet Element in their spreadsheets. Thanks to Spreadsheet Element Synchronization, any update in a Spreadsheet Element results into an update in the Spreadsheet Element images.

Spreadsheet Element synchronization supports cross-spreadsheet references, i.e., the possibility for a spreadsheet to reference a cell of another spreadsheet, overcoming all the security barriers that operate at the system level. Spreadsheet users are so able to share data by managing the spreadsheet access right at the application level.

A. Functionalities to support Spreadsheet Element Sharing through Synchronization

We call *View* the window that the owner of a spreadsheet opens on a Spreadsheet Element by granting read access rights on it, and we call *Image* the copy of a remote View displayed in a spreadsheet.

Spreadsheet Element Synchronization is based on two functionalities, respectively called *Expose* and *Join*. *Expose* is the functionality through which a spreadsheet creates a View, whereas *Join* is the functionality through which a spreadsheet creates an Image of a View. *Expose*/*Join* create a permanent asymmetric connection from a source spreadsheet, i.e., the one that exposes the View, to the target spreadsheets, i.e., the ones that include Images of the exposed View. The "permanent" adjective denotes the fact that the View-Image relationship remains active until it is explicitly removed, whereas the "asymmetric" adjective denotes the fact that the two spreadsheets play different roles, namely the source spreadsheet owns the Spreadsheet Element whereas Images are just read-only copies of the Spreadsheet Elements. Any update in a Spreadsheet Element appears in the corresponding View and as a consequence in the corresponding Images.

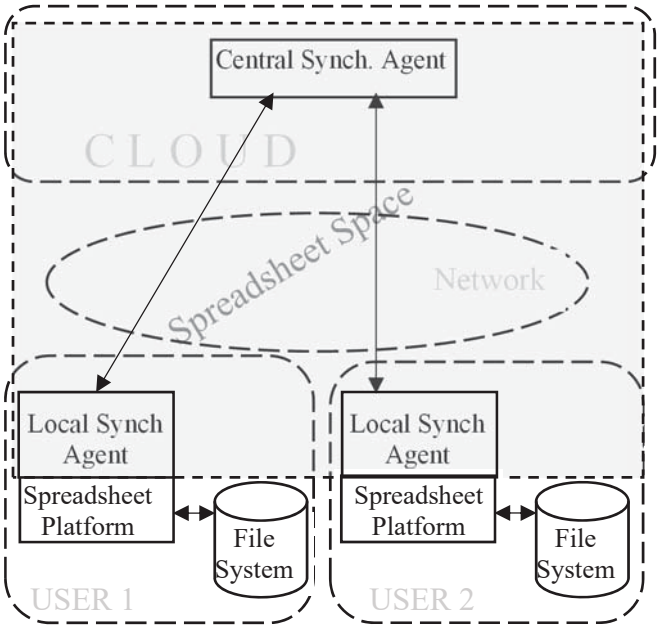

Fig. 5: Spreadsheet Element Synchronization

Three aspects deserve a deeper description:

- Formats and Styles: Views and Images preserve Spreadsheet Element formats and styles. The Images appear in the same way as the Views to which they are connected.
- Formulas vs Values: Views and Images must be able to include both values and formulas.
- Update policy: Upon the update of a Spreadsheet Element exposed as a View, it is expected that the update appears in the corresponding Images. The time at which that happens depends on how the source and the target spreadsheets have configured operation. Both *Expose*, at the source, and *Join*, at the target, can be configured either in manual mode or in automatic mode, as indicated in the table below.

	Manual	Automatic
Expose	View update takes place upon an explicit command issued by the exposing user.	View update takes place at the update of the corresponding Spreadsheet Element.
Join	Image Refresh takes place upon an explicit command issued by the joining user.	Image refresh takes place at the update of the corresponding View.

V. Spreadsheet Overlay

The distinctive feature of Spreadsheet Element Synchronization is that it allows spreadsheet users to expose parts of their spreadsheets to other users. So User A can expose a cell range to User B, who in turn exposes a table to User C, who does not have access to User A data.

It is immediate to observe that from the application point of view Spreadsheet Element synchronization is scalable whereas Spreadsheet Synchronization is not. Fig. 6 shows a workgroup collaboration scheme represented as a directed graph in which several spreadsheets shares Spreadsheet Elements. We call such a scheme a Spreadsheet Overlay.

A Spreadsheet Overlay can be associated to a graph. It can be created through a graphical creation environment in which the graph nodes and edges have properties. Each node, corresponding to a spreadsheet, is associated to a set of properties that includes the owner identifier, while each edge is associated to a set of properties that includes a description of the Spreadsheet Element that the source user exposes to the target user.

VI. Spreadsheet Overlay Use Cases

The following use cases help to understand how Spreadsheet Overlays can be applied in real situations. The first use case, called Product List Distribution, is concerned with the situation in which a product list manager maintains a product list in a central spreadsheet or in a Software Platform, and makes the regional product lists available to the regional managers.

The regional managers display their regional product lists and create personal analyses and presentations. As the regional manager spreadsheets and the central spreadsheet constantly maintain alignment, the regional area personal analyses and presentations automatically follow the evolution of the product list (e.g., product price variation, availability, etc.).

The second use case, called Budget Consolidation, is concerned with the situation in which a Managing Director (MD) of a company prepares the global company budget by integrating the budgets of different departments. The MD first creates a Budget Form and makes it available to the Department Directors (DD), who install it in their spreadsheets, thus triggering its exposition as a View directed to the MD.

The MD joins the DD Views, displays the Images of all the DD budget Forms and creates the entire company budget by integrating the corresponding elements of the DD Forms. The MD can browse the department budgets and the total budget and ask the DDs to make updates when necessary, counting on the fact that such updates immediately appear in the MD worksheet and affect the global company budget. At the end of the process, the MD unjoins the Views and freezes the final budget.

The third use case, called Fact Table Processing, is concerned with the situation in which a data source exposes a Fact Table that includes a sequence of transactions and grows over time as new transactions occur. The View consists of a dynamic number of rows, each of which includes a time stamp and a set of values related to the parameters of the transaction. An operator processes the View, using spreadsheet formulas, pivot tables or VBA code, to extract sub-tables and/or to create dynamic dashboards and calculate dynamic Key Performance Indicators to be exposed to users for further processing.

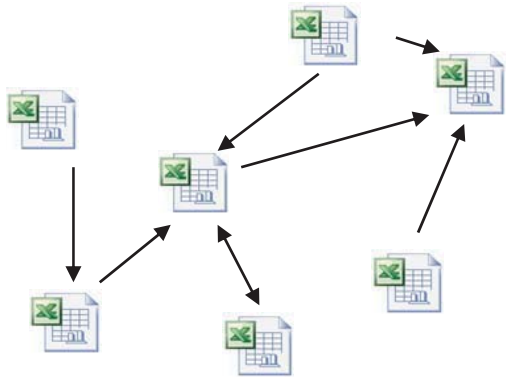

Fig. 6: Software Defined Spreadsheet Overlay

VII. Experiments

We developed a test system and made it available as a free service over the Internet (www.spreadsheetspace.net) for Microsoft Excel. The platform includes a Spreadsheet Client and a Cloud Server. In addition, it offers a Web Service Interface for those who want to implement a connector and a management console.

The Spreadsheet Client exploits COM Excel Addin and C# technology. It offers a standard ribbon interface through which Microsoft Excel users can activate the synchronization services. The Cloud Server exploits Open Source Java technology and runs in the Amazon EC2 Cloud. The Web Service Interface allows application developers to create software platform connectors using standard interfaces based on REST Web Services and JSON data format. Finally, the Management Console allows users to configure the synchronization service.

We ran several field experiments, in large/small companies and public administrations. The following list summarizes the indications that we obtained from such experiments and that contributed to the development and to the engineering of the platform. We classify such indication in four categories, namely indications related to the Spreadsheet Client, indications related to the Cloud Server, indications related to Operation Administration and Management, and indications related to functionalities:

A. Indications related to the Spreadsheet Client

- Installation: The Spreadsheet Client must be installed/uninstalled using standard procedures and with no administrator privileges.
- Firewall/Proxy traversal: The Spreadsheet Client must include appropriate procedures to connect to the Cloud Server from private networks protected by firewalls and proxies in a transparent way.

B. Indications related to the Cloud Server

- Public Cloud server: Scalability imposes that the public Cloud Server must play the role of a View router. All operations, in particular compression and encryption, must

take place in the Spreadsheet Clients whereas no operation must take place in the Cloud Server.

- Private Cloud server: To support full data privacy the Spreadsheet Server must be deployable in enterprise data centers and private administration domains instead of using the public service available in the Cloud;

C. Indications related to Operation, Administration and Management

- Management Consoles: A set of management consoles tailored to the different user roles must provide the appropriate tools for View/Image management as well as for SO management, user management and performance management.
- Monitoring: The Cloud Platform must log the Spreadsheet Client presence, i.e., activation and reachability, the Spreadsheet Client activity, i.e., View Exposes, Joins, Updates, and Image Refreshes, and the Cloud Server Activity. All the events, appropriately time-stamped, must feed both traditional textual logs and live dashboards for the different user roles.

D. Indications related to functionalities

- Versioning: As manual synchronization allows exposers, both spreadsheets and software platforms, to create successive versions of the same data, a mechanism to track the data versions and to navigate over such versions is mandatory.
- Send/Receive: Pilot users requested a simple functionality to send/receive spreadsheet elements to/from other users without leaving the spreadsheet. Although such a functionality seems to be already covered by email, users pointed out that its direct support in spreadsheets facilitates people collaboration and eliminates the errors due to cut/paste/switch to email etc.
- Other functionalities: Two additional functionalities based on spreadsheet synchronization are currently under investigation. Such additional functionalities, not included in the current pilot, include:
  - Selective Join, to join only a row of a View of a table instead of joining the entire table. Such a functionality was proposed to subscribe to an item of a list, e.g., a stock record in a stock record list;
  - Bidirectional-Share, to support simultaneous spreadsheet element co-editing. Such a functionality does not differ from the one offered by Google Spreadsheet and Zoho, but it is based on Spreadsheet Element Synchronization and, as a consequence, supports information confidentiality.

VIII. Conclusion

In this paper we have introduced Spreadsheet Element Synchronization to support spreadsheet sharing. Spreadsheet Element Synchronization enables the creation of a virtual space for spreadsheet-based collaboration. The paper starts from the fact that Cloud based sharing does not support information confidentiality while synchronization based sharing does. It then focuses on Spreadsheet Element sharing and introduces Spreadsheet Overlays, i.e., software defined systems of synchronized spreadsheets. It finally presents some Spreadsheet Overlay use cases and reports the feedback obtained in field experiments.

[1] Nardi B. A. 1993. A Small Matter of Programmig, Perspectives on End User Computing, MIT Press.

[2] Ko A.J., Anraham R., Beckwith L., Blackwell A., Burnett N, M., Erwig M., Scaffidi C., Lawrance J., Lieberman H., Myers B., Rosson M. B., Rothermel G., Shaw M., and Wiedenbeck S. 2011. The State of the Art in End User Software Engineering, ACM Computing Surveys, Vol. 43, pp. 21-44.

[3] Scaffidi C., Shaw M., and Myers B. 2005. Estimating the number of End Users and End User Programmers, Proc. IEEE Symp. Visual Languages and Human Centric Computing (VLHCC '05), pp. 207-214, 2005.

[4] Erwig S. 2009. Software Engineering for Spreadsheets, IEEE Software, Vol. 26, N. 5, pp. 25-30.

[5] Grossman T. 2002. Spreadsheet Engineering: A Research Framework, Proc. European Spreadsheet Risks Interest Group Symposium, Cardff, Wales, July.

[6] Mangiante S., Maresca M., Roncarolo L. 2012. SpreadComp platform: A new paradigm for distributed spreadsheet collaboration and composition, Proc. 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing, Pittsburgh, PA, USA.

[7] Fisher D., DeLine R., Czerwinski M, Drucker S. 2012. Interactions with Big Data Analytics, ACM Interactions, Vol. XIX (3) May-June, 50-59.

[8] Google 2015. Google Docs, http://www.google.com/docs

[9] Microsoft 2015. Microsoft Excel Online, https://office.live.com/start/Excel.aspx,