Semantic Alliance Framework:

Integrating Documents and Semantic Services

by

Catalin David

a thesis for conferral of a Master of Science in Smart Systems

Michael Kohlhase

Name and title of first reviewer

Andrea Kohlhase

Name and title of second reviewer

Date of submission: May 31, 2012

School of Engineering and Science

Contents

Abstract

In the context of increasingly complex software programs and thus increasingly complex documents, a solution must be found to assist users in authoring, reading and especially *understanding* the documents they interact with. In the current work, I present an architecture and software framework for semantic allies: semantic systems that complement existing software applications with semantic services and interactions based on a background ontology. On the one hand, the framework follows an invasive approach to technology integration (applications are "invaded" with semantic services): users can profit from semantic technology without having to leave their accustomed workflows and tools. On the other hand, the framework tries to be application-independent, but still relies on the (open) API of the invaded application. We validate the `Semantic Alliance` framework by instantiating it with a spreadsheet-specific interaction manager, thin extensions for `LibreOffice Calc` 3.4, `MS Excel` 2010 and `Google Docs` Spreadsheets, and an HTML renderer.

Acknowledgements

The present work benefited from the input of Andrea & Michael Kohlhase, who provided valuable comments, ideas and assistance to the writing and development of the research summarised here.

The current work is part of the `SiSsi` project, a joint project between DFKI Bremen and Jacobs University Bremen. More information can be found at [SiSsi]. The `Semantic Alliance` framework is licensed under the GPL and is available at `https://svn.kwarc.info/repos/sissi/trunk/`.

I would also like to acknowledge that parts of the content are already submitted for publishing part of the Mathematical Knowledge Management 2012 Conference paper [Dav+12].

I hereby acknowledge that this thesis is independent work and it has not been submitted elsewhere for the conferral of a degree.

Catalin David

1 Introduction

Recently, the search giant Google has surprised the Internet community by the introduction of a new search feature: the "Knowledge Graph" (see [Goob] for project page). Unlike the traditional search algorithms, with this technology Google commits to being more than just an indexing service: the algorithms now try to understand the *meaning* of the indexed data and how this data is interconnected. The "Knowledge Graph" is invisible to the user, but it is used in generation of results and answers to Google Searches. The "Knowledge Graph" is explored upon each Google Search and it is most efficient when users are looking for an answer, e.g.: "the height of Eiffel Tower". Here, instead of providing search results, Google Search provides an answer, as well as additional information about Eiffel Tower, as we can see in Figure 1.

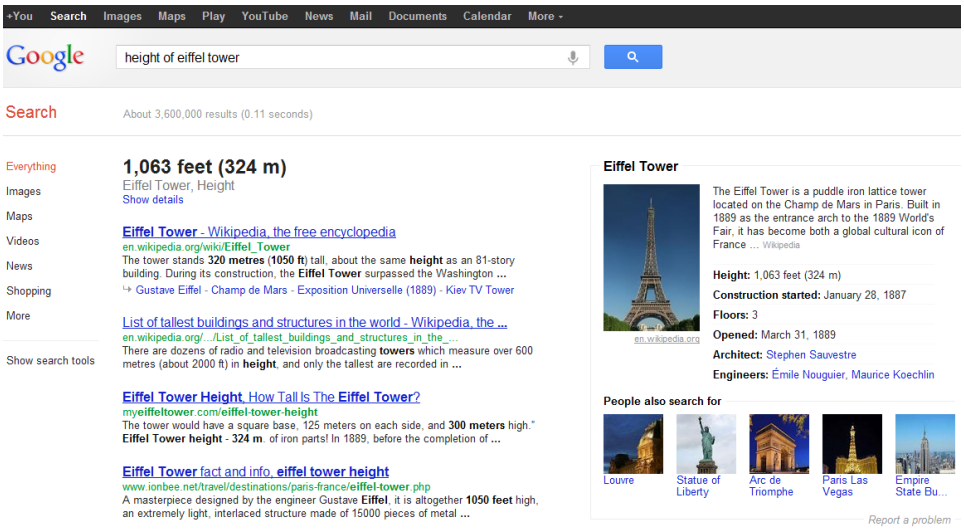

Figure 1: Google Knowledge Graph provides the answer to the query [Gooc]

The data for the "Knowledge Graph" comes from what Google indexes: documents. In the past, search engines would only index HTML documents, but now, more and more types of documents stored online are indexed: PDF, spreadsheets, presentations etc. Besides web pages, information is also stored in such documents and users are interacting with a specialized software for each type of document; **document players** are applications whose purpose is to give the user (read/write) access to structured data collections that can be interpreted as "documents" or "collections of documents" in some way. Prime examples of this category of applications include office suites, CAD/CAM systems and Web 2.0 systems.

The document players with which most users interact on their computers are usually not concerned with the documents within them – they are

adjusted to display the data rather than to understand and reason about the data. With the explosion of the Internet, initiatives such as the Semantic Web (detailed in Section 7) have looked at ways in which data (e.g.: in documents) can be enhanced. One idea is to take advantage of the computational power of computers; for that, the data has to evolve from a binary format and become more meaningful for computers such that they can understand and reason about it. In general, the additional information that is "attached" to data describes it or links it to existing objects that the computer can already understand – this information is called **metadata**, data about data. Since the additional information is concerned with the meaning (semantics) of data, we call this **semantic information** and the data enhanced with semantic information **semantic data**.

Semantic data is important because it bridges the gap between content and form – what an object is and how it is represented. Tim Berners Lee, the inventor of the World Wide Web (WWW) and director of the World Wide Web Consortium (W3C), has acknowledged the importance of semantic data and has already pushed forward initiatives such as the Semantic Web and Linked Data. His vision is that the next version of WWW (3.0) will be the *"Social and Semantic Web"*, an upgrade from the current WWW (2.0), the *"Social Web"*. In the *"Social and Semantic Web"*, data always has a meaning and all knowledge items are inter-connected. The distinction between form and content can also be seen in other web standards, such as MathML [Aus+10a]: MathML comprises two substandards, one for mathematical notations (Presentation MathML) and one for structure and content (Content MathML). Mathematics is a topic where the disambiguation between content and form is important, as there might be multiple notations for the same meaning (e.g.: binomial coefficients $\binom{n}{k}$ versus C_n^k).

Semantic technologies represent software systems built on top of semantic data. Research in this area is based on data with a various degree of semantic enhancements, from just relations between objects up to fully semantic data (a complete, formal representation that computers can understand).

Semantic technologies are used in research regarding the latest innovations in web standards (mainly driven by the Semantic Web), application interaction and data sharing (e.g.: Linked Data [Hea+]), but also in Mathematical Knowledge Management (MKM), Logics and Theorem Proving. MKM, Logics and Theorem Proving are interesting research areas for semantic technologies due to their degree of formality: in order to build proof systems, computers must first understand the concept of inference and possibly mathematics which can be formalized more easily. Thus, for research in these topics it is both easier and more important to develop semantic technologies: they can be prototyped on semantic data and documents where meaning is well-understood and then be transferred to other domains, where meaning is less clearly given. These semantic technologies are frequently

realized in standalone applications (e.g. [Mata; Matb; Act; Cin] for Mathematics). The advantage is obvious: a standalone system can be designed autonomously without interoperability constraints towards other systems or users' previous practices with other systems. Thus, with 'no' compromises, standalone semantic technologies are typically very specialized in the kind of service they offer and very good at that.

The main disadvantage of standalone semantic systems is that they are insular. On a conceptual level, workflows of users are centered around goals to be achieved. Therefore, if the main goal is not the one solved by the standalone system, then systems must be switched, so the insularity of standalone systems disturbs the workflow. On a technical level, the insularity often results in interoperability issues with other systems, augmenting the disturbance of a user's workflow e.g. by the necessity of explicitly reentering data into a possibly new structure of another system. These effects are aggravated by cognitive issues: important context information and thus understanding may get lost when switching systems: [Joh10] shows that even a focus change on a small laptop screen flushes information from short memory. All of these issues create a gap between standalone systems and other parts of the information infrastructure and workflows, and conspire to keep potential users from adopting standalone semantic systems.

An option for overcoming the insularity issues is to view the semantic systems as **semantic service providers** – for this, they should not be stand alone, independent systems, but they should support integration with other applications; we consider semantic services to be already existing. Then, semantic services can be incorporated in the workflows of the users, without disrupting the current activity, while providing the user with the advantages of a semantic system. In earlier research on the adoption issue (cf. Semantic Authoring Dilemma; see [Koh05a]), it has been argued for the creation of *invasive semantic technology*, i.e., a technology, where the semantic services are embedded in (figuratively: invade) the host application, such that the services can draw on users' previous knowledge and experience with well-known authoring tools. This approach was inspired by HCI-driven *directness* requirements in [HutHolNor85] and re-use ideas from Software Engineering in [Ama03].

When designing invasive semantic technology, one approach is to directly embed the semantic services as a semantic extension for certain applications. In this way, the user workflows are not disturbed and the semantic functionality is made available to the user. Still, with this approach, the same semantic functionality would have to be reimplemented for similar applications, in possibly different programming languages, with different constraints imposed by each application. Moreover, architectural differences like the ones between desktop applications, server-based web applications or even mobile applications require radically different solutions for accessing the background knowledge.

A different approach, that we explore in the current work, is to design an extensible framework for semantic services that can be easily integrated into any document player. The gain in this approach is that the semantic services can be designed once and be reused across different document players for the same document type or, in the case of generic services, even across multiple document types. From an implementation point of view, the impact of the constraints the document player might impose in the previous approach is significantly decreased, but not inexistent.

The central idea to address the remaining issues is based on a combination of the Semantic Illustration architecture in [KohKoh09d] with a new approach towards Invasive Design (more details in Section 2). This gives rise to an innovative framework for semantic extensions that is presented in this paper. This framework realizes an "invasive" user experience not by re-implementing semantic technologies in the host system, but by letting e.g. a separate semantic system contribute semantic services and interactions to the host user interface, managed by a 'semantic ally' (more details in Section 2).

The current work is closely related to the SACHS project – a `MS Excel` 2003 add-in that is designed as a semantic extension of a spreadsheet document. The SACHS implementation uses the invasive technology approach to design and integrates semantic services into `MS Excel` 2003. More details about the project can be found in Section 3.3. The current work, part of the "`SiSsi`" project (Software Engineering for Spreadsheet Interaction), explores the possibilities of porting the functionality provided by SACHS to more spreadsheet systems: spreadsheet users are locked into different applications like `LibreOffice Calc`, `MS Excel`, or `Google Docs` for reasons beyond the control of the author. As argued before, offering the SACHS functionality as *invasive technology* induces a development effort similar to the original one for SACHS (due to the complexity of the functionality). Therefore, in the current work, we use Semantic Illustration and Invasive Design as the basis and try to recreate the SACHS functionality.

It is worth noting that the framework is in no way constrained to spreadsheet documents or players, but we will look at spreadsheets in detail, as they are simpler to understand (than e.g.: CAD/CAM documents) and they are closely connected to our implementation.

In Section 2 we first elaborate on the terms of Semantic Illustration and Invasive Design and the result of their combination, followed by a presentation and discussion of the "`Semantic Alliance`" architecture, which allows to build semantic allies, reusing semantic system components and technologies to drive down the cost. In Section 3 we will look in detail at what spreadsheets are and why it is important that semantic services are integrated in them. Section 4 gives an overview of the technologies indirectly involved in the current framework implementation. In Section 5, we validate and report on our experiences with a first implementation – "`Sissi`" – of

this `Semantic Alliance` framework. In Section 6, we will look in detail at the various components of the `Semantic Alliance` framework implementation, in Section 7 we will look at related work and in Section 8 we summarize `Semantic Alliance` and draft upcoming projects.

2 The `Semantic Alliance` Framework

The `Semantic Alliance` framework is built on top of two concepts: Semantic Illustration and Invasive Design. We will first explore both of them and then see how we can combine these approaches in a framework for semantic allies: `Semantic Alliance`.

Semantic Illustration

In order to understand the Semantic Illustration, we first look at what it is, why it is needed and how the `Semantic Alliance` framework is based on Semantic Illustration.

In the **Semantic Illustration** [KohKoh09d] architecture, semantic technology is made fruitful by "illustrating" existing software artifacts semantically via a mapping into a structured background ontology $\mathcal{O}$. Semantic services can be added to an application $\mathcal{A}$ by linking the specific meaning-carrying objects in $\mathcal{A}$ to concepts in $\mathcal{O}$, that is $\mathcal{A}$ and $\mathcal{O}$ are connected via a **semantic link**. This approach comes in contrast to the Semantic Web one where information resources are enhanced into semiformal ontologies by annotating them with formal objects (e.g.: when annotating resources with Dublin Core [Dub] metadata terms such as *dc:creator*, one uses a representation of the object – a string representing the name of the author, rather than a URI to an author page). One advantage of the Semantic Illustration is that it makes the ontology reusable and at the same time it avoids the high implementation costs inferred by reauthoring the ontology for each usage, by requiring only a link to an ontology term (e.g.: URI) rather than a representation of it (as in the Semantic Web approach).

The Semantic Illustration approach is most useful when considering resource reusability. In a setting where the same ontology is shared across multiple documents of different types, a link rather than an object representation is more efficient. Even more, the systems that allow the connection between a document and an ontology become simpler: the systems do not have to be specific to the ontology, but they should be able to work with any ontology that can be linked to.

In order to understand how Semantic Illustration works, we first need to know what software artifacts (or objects) are available to the user. The set of objects that a user can interact with within a document is determined by the purpose, in general, and the implementation, in particular, of the document

player. We assume that these objects have a user interface representation with a specific purpose.

For example, the purpose of spreadsheet players (such as `MS Excel` or `Google Docs`) is to support the user to read, author and interact with ledger sheets. Therefore, based on what UI elements a user interacts with in a spreadsheet player, the objects are: cells (points in a two-dimensional table), formulas (one per cell), functional blocks, tables and legends (these are strictly related to the ledger sheet functionality; an example of other objects a user might interact with are charts). A **functional block** (introduced in [KohKoh09a; KohKoh09c]) is a rectangular region in the grid whose cells can be interpreted as input/output pairs of a specific function. For example, in Figure 4, the range [B3-D3] represents current salaries for employee Andrea as a function of time. The objects the user can interact with are totally different for word processors, such as `Microsoft Word` or `LibreOffice Writer`, where the objects are more closely related to text (e.g.: sentence, paragraph, section). We call these purpose-dependent objects **semantic objects**.

Another advantage of Semantic Illustration is its extension of a document player given semantic objects. A UI is considered good if it is semantically transparent. A **semantically transparent** user interface, as defined in [KohKoh09d], to be an interface that enables the user to access the semantic objects and their relations via UI objects. In general we call a semantically transparent UI any UI that allows the user to access the purpose behind any semantic object.

Unfortunately, the objects that users can work with and the semantic objects a document player has to offer are not necessarily the same. For example, most spreadsheet players do not have the notion of a functional block that the user can directly access. The benefits of the semantic link are that it can tie any object that the user has access to (e.g.: even a functional block) to an ontology term. With the help of this link, an ontology term can point to concrete UI objects, but also to such semantic objects, which are not usually available to the user, making the user interface semantically transparent.

Invasive Design

Invasive Design is a new approach to software design that aims to obtain a similar effect on the user as Invasive Technology [Koh05b]. The **Invasive Design** approach looks at the user interface from a user perspective. A service $\mathcal{S}$ (we consider services to be functionality providers, regardless of their type) "feels" embedded into an application $\mathcal{A}$ if the service occupies a screen-area $\mathcal{D}_{\mathcal{S}}$ that overlaps with the area $\mathcal{D}_{\mathcal{A}}$, originally claimed by the application itself. If the user has requested a service $\mathcal{S}$ from within $\mathcal{A}$ and the service "feels" embedded, then the user *perceives* $\mathcal{S}$ as an application-

dependent service. This perception is amplified if the service and its request refer to the objects with which the user interacted and the service information is displayed in the visual vicinity. The embedded services that make use of this effect are called **contextualized** services.

A common basis for both Invasive Technology and Design is that the users are important and the users are not willing to leave their environment in order to execute a task: a user is not willing to leave the document player in order to achieve a semantic task. Therefore, an *invasive* approach was needded. In contrast to the Invasive Technology approach, where the framework components are truly invasive, the Invasive Design approach relies heavily on the perception of the user with regard to embedding. This is achieved by the common logical fallacy that "correlation implies causation" – in this case, a click on the user interface and the information provided by a contextualized service showing up in the vicinity of the click divert the attention of the user away from the implementation details (e.g.: the window does not appear to be fully integrated visually).

In the next section, we will now explore how we can integrate the Semantic Illustration approach with the Invasive Design approach and what benefits this has.

2.1 Invasive Design via Semantic Illustration

The contextualization we looked at above can be assured if we combine the two architecture decisions: Semantic Illustration and Invasive Design. The semantic link that ties objects in $\mathcal{A}$ to terms in $\mathcal{O}$ (Semantic Illustration) is used to provide services $\mathcal{S}$ to the user. By coupling this approach with the approach of Invasive Design, we get a new contextual dimension: the information is related to the object the user interacts with and it is displayed as if it were part of the document player.

Still, just combining these two approaches is not enough. The user has to require a certain service (to trigger its execution) via a predefined interaction. Moreover, depending on the current context of the user (e.g.: document type, document player, focused object), not all services might be available. Last but not least, there might be services that require more than one interaction with the user. All these interactions between the user and the system need to be handled via an **interaction model**. So we need a component that handles the interaction model: **an interaction manager**. In the current context of the architecture, we consider an interaction manager to be a **semantic ally**.

One of the benefits of Invasive Design is that the service $\mathcal{S}$ does not need to be implemented as application-specific invasive technology. A *thin client* [NieYanNov03] invading $\mathcal{A}$ is sufficient to obtain the advantages of invasive technology in the eyes of the user. The only restriction is that $\mathcal{S}$ should be able to outsource the application-dependent parts (i.e.: $\mathcal{S}$ should

be parametric in order to allow the semantic ally to decide what the contextual level is). This makes it easy to offer the same service $\mathcal{S}$ for a range of document players $\mathcal{A}$ at the mere cost of reimplementing only the thin client.

The combination of two concepts detailed above, the Semantic Illustration architecture and Invasive Design, alongside with an "abstract interaction model"-driven interaction manager have led us to the design of a *framework for semantic allies*, which we call "`Semantic Alliance`". This framework has three main components:

- a platform-independent semantic interaction manager (as a semantic ally), that has access to parametric semantic services, and that
- partners with a set of thin, invasive, application-specific components that collaborate with the document player $\mathcal{A}$ to manage the UI events, and that
- has access to a set of application-independent screen area managers that can render the available services.

The key feature of this new architecture is its distributed approach. Alongside with that, some other software design goals that have guided the design of the `Semantic Alliance` framework are:

***Separation of Concerns*:** Each of the `Semantic Alliance` framework components has only one responsability. This is good because it allows the respective component to focus on a single task.

***Swappable Components*:** Due to the fact that the system is distributed and that there is a clear separation of concerns, any component in the system can be swapped with another one that has the same "concern".

***Scalability*:** Due to the separation of concerns and swappable components, this framework can scale in size as long as the communication between the components is API-based.

***Extensibility*:** If a new component needs to be added to the system, then integration is an easy task, as it will only happen at the component interaction level (the API between the new component and the existing ones). This, coupled with separation of concerns and swappable components allows us to easily expand the framework and invade new applications.

Cummulated, all the goals above provide the system with **independence**: only the invasive component of the `Semantic Alliance` framework is tied to the document player constraints (API, operating system etc.). All the other components and the communication between them should be independent from: the constraints of the invasive component, the underlying

operating system, the underlying communication environment. The document types and players for which semantic services are provided should only be relevant for the internals of the semantic ally. Also, mission-critical components should be independent from external software.

2.2 The `Semantic Alliance` Framework: a Mashup Enabler

In the previous section we have seen that with the current design, based on Semantic Illustration and Invasive Design, the framework can host a semantic ally that provides services to any document player extended by a thin client. Still, we do not have a clear description of the services that can be provided by this framework and how Invasive Design is achieved. We assume that the pertinent semantic services already exist, and we call the systems that make such services available **semantic service providers**, as in Section 1. This assumption allows a semantic ally within the `Semantic Alliance` framework to not focus on the semantic services, but only on the integration (or mashup) of semantic service providers with the original application. But in contrast to a traditional mashup that would aggregate data from multiple feeds, the `Semantic Alliance` framework mashes up GUIs with data from the semantic service providers and document players themselves. In this sense, the `Semantic Alliance` framework can be considered a *mashup enabler* for semantic allies, a system that not only transforms otherwise incompatible IT resources into a form where they can be combined, but also fosters the creation of innovative functionalities.

Let us assume $\mathcal{S}$ to be a service provided by a semantic service provider drawing on an ontology $\mathcal{O}$[1]. In the `Semantic Alliance` framework, the task of the mashup enabler between $\mathcal{A}$ and $\mathcal{S}$ is split into three parts (see Figure 2).

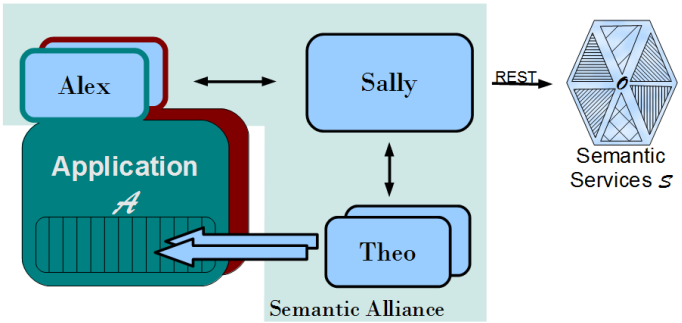

Figure 2: `Semantic Alliance` as a Mashup Enabler for Semantic Allies

[1]Since $\mathcal{O}$ is probably the biggest investment necessary for enabling semantic services (discussed in [KohKoh09a]) via `Sally`, a shared, web-accessible mode of operations for $\mathcal{S}$ is probably the most realistic setup.

`Sally`[2] The main part of the mashup enabler resides in the "`Sally`" component which coordinates the integration of the functionalities of $\mathcal{A}$ and $\mathcal{S}$ into a joint user interface with which the user interacts, thus we can consider it to be an interaction manager. This interaction manager achieves the semantic extension of $\mathcal{A}$ by Semantic Illustration through Invasive Design. The coordination of the user interface is driven by an interaction model which is based on an abstract data model for each document type. This abstract document model abstracts from the particulars of the data structures in $\mathcal{A}$ and allows `Sally` to tie interactions to specific items in the abstract document model. This model allows `Sally` to differentiate and respond differently when faced with a click on a cell in a spreadsheet document and a click on a textbox in a presentation.

`Alex`[3] This is the only completely invasive part of our architecture and it extends the application $\mathcal{A}$. In order to achieve maximum scalability, we want this component to be slim, while it should still provide the necessary communication with the document player. The purpose of `Alex` is to report relevant user interactions to `Sally` (e.g.: user clicks on a cell in a spreadsheet) and to execute instructions from `Sally` (e.g.: move the cursor to a certain position within the current spreadsheet). The **semantic map** (a collection of semantic links from semantic objects in $\mathcal{A}$ to concepts in $\mathcal{O}$) is document specific and handled by `Alex`. Since we want our documents to be self-contained, the semantic map is stored inside the document. The self-contained property of the documents provides the system with another desired functionality: independence from document players (e.g.: a spreadsheet document authored in `MS Excel` is viewed in `LibreOffice Calc`– the semantic experience should be the same). It is worth noting that any application that provides an open API (as described in Section 2.4) can be invaded in this way. Furthermore, by having an `Alex` that is thin, we limit the dependency of the entire architecture with the operating system and the document player; for examples why this is important, see Section 3.3.

`Theo`[4] In order to provide the user the illusion of invasion and fully integrate the semantic workflows of `Sally` into the document player, we need a screen area manager `Theo`. The generated content (usually by $\mathcal{S}$, but not only) is embedded into the application that is invaded.

[2]For **S**emantic **ally**

[3]We have named this component after Alexander the Great, one of the mightiest invaders in history.

[4]German readers may recognize that "Theo" is a shorthand for "Karl-Theodor" as someone who pretends towards others that he does something, but in fact he's doing something else.

This can be done at different levels, depending on the features and capacities of the underlying system (for a broader discussion, see Sections 2.3 and 2.4). Obtaining the embedding at the *operating system level* represents a true implementation of the idea of Invasive Design and it can be achieved by superimposing the information window over the GUI of $\mathcal{A}$. As argumented before, from a user perspective, even if the integration is not native, but through Invasive Design, this still achieves the purpose of invasive technology.

2.3 Framework Support for Desktop and Mobile

Nowadays, software applications are distributed in three different ways, depending on the internal architecture and features that they provide:

1. *desktop applications*
2. *mobile applications*
3. *browser based applications*

In an extensive report from 2009 [Thea], Morgan Stanley, the global financial services firm, claims that in the future, "Mobile Internet" will be *"at least 2x size of Desktop Internet"*. The claim has been supported by more recent reports [Ove] and similar prognosis by Cisco Systems, Inc. (one of the biggest providers of networking equipment) [Cis]. Therefore, most new systems take this into considerations and do not only provide a desktop application, but also mobile ways for the user to access the same content. Therefore, the `Semantic Alliance` framework was designed with all current possible information access modes in mind:

1. For a *desktop application*, all the components of the `Semantic Alliance` framework run on the local machine and use a `Theo` to provide operating-system level GUI embedding (see Figure 2). `Sally` is not constrained to live on the local machine: it can be easily shared as a semantic ally by multiple users.

2. For a *mobile application* two approaches are most common: i.) the application is just a constrained browser window used for rendering website pages from the application home domain (case in which the mobile application is just a browser application) or ii.) the application makes a distinction between content and presentation: the data (content) is stored on an application server $\mathcal{A}_{core}$, whereas the rendering of the data and the user interface to the data ($\mathcal{A}_{UI}$) reside on the mobile device. In the latter situation, the constraints made by the operating system (such as the inability to run another application) as well as the lack of computation power (e.g.: for formal verification tasks)

require the `Sally` framework components to be run as web services on a `Sally` server. Due to these constraints, `Alex` bears the role of the middleman in this architecture, integrating with $\mathcal{A}_{UI}$ for document player interactions, with $\mathcal{A}_{core}$ for data interaction and with `Sally` in order to aggregate all this data, coordinate all other components and provide an abstract interaction model.

3. For a *browser-based application*, one of the scenarios above applies: if we have little restrictions from the operating system (we can start another process), then we consider it to be a desktop-like deployment. If a web-kiosk-like system is considered, then this setup is similar to a mobile application setup.

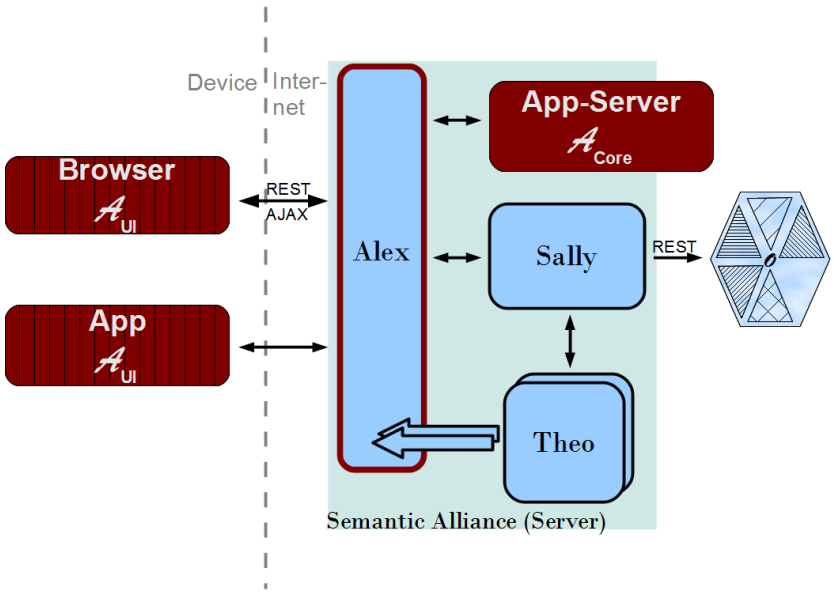

Figure 3: `Semantic Alliance` for Mobile Apps

In all these setups, the `Semantic Alliance` framework components offer the same functionality, perform the same functions and communicate with each other in essentially the same way. The only differences come from technological constraints and how the `Semantic Alliance` components generate and integrate the GUIs with the existing document player.

Observe that we need a two-way communication between `Sally` and `Alex` to realize even simple semantic interactions, whereas `Sally` and the semantic service provider can restrict their communication to a client-server communication where `Sally` takes the role of a client. Observe also that for both the "web/mobile application", the setup in Figure 3 is only possible, if the communication between $\mathcal{A}_{core}$ and $\mathcal{A}_{UI}$ can be based on open APIs. A similar postulate has been made for the "desktop" architecture.

We will now take a look at what Open APIs are and how they influence the efficiency of the current framework.

2.4 Building on Open APIs

The `Alex` component in our architecture is an extension of an application $\mathcal{A}$, which is a document player. `Alex` uses the open API (that allows extending $\mathcal{A}$ from outside) to provide communication about internals of the document player.

There are multiple types of information that the API of $\mathcal{A}$ can offer access to, in particular, this data can be separated into a content API `contentAPI` (the internal document object model or DOM), a form API `presentationAPI` (its presentation) and a document player API `playerAPI` (specific document player functionality such as rendered position, click events etc.). To make a clear distinction, we will consider a spreadsheet again. In this case the `contentAPI` provides access to the data of a cell (formula, value). The `presentationAPI` provides access to presentation data about a cell: cell background color, foreground color, borders and others. The `playerAPI` provides data such as the position of the cell on the screen and event notifications in case a user clicked on a certain cell.

In order to achieve Semantic Illustration, the document player needs to provide access to the DOM (`contentAPI`). Also, in order to achieve Invasive Design, the document player needs to provide access to document player specific representation of data (`playerAPI`). Therefore, we consider an open API of a document player suitable to be used in the `Semantic Alliance` framework, if the open API includes a `contentAPI` and a `playerAPI`. Only with such an API can the semantic ally provide contextual information. Note that "contextual" comprises two dimensions: the information is contextual from a semantic point of view (the semantic link ties objects to ontology terms), and from an HCI perspective (using the positioning features of the `playerAPI`, the information is displayed in the vicinity of the object of interest).

Interestingly, in most desktop document players (in particular spreadsheet players), there is no clear distinction between the three APIs mentioned above. In contrast, in web systems, there is a clear separation between the document itself (comprising of content and form e.g.: for a cell) and its rendering (in the document player). The underlying reason consists in the assumption that the document lives in the "cloud" and the rendered document in a browser. Moreover, the document player is not the browser itself, but just a browser window. In particular, due to security issues the communication language (e.g.: JavaScript) has no access to browser facilities. Thus, unfortunately, the `playerAPI` for document players is very limited or even inexistent for most of these. For example, a spreadsheet player like `Google Docs` provides very little basic functionality (e.g.: no position infor-

mation for cells, no events), influencing the potential for Invasive Design – the list of services can not be dynamic, but it needs to be hardcoded into the interface and there is no event API (the interactions have to be started manually by the user). Due to the lack of position, we can not achieve the purpose of invasive technology. More details about this are presented in Section 6.3.

3 Spreadsheets Need Semantic Services

Spreadsheet players represent computer applications that store, analyze and compute tabular data. The word became popular when tabular data was still represented on paper and the paper was spread, in the same sense as a newspaper. We will use the following terminology:

***Worksheet*:** A theoretically infinite grid of cells (practically constrained in size by the document player)

***Workbook*:** A document which consists of a collection of one or more worksheets

***Spreadsheet*:** A conceptual level representation of a workbook.

On a computer, a worksheet usually consists of a two dimensional representation of a grid with cells organized in rows and columns in which each cell may contain an alphanumeric value and a formula to define how the contents of the cell are generated. The formula can aggregate data from other cells or groups of cells via mathematical functions of variable complexity. The rows are usually identified by numbers (1,2,$\cdots$), while the columns are sorted alphabetically (A,$\cdots$, Z, AA$\cdots$); a cell is identified by the pair of column and row identifier (e.g.: [A1] for the top left cell of the worksheet). There exist multiple spreadsheet players with features that determine users to employ them instead of other technologies (see Section 3.1). Most of the functionality is common to all spreadsheet players, but there are implementation specific features (or bugs) that make a certain spreadsheet player more desirable over another (for a more detail analysis of the ones used in this project, see Section 5.3).

Spreadsheets are built with two operation modes in mind: data input and data output. Data input usually comes from the user – either cell by cell or batch input, while output data is generated for the user based on formulas. With this separation in mind, we can distinguish between input cells (or blocks) and output cells (or blocks). This distinction is important because with its help, more information about the purpose and provenance (user input or computed) of cells can be made available to the user. Still, most spreadsheet players do not make a clear distinction between an input cell and an output cell, requiring the user to check if there is a formula for

	A	B	C	D	E	F
1	**Salaries**	Actual			Projected	
2		1984	1985	1986	1987	1988
3	Andrea	142500	168500	253000	308500	352500
4	Michael	142500	168500	253000	308500	352500
5	**Total**	*285000*	*337000*	*506000*	*617000*	*705000*
6						
7						
8	Salaries	0.285	0.337	0.506	0.617	0.705

Figure 4: Worksheet with information about salaries from [KohKoh09a]

each cell. Based on this, we can conclude that most spreadsheet players do not expose their internal data model to the user (or the document format data model), but restrict themselves to only showing data about individual cells.

In the following sections we will look at why spreadsheets are so popular and what are the drawbacks of this popularity.

3.1 Spreadsheet Players and Spreadsheet Adoption

According to a recent study [AbrErw06], the number of American workers who use spreadsheets is of 23 million workers, which amounts to 30% of the workforce. We can assume similar numbers for all the other countries. Still, it is not clear why spreadsheets are so widely used. Compared to database systems (Database Management Systems – DBMS), the spreadsheet format seems to offer the same functionality: storage, distribution and aggregation of data. But there are major differences between the spreadsheet players and DBMS players. The document players of the two applications seem to make the difference in both targeted audience and adoption.

For spreadsheets, the document player is important – it transforms otherwise static documents into active documents with which the user can interact. Moreover, functionality such as:

- auto-update of the cell contents based on modifications in other cells the current cell formula is dependent on
- auto-update of charts

helps the users immediately realize the impact of their changes.

In addition to that, there is a difference in User Interface (UI): user access to data stored within a spreadsheet is driven by a Graphical UI – namely, the spreadsheet player, while in the case of DBMS, access to data is usually driven by a query language (e.g.: Structured Query Language – SQL). It is easy to just click and edit (as in the spreadsheet player case), while it is usually more complicated to achieve the same goal in a DBMS.

One of the important features that make spreadsheets so widely used is that, mostly without realizing it, spreadsheet users become programmers – behind each cell formula, a small program exists. This program can vary in complexity, from simple mathematical functions to complicated macros that take advantage of the document player API. Programming languages for DBMS exist as well (e.g.: SQL), but they are usually more complicated.

Spreadsheet players make it easier for users to create, edit and delete a spreadsheet – a spreadsheet is self contained, it is just a file on the local file system and it requires only rendering effort from the spreadsheet player. On the other hand, databases and DBMS players are more complicated:

- databases are usually not stored as self contained files; this is mainly due to how data is stored
- DBMS players have a complicated user and permission system for each database; this helps with security, but are not user friendly
- DBMS players are harder to set up when creating/importing a database

All these reasons have driven up the adoption rate for spreadsheets. While spreadsheet players make spreadsheets popular and widely used, there are some issues when looking in detail at spreadsheets. The issues presented next are widespread and they have been acknowledged [Pan00]. The losses incurred by errors and by misinterpretation have even led to the creation of an international task force to battle them [EUS10].

3.2 Spreadsheets as Error Source

Computational Errors Mainly due to their active characteristic and the ability of the users to modify them easily, spreadsheets are error prone. In fact, they are so error prone that recent studies showed that between 1% and 5% of the spreadsheet formulas contain an error [PowLawBak08] and that up to 95% of the spreadsheets contain at least some kind of error [AbrErw06]. Compared to professional programmers, who understand the difficulties of error-free code and are trained to avoid errors, most spreadsheet developers, being largely self-taught, are less aware of the dangers that errors pose. Moreover, this is a more general cognitive problem, as explained in [Pan00], since humans are not capable of doing complex cognitive tasks with great accuracy.

Documentation Another big issue with spreadsheets is the fact that the document players do not support documentation and therefore they are poorly and inconsistently documented. Most spreadsheets used in the financial and administration areas are complex, customized documents, targeting specific requirements, according to a certain data format. Unfortunately, humans do not have a great memory and details about the meaning

	A	B	C	D	E	F
1	**Profit and Loss Statement**					
2						
3	(in Millions)		Actual		Projected	
4		**1984**	**1985**	**1986**	**1987**	**1988**
5						
6	**Revenues**	3.865	4.992	5.803	6.022	6.481
7						
8	**Expenses**					
9	Salaries	0.285	0.337	0.506	0.617	0.705
10	Utilities	0.178	0.303	0.384	0.419	0.551
11	Materials	1.004	1.782	2.046	2.273	2.119
12	Administration	0.281	0.288	0.315	0.368	0.415
13	Other	0.455	0.541	0.674	0.772	0.783
14						
15	Total Expenses	2.203	3.251	3.925	4.449	4.573
16						
17	**Profit (Loss)**	1.662	1.741	1.878	1.573	1.908

Figure 5: A simple spreadsheet after [Win06]

of certain cell values are forgotten. The difference in abbreviations, data format and the complex structure of certain spreadsheets make it hard for non-specialized readers to understand the meaning of the data in a certain spreadsheet. Given that spreadsheets can span multiple columns (therefore the "spread" part) and the screen real estate is usually small, another issue is that the user can easily lose context (e.g.: cell references can be outside of the current view and the user has to maintain mappings such as: $C3 \leftrightarrow$ Salary for 1985). A documentation navigation system would help in this case.

(Mis)Understanding Spreadsheets can vary in usage: from home-made, containing one table up to complex financial analysis and prognosis. The example in Figure 5 is a spreadsheet of a low complexity, but we can still distinguish multiple layers of the semantic model underlying a spreadsheet (for an in-depth analysis of these layers, see [KohKoh11; KohKoh09a]):

***Data Layer*:** This layer contains the cell values and the document properties. These are actively managed by the document player. The meaning of these values is defined in other layers.

***Surface Layer*:** This layer contains visual cues for the interpretation of the displayed data. The table structuring itself (as a visual cue) is used as a cognitive structuring device. For example, the cells in Figure 5 can be divided in three areas based on their colors: a dark area which contains actual and past values for expenses, revenues; a lighter, yellow area which contains values projected from the actual and past values; and a white area that surrounds the other boxes and contains explanatory text and header information.

***Formula Layer*:** This layer contains the formulas associated with the cells. A formula is an expression built from constants, variables, an extended set of numeric and logic operators and references to other cells. The document player usually takes care of the automatic update of the values based on modifications to dependent cells in the formula; of course, the user can override this behavior.

***Background Knowledge Layer*:** This layer is missing in a spreadsheet, but it is needed for understanding the meaning of the values. For example, in Figure 5, the values are presented, but the company name is not present in the spreadsheet. This type of information is not available to the user without context information.

Errors and misunderstandings at the formula and surface layer and especially at the bakground layer can be corrected by a semantic user assistance system that invades the document player and helps the spreadsheets developers and users. For example, in [AndNov08], a multi-layer interface approach is used to handle the problem of the apropriateness of the explanations. In other work, such as [BraPet08; HodMit08], support for understanding spreadsheets is offered by vizualization techniques and data/formula dependency graphs, but they only cover the data and formula layer.

An approach that focuses on the understanding of the background layer is usually harder to devise because, unlike help systems for the other knowledge levels (surface, data, formula), the background knowledge help offered is spreadsheet-dependent. An approach to such a help system is e.g. presented in [Din09] with a documentation-through-annotation approach. The SACHS (**S**emantic **A**nnotation for a **C**ontrolling **H**elp **S**ystem) project [KohKoh09e; KohKoh10; KohKoh09a; Koh10] aims to overcome usability issues for `MS Excel` documents and covers all layers of the semantic model presented above. SACHS has been designed based on the concepts of Semantic Illustration and as *invasive technology*, allowing the user to access semantic services from within the spreadsheet player. Since it is based on concepts closely related to the ones on which the `Semantic Alliance` framework is, we will now look into detail at the SACHS system and see the functionality it provides.

3.3 SACHS

The SACHS system is an add-in for `MS Excel` 2003 (written in Visual Basic for Applications) that aims at providing semantic help facilities for spreadsheets. It has been developed with a use case in mind: the DFKI (German Center for Artificial Intelligence [Deu]) controlling system, but it is usable on any spreadsheet that contains knowledge stored as OMDoc (a semi-formal semantic format based on XML described in Section 4) and semantic links.

An OMDoc document provides the background ontology, so in SACHS the developers facilitate users access to the *semantic link* (the concept was developed in this project) that binds each cell (or group of cells) to a fragment of the OMDoc document. For example, in Figure 6 we can see the semantic links between the spreadsheet document cells and the accompanying OMDoc document. This semantic map (collection of semantic links) is kept in an additional worksheet within the current spreadsheet.

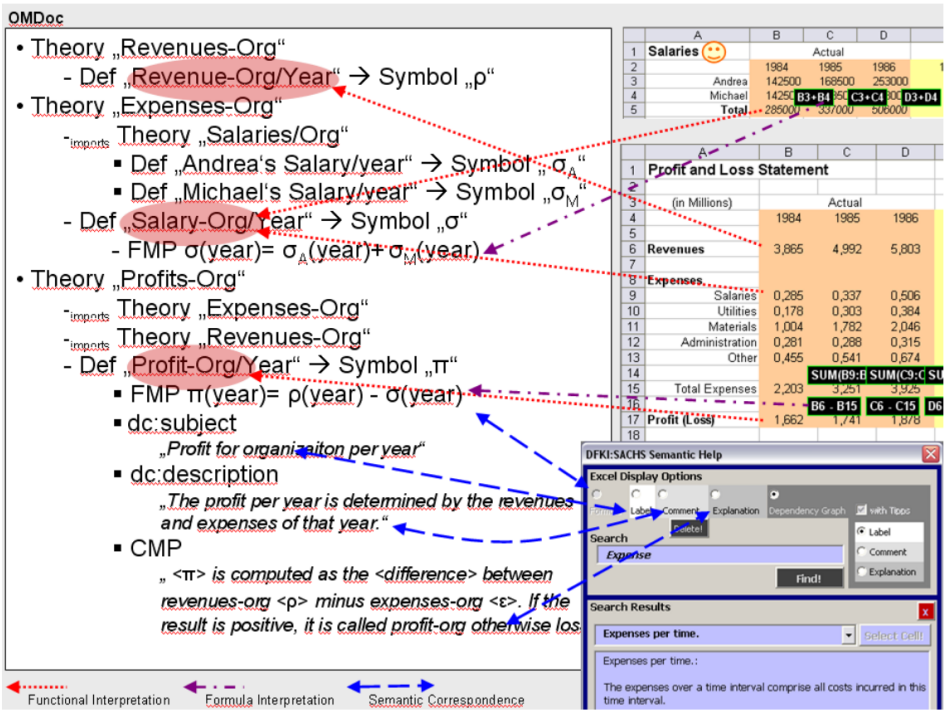

Figure 6: The SACHS Information Architecture and Control/Search Panel from [KohKoh09a]

Based on this map between cells in $\mathcal{A}$ and concepts in the background ontology $\mathcal{O}$, semantic services are developed and made available to the user:

***In-place Help at the Appropriate Level*:** This service is interesting from two perspectives. First of all, it provides information services for readers of a spreadsheet, so they can find out more about the meaning of a certain cell. Also, the user can select the level of detail at which the information is presented: either a label, with concise information or a more thorough description as a definition.

***Functional Block Highlighting*:** For understanding the connection between a certain cell and the rest of the spreadsheet, it is important to

see what other cells are mapped to the same concept in the background ontology.

***Semantic Navigation via Dependency Graph*:** this service allows the user to visualize dependencies of the background ontology starting from the current cell. This visualization model allows interactions between the spreadsheet and the graph (i.e.: a user action in either the spreadsheet or the graph might trigger a reaction in the other):

- a click on a cell in the spreadsheet will show the dependency graph for the ontology item attached to the cell and
- a click on a node in the graph will change the `MS Excel` cursor in the spreadsheet to items pointing to that knowledge item (if any).

***Framing*:** This service allows the information to be presented to the user from different perspectives (frames). Depending on the user's background knowledge, he can choose the perspective through which the information is presented (in which frame) in order to allow for a better understanding of the concepts.

Assessment of Values (potential feature): besides the coupling between the value of the cell and the background ontology item it represents, it is important for the user to also understand what a certain value means. This is where assessment of values comes in – it allows the ontology author to specify what a good value for each concept is and it facilitates the interpretation of the spreadsheet.

In the following section we present the `Sissi` implementation of the `Semantic Alliance` framework that we try to validate. The validation process looks at the development costs required to reach feature parity with the SACHS project, but not for only one document player, but for multiple spreadsheet players: `LibreOffice Calc`, `MS Excel` and `Google Docs`.

4 Preliminaries for `Sissi`

The following set of technologies has been developed over the years at Jacobs University, part of the KWARC group. The systems presented below are aimed towards STEM (Science, Technology, Engineering, Mathematics) disciplines, as these domains make extensive use of mathematics, but the systems are not in any way constrained to these domains. These technologies represent the basis for the ontology $\mathcal{O}$ and the semantic services $\mathcal{S}$.

OMDoc

OMDoc (2000) [OMDoc] is a markup format and data model for **O**pen **M**athematical **Doc**uments. OMDoc is a semantics-oriented representation format and ontology language for mathematical knowledge. It is an XML based format and it is used in a large set of projects: in Automated Theorem Proving [Mul06], eLearning [KohKoh08; Koh07; Mel+03], eScience [HilKohSta06], Document Retrieval [KohSuc06] and in Formal Digital Libraries [Log]. The OMDoc format can represent *semi-formal* knowledge: it uses already existent representations for mathematical formulae (OpenMath [Bus+03] and Content MathML [Aus+10b]) and extends them with representations for context and domain models. Furthermore, OMDoc provides a strong, logically sound module system based on structured "theories" (content dictionaries extended by concept inheritance and views [MMT]). Further information can be found on the project website at [Koh]. In contrast to other ontology modelling languages like OWL [McGHar04], the OMDoc format does not commit to a formal logical language, and therefore lacks a native concept of inference. This allows the format to not constrain the author in representing the knowledge and also to not force the author to fully formalize knowledge. This is ideal because authors can choose their own level of formalization and it is important because the (non-technical) authors do not have to work around the limitations of the underlying logical system.

OMDoc is an XML based format, therefore it is hard for humans to both author and read. For that reason, specific editors have been designed for the task of authoring OMDoc documents. The first approach is an OMDoc-based semantic Wiki which integrates server-based editing with user-adaptive and context-based presentation [LanKoh06; Lan07]. The second approach is the one of *"invasive technology"* [Koh05b] – OMDoc-aware editing facilities are built into existing editing frameworks to make the most of existing functionalities and get around the learning curve involved with a new editor. Such approaches are detailed in Section 7.

In the `Sissi` implementation, the ontology $\mathcal{O}$ is represented in OMDoc and all the services $\mathcal{S}$ are built on top of this ontology.

sTeX

sTeX (2004) [Koh06b; Koh04] is a collection of macro packages that allow semantic markup in TEX or LATEX documents. sTeX is an extension that transforms TEX or LATEX into a document format for mathematical knowledge management. sTeX is, in certain regards, making use of the invasive technology approach – scientific and technical writers already use the TEX or LATEX formats to write down their work; rather than using a different system to integrate such documents with semantic systems, the authors can

use sTeX as an authoring format, staying in their own environments (both authoring and format wise) while still making use of semantic technologies.

With special tools (e.g.: LaTeXML [Gin09]), a transformation to a machine understandable format such as OMDoc is simple. Thus, sTeX becomes a strong authoring format for semi-formal ontologies.

In the `Sissi` implementation, the ontology $\mathcal{O}$ is written in sTeX and transformed into OMDoc.

TNTBase

In order to accommodate for the need of large scale collaborative authoring of mathematical documents in XML-derived formats, such as OMDoc, a storage system has been developed: TNTBase (2009) [ZhoKoh09]. TNTBase is a versioned XML database with a client-server architecture. It is based on Berkeley DB XML, but also integrates with a Subversion server. DB XML stores the HEAD (last) revision of each XML file, for easy access to the data, as well as operations on the HEAD revision (DB XML supports efficient querying via the DB XML API and XQuery). The non-XML files as well as differences from the previous revisions are stored in the SVN endpoint. Further details about this application can be found on its webpage [TNT]. This system is relevant because it can store arbitrary XML data (including OMDoc) and can perform operations on the stored XML documents.

In our setup of the `Sissi` implementation, TNTBase provides storage for the ontology and raw semantic services on top of it.

JOBAD

JOBAD (2009) (**J**avaScript API for **O**penMath **B**ased **A**ctive **D**ocuments) [JOBAD; Koh+09b] is a JavaScript library that allows the creation of added-value services on top of semantic information stored in semantically-annotated (X)HTML documents (MathML is now part of the latest HTML standard, before XHTML was the only way to embed combine MathML and HTML).

JOBAD adds semantic services to mathematical documents enhanced with semantic annotations (e.g.: the meaning of mathematical formula x^2 is the power function applied to arguments x and 2). These semantic services transform static HTML documents into active mathematical documents. JOBAD facilitates the access to the DOM (Document Object Model) for HTML documents and it allows user interaction additions via its module system.

JOBAD is a mashup enabler – one can create mash-ups for mathematical documents by coupling external services with semantically annotated documents – documented in [Dav+10; GicLanRab09; Dav10; CirGinLan11]. Examples of such mashups are: definition lookup (for a certain symbol in a

document), unit conversion of quantities and semantic navigation based on RDFa metadata.

JOBAD also follows the principles of invasive technology – using its services, users have access to e.g.: definition lookup inside their current context. For example, when a student reads about the definition of the mathematical concept of "monoid" (semigroup with identity property) and wants to check the definition of the semigroup concept (without leaving the current context).

TNTBase serves raw ontology fragments or semantic services based on the ontology. The served documents are enriched with JOBAD-provided services. These are meant to enhance the interactions between the user and `Theo`.

The Planetary System

The Planetary System [Koh+11] is based on all the aforementioned technologies and represents a vision towards Web 3.0, the Social and Semantic Web. It represents a math-enabled Web 3.0 information portal in which the dynamic, user generated content from Web 2.0 is enriched with semantic features. It has started in August 2010 as a redesign of the mathematics website PlanetMath [Plab], but it is a general software that can be instantiated by a user community to individual "planets" that aggregate content on a specific topic. This project is a concretization of the principles in the Semantic Web applied to math-dependant content. More details about this project can be found at [Plaa].

This system represents an aggregator of the above mentioned technologies: it provides an authoring environment for sTeX documents, automatic conversion to OMDoc and later to HTML, automatic connection to TNTBase (for storage and XML operations) and an active document enhancer provided by JOBAD.

The Planetary System makes use of the integration with all the technologies presented in this section in order to become the semantic service provider in Figure 2 for the `Semantic Alliance` framework.

5 A Validation of the `Semantic Alliance` Framework with `Sissi`

In this section we will look at the overall implementation of the `Semantic Alliance` framework, the goals it tries to reach and the design principles for the implementation. The current implementation looks at spreadsheets, so towards the end of the section, we will also inspect the relevant spreadsheet players used in this implementation.

5.1 Sissi: An Implementation of Semantic Alliance

To validate the Semantic Alliance architecture described in Section 2 and to make sure that the drawbacks of SACHS are avoided (dependendability on OS, document player and document player API) are avoided with this architecture, we have implemented it for spreadsheet systems (within our SiSsi project [SiSsi]). Concretely, we have implemented Sally and Theo for the desktop setup and we have built Alexes for LibreOffice Calc (3.4) [Hom], MS Excel (2010) and Google Docs Spreadsheet application.

To evaluate the new aspects of the Semantic Alliance framework (e.g.: support for multiple document types and players), we have implemented the Semantic Alliance architecture up to the level of reaching feature parity with the SACHS system (see Section 3.3). This setup allows us to evaluate the feasability of the framework, as well as its efficiency and extensionality. In particular, we reached for feature parity for two essential semantic services offered by the SACHS system for MS Excel 2003 spreadsheets:

- *Definition Lookup*
- *Semantic Navigation*

By implementing the integration of these two services in the user workflows, we use all the components in and outside the Semantic Alliance framework and prove that the integration for further, more complicated services can be achieved.

In SACHS, for "Definition Lookup", the user selects a cell and the user will see displayed the definition of the term that cell is tied to (from the background ontology via the semantic map). For this purpose, a MS Excel native popup would show up close to the chosen cell. If the definition would contain references to other concepts that are semantically linked to cells in the spreadsheet, then the "Semantic Navigation" service (enabled in SACHS' dependency graph display option) allows the user to navigate to the respective cell on click of the according graph node. The evaluation will be with regards to the actual expenditures of reaching that level.

Before we go through the implementation and framework component analysis (in Section 6), we will take a look at the factors that influenced the implementation decisions (in Section 5.2) in Semantic Alliance.

The Semantic Alliance framework implementation this paper addresses can be found online at https://svn.kwarc.info/repos/sissi/branches/catalin.

5.2 Requirements Analysis

In this section, we will look at the implementation specific essential principles that we want our framework to achieve. In general, this can be considered

as a "lessons learned" design strategy, based on the experience gained from the implementation of the SACHS and CPoint projects.

In particular, we want the `Semantic Alliance` framework to:

Support Multiple Operating Systems: In contrast to SACHS and CPoint, that only worked on Windows, we want to provide a consistent "semantic experience" to users of multiple operating systems, in order to foster the adoption of such systems.

Support Multiple Document Types: In the current project setup, we only focus on spreadsheets, but for the future, we want to be able to build a semantic ally for any document type, whether it is a CAD/CAM system or a presentation system. Additionally, we want to achieve that with the smallest degree of investment – rather than reimplementing the entire ecosystem, focus only on developing the necessary, document-type specific interaction handling and on integrating further semantic services into the architecture.

Support Multiple Document Players: For each document type, there are multiple document players (e.g.: `MS Excel`, `LibreOffice Calc`, `Google Docs` and others for spreadsheet systems). Unfortunately, they are usually very different: they don't have the same API, the same language to write the extensions in etc. Still, the same services should be available for the users, regardless of their choice of document player. Moreover, the services should be document player transparent: the integration should be done once per document type, rather than for each document player. This allows the easy addition of further document players per document type, by adding only a new `Alex` component for a specific document player. In comparison to SACHS, that only worked with `MS Excel` 2003, we want to have the same services available for a range of spreadsheet document players, specifically `LibreOffice Calc` 3.4, `MS Excel` 2010 and `Google Docs` in our evaluation experiment.

Provide a coherent user interface and user experience across document players:
Besides having the same services across multiple document players, the user interface and experience should be the same. This makes it easier to design an integration and a user interaction once and reuse it across document players and operating system, providing the user with the same overall experience.

These objectivess generate some technical requirements for the components presented in Section 2:

***`Sally` has to be Operating System (OS) independent*:** In order to have the semantic ally run on any computer, it should be operating system

independent. Programming languages that run on top of a virtual machine (e.g.: Java) seem to be the most attractive.

***Theo has to be OS independent*:** We want the users to have the same experience across operating systems, so it is essential that the displayed information is in the same place and looks the same.

***Theo should support latest standards in terms of mathematics and graphics rendering*:**
At least for a mathematics heavy domain, such as spreadsheets, it is a good idea to have a strong mathematics rendering engine. Also, the dependency graph requires a strong graph rendering engine to display complex mathematical structures properly. In general, based on this and the previous requirement, when aggregating data from web services, we want no constraints from the choice of renderer. Since we are aggregating mostly *web* services, a good idea would be to use a strong browser rendering engine (that supports MathML, SVG for mathematics and graphics and HTML5 in general).

Alex should be as thin as possible . `Alex` is the only part that is document player dependent and that facilitates the communication between the document player and `Sally`. Every document player has its own implementation of `Alex`. Therefore, it is important that this component is as small as possible and that an abstraction over the operations it provides exists in `Sally`, while the implementation is document player specific.

5.3 Spreadsheet Players

In this section, we will give an overview of the different spreadsheet players, from a user's perspective, used in our implementation: `MS Excel` 2010, `LibreOffice Calc` 3.4 and `Google Docs` Spreadsheet.

Microsoft Excel 2010 `MS Excel` 2010 is part of the Microsoft Office productivity suite provided by Microsoft.

The main advantage of `MS Excel` 2010 is that it is a polished product – Microsoft Office was launched 21 years ago, with the current version being the 14^{th} release of the software. As a testimony to that, Microsoft Office is believed to have the largest market share on Windows and Mac operating systems (though no recent figures exist). This position as a market leader is favorized by the initial adoption rate and the high degree of use in business settings.

Microsoft Office 2007 brings an innovative design to the UI of all Office products – the Ribbon UI. This UI consists of a static height, multi tabbed

bar at the top of the application window, filled with graphical representations of control elements, grouped by different functionality. According to Microsoft, this interface has the purpose to simplify the User Interface and reduce the number of actions required to achieve a task. This new UI was received with mix reviews, but most affected by this were power users who were accustomed to the traditional interface (menus, icons, bars). According to Microsoft, the number of actions required to achieve a task has reduced in most cases and the interface is cleaner.

As a negative aspect, `MS Excel` can not be purchased by itself, but it is part of the Microsoft Office productivity suite which is proprietary and commercial software.

LibreOffice Calc 3.4 `LibreOffice Calc` is part of the `LibreOffice` productivity suite provided by the Document Foundation.

`LibreOffice Calc`'s strong points are that it is free and open source and that it supports multiple Operating Systems (OS). This makes it attractive for users, as the documents will look similar, regardless of the operating system. As an extra, it can be used on Linux-based operating systems, where Microsoft Office is not available. Also due to their Java-based implementation, the user interface is the same, regardless of the OS.

A potential drawback is that `LibreOffice Calc` is only partially compatible with the `MS Excel` standards, so documents authored in `MS Excel` might look different in `LibreOffice Calc`.

Google Docs Spreadsheets `Google Docs` is the online productivity suite offered by Google.

One of the advantages of `Google Docs` is that this platform is free for personal use, but it is not open source and it is not free for corporate use. Another big advantage is that the users do not need to install a full productivity suite on their computers – the documents can be authored in any recent web browser. Also due to this fact, Google Docs provide the same experience, regardless of the computer that the user is employing.

What really sets `Google Docs` apart from the desktop spreadsheet players is the "social" experience – the same `Google Docs` document can be authored by multiple people at the same time. Google also makes it easy to administer and share the document with other people. An interesting feature of `Google Docs` in the context of "online editing" is that the users can author the documents while they are offline and have them synced to the Google servers as soon as they are online (via an extension). In this way, Google achieves the functionality of desktop spreadsheet players, but also allows Social Web functionality.

One of the possible drawbacks of Google Docs is the license agreement – Google (as a company) has access to the data in `Google Docs` (but Google

personnel does not) in order to improve its own services.

5.4 Planetary System as Semantic Service Provider

For the `Sissi` implementation of `Semantic Alliance`, we use the Planetary system [Koh+11; Plaa] (presented in Section 4) as the underlying semantic service provider. This component facilitates access to the ontology $\mathcal{O}$ and the services $\mathcal{S}$ that are provided for `Sally`. We will now look at the details of these components and how they are used in the current implementation.

`Ontology` $\mathcal{O}$ The ontology is usually shared by multiple users, therefore it requires some initial effort since it needs to be designed well, such that it can represent all the real world objects, as well as the relations between them. Also, it should allow reusage of data and accommodate later extensions in an easy manner. In addition to that, some services require a high degree of formalism (e.g.: automatic verification of CAD/CAM components) and mathematics (e.g.: prediction of values in spreadsheet systems), while others only require a semi-formal ontology. With a varying degree of formalism, the ontology has a great impact on the services provided. A semi-formal, but structured ontology allows for services such as definition lookup and semantic navigation (to other concepts), while a formal ontology coupled with existing software can provide services such as change management (e.g.: how will a change in one document affect other items in the document and possibly other documents), automatic verification (e.g.: a CAD/CAM design conforms to certain ISO standards).

In our current setup, the ontology $\mathcal{O}$ is a collection of files authored in sTEX, which are transformed to OMDoc and stored in TNTBase. TNTBase indexes them by semantic functional criteria and can then perform server-side semantic services. The Planetary instance is tightly connected to TNTBase and will relay the semantic service queries to TNTBase.

`Services` $\mathcal{S}$ The services provided are based on the ontology and the semantic link that ties document elements to the ontology. In the current work, we pursue only two services: definition lookup and semantic navigation.

6 `Semantic Alliance` Components in `Sissi`

In the following section we describe the realized components of the `Semantic Alliance` framework in the `SiSsI` project, which show the feasability of the framework.

6.1 Sally– Merging Interfaces and Interactions

Sally is the central piece of the Semantic Alliance framework: it interacts with Alex, Theo and the semantic service provider (for the ontology $\mathcal{O}$ and the services $\mathcal{S}$)[5]. As one can see in Figures 2 and 3, the Semantic Alliance framework is designed for both a desktop use and a mobile (web-based) use. These two flavors of the architecture differ only in the communication environment to their respective Theo and possibly Alex. In order to satisfy the cross operating-system requirement, Sally is developed in Java.

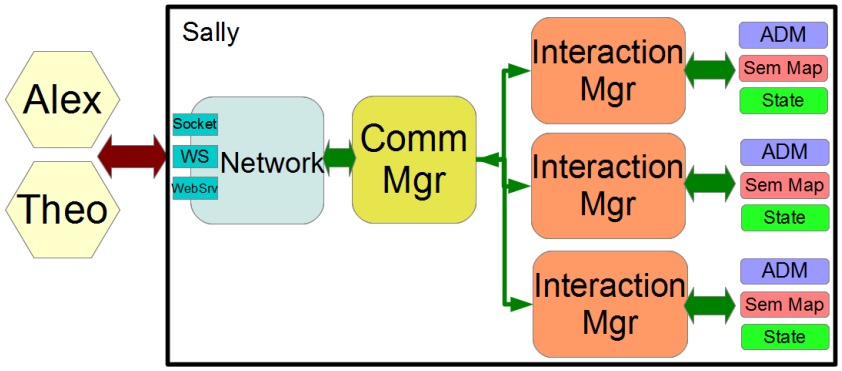

Figure 7: Sally

Sally is built with a clear separation of tasks in mind, with an architecture based on layers. In order to achieve its role as a semantic ally, it needs a layer that coordinates interfaces and interactions for an Alex– Theo pair: the **Interaction Manager**. In order to be able to communicate with all the other components of the Semantic Alliance framework, Sally needs a networking part, provided by the **Network** layer. The Interaction Manager is an abstract Sally layer that manages interactions for a certain document type – in particular, there are multiple document-type specialized instances for each Alex– Theo pair. Since we have multiple Interaction Managers and only one Network endpoint (with the separation of tasks in mind), we need another layer that transfers information from the Network layer to the necessary Interaction Manager (and vice versa): the **Communication Manager**. We will now analyze the Sally layers of our architecture (the basic ones described here, as well as others, all presented in Figure 7):

Network Since the architecture runs according to a client-server model, with Sally being the server, the other components of the Semantic Alliance framework, Theo and Alex, need to be able to connect to and communicate with Sally.

[5]We assume here that $\mathcal{S}$ runs as a web service and is accessed via the Internet. Still, even a local setup can be done for offline or single-computer situations.

Therefore, in the current implementation we have the following possible communication mechanisms for `Alex` and `Theo`:

***Sockets*:** `Sally` is available via one of the basic communication mechanisms on a computer network.

***WebSockets*:** In some constrained environments, the use of sockets might be disabled (e.g.: web browsers and possibly mobile applications). A new protocol, WebSocket, tries to allow socket-like communication between a web server and a web client (usually, web browser). In our implementation, `Theo` is constrained to use WebSockets (see Section 6.2 for details) to connect to `Sally`.

***Web Server*:** In case `Alex` or `Theo` can't communicate through the above mentioned mechanisms, other approaches exist in order to achieve almost real-time bidirectional communication, as detailed in [BozMesDeu07]. We use the HTTP poll approach in one of our setups (see Section 6.3).

Due to the fact that there are multiple network mechanisms, the Network layer provides an abstraction on top of them (e.g.: start server, send message) that can be used later on by the Communication Manager (which is not aware of the underlying communication details).

Communication Manager In our architecture, the Communication Manager is a stateless layer that provides the functionality of a router: it transforms network messages into an internal representation and then forwards them to the appropriate Interaction Manager (and vice versa).

Besides the router functionality, the Communication Manager has the role of initially coupling an `Alex` connection to an appropriate `Theo` connection (note that there might be multiple types of `Theo`) and creating an Interaction Manager with these elements.

Interaction Manager The Interaction Manager is the coordination and decision layer in this architecture – based on the current status and user interaction events, it decides what the next steps are and it coordinates the actions of the other components of the `Semantic Alliance` framework.

The Interaction Manager is a stateful layer; it keeps track of the current status of the interaction between the user and `Theo`, as well as the relevant user interactions within the document player (reported to `Sally` via `Alex`). With this knowledge, it can coordinate `Theo` to e.g.: display a certain semantic service upon a user click, and `Alex` to e.g.: move the cursor in a spreadsheet in the Semantic Navigation service. Because it can instruct `Theo` to display windows on top of the $\mathcal{A}$ window, the Interaction Manager, and therefore `Sally`, becomes a screen area manager for application $\mathcal{A}$. The

Interaction Manager coordinates interactions in two dimensions: the interactions between the `Semantic Alliance` framework components, as well as the interactions between the user, `Semantic Alliance` components and semantic services.

Each instance of the Interaction Manager couples an `Alex` and a `Theo` and coordinates their actions based on user interaction. Since `Sally` is responsible for interactions with semantic objects, it also holds a representation of the **semantic map** (a collection of semantic links) – even though this component belongs to the document, the document player knows nothing about semantics and therefore `Sally` is responsible for updating it. Alongside with that, `Sally` also holds a representation of the spreadsheet in internal data structures: an **Abstract Document Model** described below (in the current setup, for spreadsheets only). For semantic services or actions that require multiple steps or which are conditioned by a previous action, `Sally` also keeps track of the **interaction state**. The Interaction Manager keeps a close connection with these `Sally` components and based on their information, it decides the next steps (all components are presented in Figure 7).

Abstract Document Model (ADM)

For each document type, there are multiple document players. Different document players might have different understandings of what an object is in their model (e.g.: material, processes and dimensions might all vary in different implementations of CAD/CAM systems). Moreover, as noted in the previous sections, the document player might not expose semantic objects to the user. That is why it is important for `Sally` to have an abstract document model (ADM) that comprises a unified representation of both generic document (in our case spreadsheets) and semantic objects in own data structures, for each document type it supports. We call objects in the ADM **abstract objects**. Abstract objects are mapped to spreadsheet objects (**real objects**). Using the semantic link that makes the connection between a real object and an ontology term, we can also map abstract objects to ontology terms. Diagram in Figure 8 shows how these components are connected.

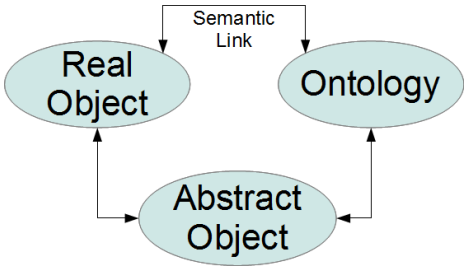

Figure 8: The connection between document objects, ADM and Ontology terms

6.2 `Theo`– Managing Screen-Area

The main purpose of the `Theo` component is to render HTML5 semantic services in the right place, upon request by `Sally`. In the current setup, this component is realized as an instance of XULRunner [XULb] – the naked layout and communication engine behind Mozilla Firefox and Thunderbird. The layout of the interface items is given in the XUL format [XULa] and the interactions are handled via JavaScript.

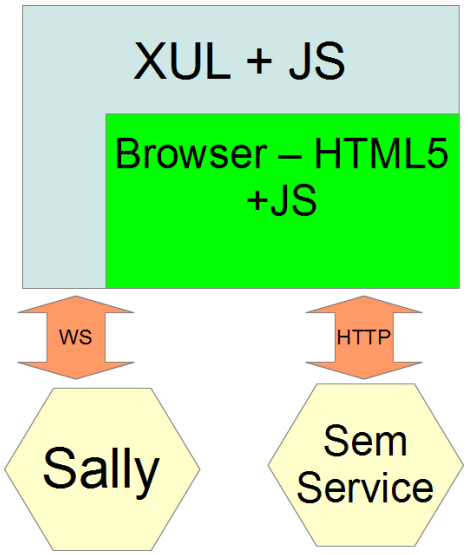

Figure 9: `Theo`

For math-heavy applications, it is important that `Theo` allows HTML5 presentation of both interface items (buttons, text boxes etc.) and HTML content (as web pages). Note that `Theo` can be considered to be just a renderer for `Sally`, which sends content, placement and size information via WebSockets[6]. Note that the communication channel is bidirectional: `Theo` events (e.g.: clicking on a node in a dependency graph) are communicated to `Sally` which then coordinates the appropriate reaction.

`Theo` is usually just a placeholder for contextual information coming from web services. In such cases, the main element of the XUL window is just a browser window (a good abstraction is a Firefox window with no decorations) which displays the information at a certain URL. The URL is usually pointed to a semantic service provider offering the service $\mathcal{S}$. Any relevant action taken on that webpage (e.g.: clicks on URLs) is transmitted to `Sally`. Again, due to security constraints, the semantic service provider can't talk directly to `Sally`, but the communication must go through the XUL window, as we can see in Figure 9.

6.3 `Alex`– Document Player Invader

`Alex` is the invasive part of our architecture. It is developed for document players that have extension support and it is used for exporting and managing requested UI events. `Alex` uses multiple layers of the document player API (presented in Section 2.4) to transmit relevant information to `Sally` and to modify the document according to `Sally` instructions (see Figure 10).

[6]The natural communication via sockets is prohibited by XULRunner for security reasons; WebSockets provide a safer abstraction that we can use in this context

The invasive design requirement of the `Semantic Alliance` framework is that the employment of semantic services is done from within the respective application. For the implementation of this requirement, we can differentiate here between two types of users:

Casual Users: `Sally` is running on the local computer; in this case `Alex` has to take care of starting and stopping the semantic ally.

Business Users: `Sally` is running on the remote machine and it is made available to multiple users; in this case, `Alex` only has to connect to the remote `Sally` instance. This case is more interesting in a business setting, where the ontology $\mathcal{O}$ and semantic services $\mathcal{S}$ are shared across multiple users.

The other pylon of this architecture is Semantic Illustration. As we noted before, the document, together with the semantic map, is meant to be self contained; this implies that the semantic map is stored alongside the document. `Alex` takes care of storing the semantic map, as well as transmit this information to `Sally` at the beginning of the communication (more information about the communication workflows can be found in Section 6.4). `Sally` is the semantic ally and in our implementation, it is responsible for all semantic interactions. This allows `Alex` to be thin and have only two responsabilities:

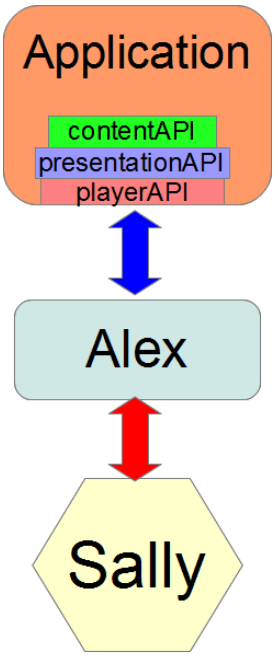

Figure 10: `Alex`

- reports cell click events in the application $\mathcal{A}$ to `Sally` together with the cell's position information (X and Y coordinates in pixels) and the user's display option (definition lookup or graph exhibition) – needed for contextualized definition lookup – and
- move the cursor to a cell when requested from `Sally`– needed for semantic navigation.

Depending on future semantic interactions based on other semantic objects, the UI listeners and UI actions might need to be extended. Still, the new listeners will be only expansions of the listeners that are already in place, while the actions will strictly concern the semantic objects.

In the following we will look in detail at the `Alex` extension for `MS Excel` 2010, `LibreOffice Calc` 3.4 and `Google Docs` Spreadsheets.

`Alex` for `MS Excel` 2010

`MS Excel` is part of the Office productivity suite provided by Microsoft. It is a spreadsheet application that works on both Windows and Mac operating systems.

The `Alex` for `MS Excel` 2010 has been developed on top of the `.NET` framework provided by Microsoft. It is implemented in C# and it uses the well documented Microsoft Office API and the `.NET` API. So far, with the current implementation, two issues were present during the development:

Focus: Whenever the `MS Excel` window was notified by the operating system that it lost focus, the visual markers for the selected cell and column would disappear. This makes it hard to achieve invasive design, as the user loses track of the semantic object he interacted with. This is due to technical reasons: the `Theo` window is overlayed on top of the `MS Excel` window and receives focus, thus making the `MS Excel` window lose focus. A workaround for this was to manually emphasize the semantic object by adding and removing borders.

Position Information: The `MS Excel` API does not provide access to the size of the header indicator bars (they provide the user the column / row indication: [A,B,...] and [1,2,...]). This is important because the screen position of a cell is computed by summing three values: `MS Excel` window offset from the (0,0) corner of the desktop plus grid offset within the window plus cell offset in the grid. The information about the grid offset within the window is computed based on the height of the Ribbon bar and the size of the indicator bars (A,B$\cdots$, 1,2,$\cdots$). Unfortunately, the `MS Excel` API provides no way to access the size of the bars, so the values are hardcoded. More information about this can be found at [MSD], where the author contacted Microsoft user support.

`Alex` for `OpenOffice Calc`

`OpenOffice Calc` is part of the open-source productivity suite provided by The Document Foundation [Theb]. The codebase behind `LibreOffice` application is old, it is actually a branch of the `OpenOffice` productivity suite. Due to the fact that they are so similar code-wise (the split between the code was done recently), as well as the fact that `Alex` is not building on intrcate API features, the `Alex` extension can be run in both environments successfuly.

The framework behind `OpenOffice` was designed such that there is a separation between functionality and UI, i.e. content and form. Therefore, `OpenOffice` follows a client-server architecture, in which the GUI is just a client asking for functionality from the server. Therefore, extensions can be

either functionality extenders – server level, require no GUI (e.g.: automatically convert any Office document to PDF) – or more common, that expose server-side functionality via a GUI. The extensions are based on the UNO (Universal Network Objects) IDL (Interface Description Language) component model which is designed to offer interoperability between different programming languages. This allows extensions to be built in several programming languages: mainly Java and OOBasic, but there is also a Python API (for server-side functionality).

The lack of documentation and the complicated object model have led to a longer development time for this component. Still, the problems encountered in the `MS Excel` counterpart are not to be found here.

`Alex` for `Google Docs` Spreadsheets

`Google Docs` Spreadsheet is an alternative to spreadsheet players that has as a strong point the fact that one does not need to install any application (the spreadsheet player is on a website) and also the capability to collaboratively author a document. The `Google Docs` framework is based on the Etherpad project [Inc] which aims to make it easy to collaboratively author text documents online. This aspect of the `Google Docs` framework is tightly related to the server-client system architecture: the servers store and update the data, while the clients (web browsers) allow users to make changes to the data. Essentially, the client is just a presentation layer for the data provided by the server. Again, we can observe the separation between content and form, from a framework perspective.

`Google Docs` can be extended using the Google Apps Script [Gooa] in the same way that desktop spreadsheet players (such as `MS Excel` and `OpenOffice Calc` can be extended via extensions). The language for Google Apps Script is JavaScript and it is meant to unify the development language across `Google Docs` and `Google Apps Engine`.

In the next section, we will look at what constraints such a setup imposes and how we can overcome them with a variation of the architecture.

Constraints and a New Architecture Unfortunately, there are some drawbacks that prove to be important for the design of an `Alex` component for `Google Docs`:

***Restricted JavaScript*:** The language that can be used to enhance the `Google Docs` spreadsheets is only a subset of the JavaScript functionality commonly implemented in web browser engines (e.g.: *setTimeout* function is not available); this, coupled with other drawbacks presented here (API constraints) prohibit crucial desired functionality.

***No Bidirectional Communication*:** The connection between `Alex` and `Sally` should be bidirectional – this is needed because actions within

`Theo` can result in changes within the application $\mathcal{A}$. While in `MS Excel` and `OpenOffice Calc` this communication was achieved through sockets and a similar web-friendly alternative (WebSockets), there is no implementation in the `Google Docs` API for either sockets or WebSockets (due to security reasons). Still, `Google Docs` API provides an HTTP request mechanism that is used in this case to simulate bidirectional communication via polling (continuously sending requests to the server).

***Scripts for Data, not Browser*:** The clear difference between the document player and document data (aside from the task distribution) leads to the following design decision: user authored Google Apps Scripts are not running inside the browser, but they are running on the Google servers hosting the data and not inside the browsers. This leads to the lack of a `playerAPI` (detailed later in this section) and this is a crucial missing feature of `Google Docs` for the current work (see Section 2.4). This also introduces the issue of latency – the communication between the browser and the Google server is influenced by current network conditions. This means that scripts might be already executed on the server, but the result would not be visible in the user's browser.

***Time Limit*:** All JavaScript scripts devised by the user are restricted to a runtime of 5 minutes. This means that, even though `Alex` should be omnipresent, it has to be stopped every 5 minutes and restarted. This is possibly dangerous, as `Alex` might be in the middle of an operation (e.g.: move the cursor) when the time limit expires, requiring it to constantly be aware of this time limit.

***Secure Environment*:** The Internet is not a secure environment. The most important type of attack to personal information on the client side (e.g.: browser) while surfing the Internet is a Cross Site Scripting (XSS) attack. This attack usually consists of running rogue JavaScript on another website with the potential effect of transmitting sensitive data to another website (the rogue script provider). In order to restrict XSS issues, `Google Docs` has completely disabled the feature of embedding HTML code.

With these constraints in mind, a new architecture was created (see Figure 11). In this case, the `Alex` component is located and executed on the Google servers. This requires that `Sally` is accessible from the outside world. For this purpose, we have

- built a simple HTTP client into `Sally` that listens for web page requests, and

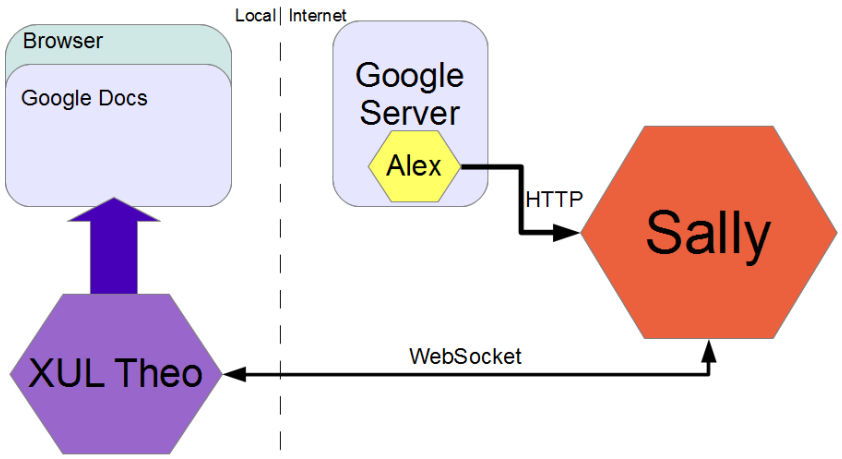

Figure 11: `Semantic Alliance` for `Google Docs`

- deployed `Sally` on a machine that can be reachable from the Internet.

While in the case of the desktop applications, it was either `Alex` or `Sally` that started `Theo` (depending on the setup), in this case we can assume that the browser in which the user is viewing the spreadsheet does not have enough privileges/functionality to start another application (this is due to security reasons – it would then be easy for a browser to start any application and access personal data). Therefore, in this implementation, the user has to start `Theo` by hand and instruct it to connect to the remote `Sally`. The communication between the Google servers and the browser is relied on for the transmission of API commands (e.g.: move the cursor).

With this setup, we have the basic components set up, the communication is bidirectional (almost), so we can move on to analyze the functionality this setup offers.

Functionality Another difference that is important to realize is that in the `Google Docs` model, there is a clear separation between the document player and the data. This, coupled with the possible latency described above make this noticeable for the user (even though `Google Docs` tries to hide it via asynchronous requests). Moreover, this separation leads to missing parts of the API. Note that in order to achieve Semantic Illustration via Invasive Design, we need all three components of an Open API (see Section 2.4):

`contentAPI``:` Allows access to the data in the cells

`presentationAPI``:` Allows access to the presentation of cells (e.g.: background color)

`playerAPI``:` Allows access to UI events, controls and position information

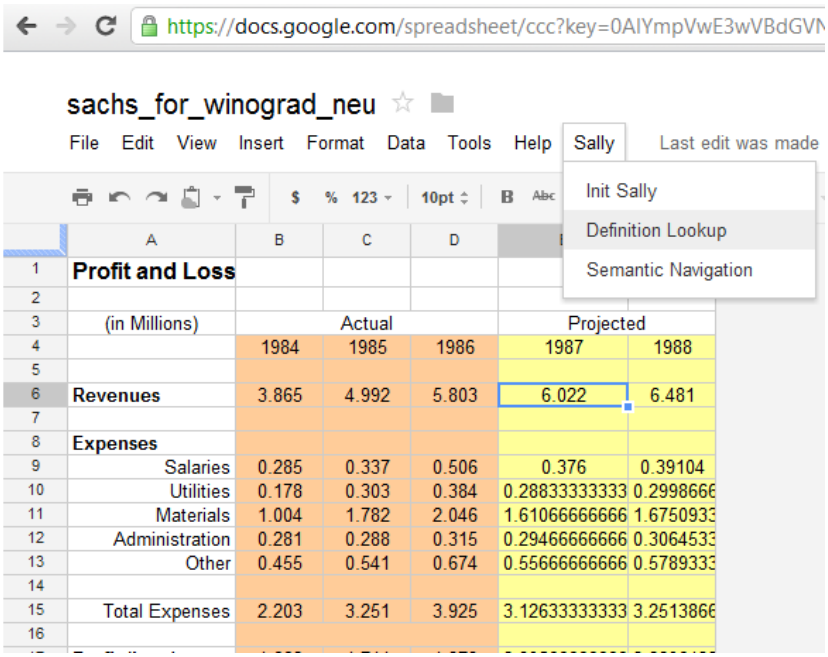

Figure 12: Hardcoded Menu for `Google Docs`

Google Apps Script provides access to a `contentAPI` and `presentationAPI`– since this information resides on the Google servers, it is easily accessible for the script. Unfortunately, the `playerAPI` is very restricted for our needs:

***No Cell Click Event*:** The Google Apps Script API does not provide a method to subscribe to the cell click event. This is important because without this we can not trigger the semantic interactions. Instead, a hardcoded list of services was built into the `Google Docs` UI and the user is required to select a cell and then select a menu entry (see Figure 12).

***No Position Information*:** This is the most important missing feature of `Google Docs`. The lack of this feature means that Invasive Design can not work any more: the semantic information is contextual (in terms of information), but it is not contextual in placement – the window will appear at an arbitrary position. The lack of this feature is mainly caused by:

***Distributed Architecture of `Google Docs`*:** most of the API functionality regards interaction with the data which happens on the server. Little functionality is currently provided to access/control the layouting engine. Even if that were possible, due to

***Security Issues*,** one could not find the position of a certain HTML element on the user screen, but only within the current tab (not

even the browser window). This information should not be available to JavaScript (the active part of an HTML page), as it is considered to be a security risk.

We can conclude that an `Alex` for `Google Docs` implementation exists, with development costs higher than in the desktop applications (due to the addition of a web server), but this implementation does not satisfy our requirements for Invasive Design due to API constraints.

In the next part, we will look at how the components of the `Semantic Alliance` framework interact.

6.4 Interaction Analysis

In the `Semantic Alliance` framework the communication between `Alex`, `Sally` and `Theo` relies on the fact that there is a clear separation of concerns. The interaction between the `Semantic Alliance` components is mainly driven by the interaction between the user and the spreadsheet player or between the user and `Theo`. Here, we will look in detail at how the components of `Semantic Alliance` framework interact in order to provide semantic services to the user and through which phases the system goes, as well as the components involved in each phase.

Note that the framework is invisible to the user – the user perceives it as a semantic ally tightly integrated with the document player (while within the framework, the semantic ally tasks are handled by `Sally`). This motivates the decision for UI elements labeled "Sally" – related to the user's perception.

Startup This is the initial step that needs to be done in order to establish communication between the `Semantic Alliance` framework components and make sure everything is functional. Therefore, `Alex` has to provide ways in which the user can start/stop other `Semantic Alliance` components (read below) and the communication with the respective components (in a clean way), without the user realizing it. For that purpose, `Alex` adds a *Start Sally* toggle button to the user interface of the document player.

The setup and user base dichotomy presented before (i.e.: casual versus business users) influences the system setup and therefore this initial step:

Business Users: `Sally` lives in the "cloud" and is shared by multiple users. In this case, only the `Alex` and `Theo` components of the `Semantic Alliance` architecture are on the user's machine. This means that `Alex` has to initialize the communication with `Sally` and it has to start `Theo` and make it connect to `Sally`.

Casual Users: All components of the architecture (including `Sally`) live on the local computer. In this case, to keep `Alex` thin, we have decided to assign the task of starting `Theo` to `Sally`. This is the implementation for which I have opted, but it is not hard to switch to the other setup with the current framework.

Up to this point, all components are started and `Alex` and `Theo` establish a connection to `Sally`. Still, `Sally` does not know yet where the connections come from or what components they represent.

Identify In this phase, the `Alex` and `Theo` components that have connected to `Sally` will identify themselves to `Sally`. This is important for the framework because we want the semantic ally to support multiple `Theo` and `Alex` components (of multiple types), as well as multiple document types.

Here, the `Alex` identifies itself to `Sally` with the document type (in our setup a spreadsheet) and the preferred type of `Theo` component (in our setup a XUL `Theo`). The `Theo` component also identifies itself to `Sally`, but only with the type of rendering it offers for now (e.g.: XUL based or HTML based).

Once both `Alex` and `Theo` are connected, `Sally` couples these two components (based on their identification information) with an Interaction Manager (described previously) and determines a list of available services

Semantify Some documents might have a semantic map stored inside. Since `Sally` is responsible for all semantic interactions (including adding, updating or deleting semantic links), it is important that it is aware of the existing semantic map and takes responsibility for it. In this phase, `Alex` sends to `Sally` the semantic map that the current document has.

User Interaction Once `Sally` takes responsibility for the semantic map, the only thing left to do is for `Alex` to listen to UI events in the spreadsheet player.

Let's assume the user clicks on a spreadsheet cell (or range of cells). Since `Alex` has access to the `playerAPI`, it can subscribe and handle such events. When such an event is triggered, it transmits information to `Sally`:

***Cell Identifier*:** The position of the cell in the current spreadsheet and the spreadsheet name. This information is necessary since the semantic link is built upon it (and `Sally` needs to find the ontology term this cell points to) and this is the semantic object the user interacted with (and is therefore relevant to `Sally` for other services). This information is provided in the A1 style notation (e.g.: $SpreadsheetName$C3)

***Cell Position*:** The coordinates of this cell on the user screen. This information is necessary for the achievement of Invasive Design – it enables

`Theo` to position a window on top of a spreadsheet player, in a location near to where the user clicked (see Figure 13 for an example).

The Network layer in `Sally` receives this information and it is passed to the Interaction Manager layer. This layer decides the next steps. Based on previous interactions between the user and the system, there are two possibilities:

Default Service: This means that with every interaction, `Sally` instructs `Theo` to display the default service.

No Default Service: In this case, `Sally` instructs `Theo` to show a window with the list of all available services. The user picks a service by clicking on it and this information is relayed back to `Sally`. `Sally` then requires the selected service and instructs `Theo` about the position and information that needs to be shown.

Now, the user can interact with both the spreadsheet player and `Theo`. In order to preserve the illusion of invasiveness, it is crucial that both user interaction elements are synchronized. For that purpose, any relevant UI event in either `Theo` or $\mathcal{A}$ will be reported to `Sally`. The Interaction Manager in `Sally` then decides which are the actions to be taken (e.g.: a click on a concept URL in a `Theo` window with the purpose of semantic navigation should also reposition the cursor in $\mathcal{A}$ to a cell which is linked to the new concept).

It is important to realize that both `Alex` and `Theo` contain no semantic logic – they just report events to `Sally` who takes all the decisions. In this regard, `Theo` can be considered to be a "renderer" for `Sally`, as it displays information sent from `Sally`. This achieves a true separation between content and form, with `Sally` taking care of the content that should be rendered, while `Theo` only displays this information with a certain layout.

In the following section, we will look at how these interactions help to achieve Semantic Illustration through Invasive Design for our validation.

6.5 Discussion

To gain an intuition on the runtime-behavior of the `Semantic Alliance` framework, let us look at the new realization of SACHS functionality described earlier. In the concrete examples, we reuse the SACHS ontology already described in [KohKoh09b], but adapted to the new setting (stored in TNTBase, semantic services offered by The Planetary System).

Definition Lookup & Semantic Navigation in `Sissi` In the `Sissi` implementation (see Figure 13), `Alex` responds to a click of cell [E9] by requesting a **definition lookup** window from `Sally`, which requests an

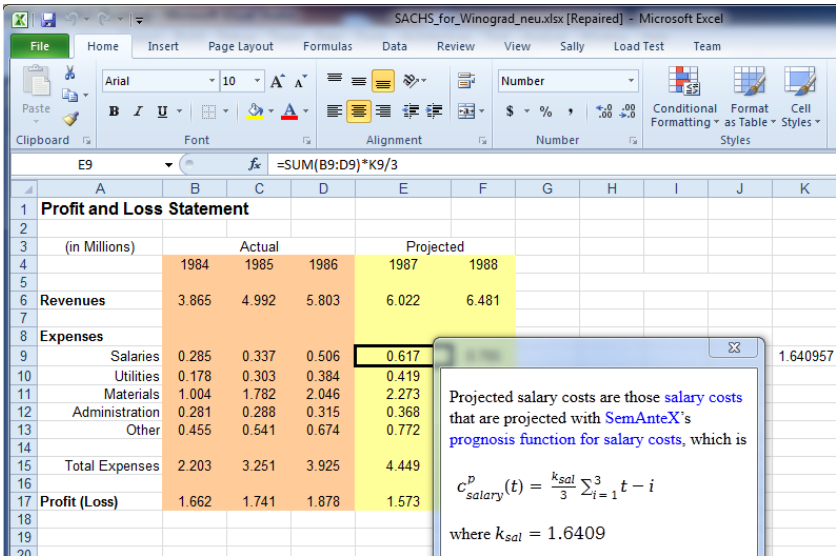

Figure 13: Definition Lookup in `Sissi`

HTML5 document from the semantic service provider $\mathcal{S}$, on whose arrival `Theo` overlays the $\mathcal{A}$-GUI at the appropriate location with a browser window containing the requested information.

The definition lookup or the dependency graph service is invoked, as of now, by the user's selection in the "Sally menu" (visible in Fig.13 resp. 14). As this is an interaction management task, it naturally belongs to `Sally`'s tasks, so in the near future interaction configuration will be enabled via `Sally` itself.

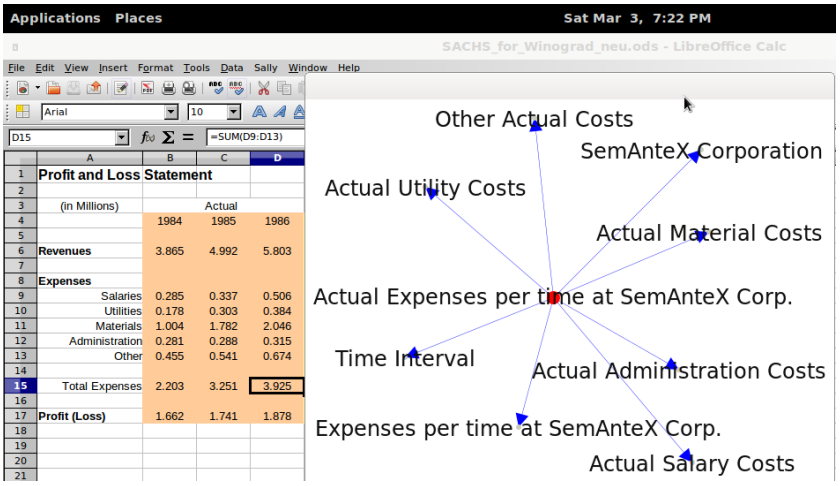

Figure 14: Semantic Navigation in `Sissi`

Now, let us look at Figure 14, which presents the **dependency graph** of cell [D15], that is, the graph of "Actual Expenses per time at SemAnteX Corp". The nodes in this graph are hyperlinks to their (via the semantic map) associated cells. For example, the concept "Actual Utility Costs" from $\mathcal{O}$ is associated with cell range [B10,D10] on the same worksheet. If the user clicked this node, then `Theo` – which is in control of the window that displays the dependency graph – reports this click event to `Sally`, that interprets it as a navigation request and relays this to `Alex`, which in turn moves the cursor to cell [D10] in the GUI of $\mathcal{A}$ (based on the evident heuristic). Note that semantic navigation works workbook wide. Figure 14 is a screenshot taken from a `LibreOffice Calc` document on a Linux machine, which verifies `Sissi`'s platform- and application independence.

This means that we reached feature parity wrt. SACHS' definition lookup and semantic navigation as described. But even with these very limited `Alex` prototypes we already obtain some semantic features that go beyond the old SACHS implementation:

- The JOBAD interaction framework (see Section 4) employed in Planetary allows to make the interactions semantically interactive by embedding semantical services in them as well. For instance, if a fragment of the definition lookup text (e.g. "salary costs" in Figure 13) is linked to an ontology concept that is itself associated with a cell in this workbook (in our example, "salary costs" are associated with the cell range [B9,D9]), then semantic navigation (which was only available via concept graphs in SACHS) comes for free.

- The SACHS system was severely limited by the fact that native `MS Excel` 2003 popups are limited to simple text strings, which made layout, formula display, and interactivity impossible. In the `Semantic Alliance` framework, we have the full power of dynamic HTML5[7] at our hands. In particular, OpenMath formulae can be nicely rendered (see Figure 13), which gives spreadsheet users visual (and navigational) support in understanding the computation (and the provenance) of the respective values.

- Planetary has an integrated editing facility that can be used for building the background ontology as an invasive service and with a little bit more work on `Sally`, it can be used for managing the semantic map.

Note that in the `Semantic Alliance` architecture, `Sally` is a reusable component, which implies that Planetary services offered via `Sally` are open to other applications $\mathcal{A}$ as long as they are extended by a suitable (thin) `Alex`.

[7]The full power of dynamic HTML5 means in particular HTML+CSS for text layout, MathML for formula display, SVG/Canvas for diagrams, and JavaScript for interactivity.

To conclude our discussion let us see how our initial claim that the `Semantic Alliance` framework allows the rapid deployment of semantic services in applications holds up to the implementation experiences. We can look at `Alex` for `MS Excel` and its development costs: Even though the author was not familiar with the `MS Excel` .NET backend, he succeeded building `Alex` for `MS Excel` in about 40 hours. Actually, much of this time was not spent on realizing the basic functionality, but on getting around idiosyncrasies of the `MS Excel` system. As a confirmation, a similar development time was needed for `OpenOffice Calc`– most of the development time was used to achieve simple, but undocumented tasks, based on the experience of other users shared in public web forums. The Google Docs implementation took more time due to the architectural changes, but not more than 80 hours. The validation of the framework consists in demonstrating its feasibility and confirming its claimed value, the nice cost-benefit ratio, which is exactly based on low API development costs.

7 Related Work

We have presented the `Semantic Alliance` framework as a mashup enabler for semantic services with already existing applications, that allows to use those services from within these applications to overcome users' potential motivational bootstrap hurdles, i.e., that yields Invasive Design. Here, we want to portray `Semantic Alliance`'s contribution by assessing related work. In general, we look at the main ways to achieve the integration of semantic information and services into non-semantic documents, resulting in semantic documents. In particular, we review existing relevant (semantic) extensions of document players resp. documents and the frameworks used with respect to our architecture.

One approach to obtain semantic documents is to extend the existing document formats with support for semantic information and services. This approach has been used in the "Semantic Web" movement, a movement that aims to add semantic content to web pages such that computers can understand more about the content presented. Standards such as RDFa [RDF; W3C08] have emerged that try to standardize how the additional information will be encoded within the webpage and several tools [RDF10] have been designed just for that. This approach only partially works since only newer browsers support such features, and the benefits are not obvious to the user. The same backwards compatibility issues appear in most document types. More importantly, while in the case of HTML, the World Wide Web Consortium (W3C) decides on the standard that all browsers should adopt, there might not be such regulating organizations for each document type, possibly resulting in many different, conflicting standards. The `Semantic Alliance` framework is built such that it can adapt to any document type,

regardless of their version/variant, and that it is dependent on the document player only in a small portion.

Another option to achieve the aforementioned integration is to design a framework that enables the creation of mashups, a *mashup enabler*. In recent years, a big variety of mashup enablers was created. "Greasemonkey" [Pil05] is a well-known example for a client-side extension of a web browser. In particular, it is a Firefox extension that allows to write scripts to persistently alter web pages for a user on the fly. A variation of the "Greasemonkey" idea is JOBAD – a mashup enabler for semantic services, as presented in [Dav10]. The idea behind JOBAD is of a framework in which developers can build services that mashup the current semantic document with other information (e.g.: in the case of the work presented, with Wolfram—Alpha [Wol]). But Greasemonkey as well as JOBAD and those other mashup enablers are limited to offer resulting web services. With the `Semantic Alliance` architecture a very different kind of service is enabled: an application-based service. Note that this application might be a web app, but can also be a desktop-centered component of an office suite.

One option to integrate semantic information with data is the creation of an entire semantic system, in parallel to the document player. This method allows the creation of semantic data for particular document types. Such an approach is considered in the RightField system [Rig], used in bioinformatics. This system allows users to embed ontology annotation within spreadsheet documents, on any operating system, but not as part of a document player workflow (due to the fact that it is a separate system). I have argued against such an approach because users are not willing to switch from their own environment to another system (due to possible interoperability issues, cognitive issues etc.). The `Semantic Alliance` framework integrates semantic services directly into the applications and normal workflows of the user via Invasive Design.

A direct application of the work done for the `Semantic Alliance` framework was the ability to integrate semantic services into Office applications in an invasive way. The SACHS project stands as a precursor of the current project, but it is built for `MS Excel` 2003 and it can not be extended for other document types or at least document players. CPoint is a content editor built for Microsoft PowerPoint – it allows the transformation of implicit knowledge stored in a PowerPoint presentation into explicit knowledge that both humans and the computer can understand. Both these projects achieve the goals of invasive technology in the eyes of the user – the user thinks he is interacting with the host application, but they do not provide a framework for integrating more services, more document types or even more document players. The `Semantic Alliance` framework overcomes these issues and provides an extensible framework for mashup enablers, for multiple document types and multiple extensible document players.

8 Further Work and Conclusion

I have presented the Semantic Alliance architecture and software framework that allows the user from within different standalone document players to use a semantic ally service (or semantic ally for short). From the user's point of view, he uses specialized semantic allies tailored to the respective application, whereas from a technical perspective, the main component of the framework is a single, universal semantic ally Sally mashing up distinct semantic services with the respective application's GUI. This is possible due to an innovative task distribution in Semantic Alliance based on a combination of the Semantic Illustration architecture and Invasive Design.

In particular, the application-specific parts of a service are outsourced to Alexes, which are just responsible for managing the application's UI events and thus can be built thin. In our reference implementation of Alex for MS Excel, the development merely took a week. The rendering parts of a service are executed by Theos. Sally, the technical semantic ally, acts as an interaction manager between the components and services on the one side and the user on the other. As such it requires most development effort and time, but once implemented, this functionality can be reused across document players, as well as across document types.

In the current work we also describe the implementation of Alex components, for MS Excel and OpenOffice Calc as desktop spreadsheet players. The setup for Google Docs spreasheets is also presented, as a web browser based document player. The goal is to achieve the same results as in the desktop setup, but due to the limitations of the extension language, Google dependent setup and the API offered by Google Docs, the development is hindered and the results are no match for the ones in the desktop setup, i.e.: Invasive Design can not be attained.

The Sally implementation in Sissi is realized in Java, integrating much of the semantic functionality via web-services, and our Theo is browser-based. Therefore, the setup of Semantic Alliance is fully operating-system independent. So far Sally is used to interact with different types of spreadsheet programs, but in the future we want to exploit this to incorporate different applications with semantic allies. This will be simple for the other elements of the OpenOffice and Microsoft Office suite, hence, we are looking forward to blend semantic services between different document types in the near future. Moreover, work is currently under way on an Alex for a CAD/CAM system as envisioned in [Koh+09a]. It is expected that the main work there relies in the establishment of an abstract document model (which resides in Sally and is shared across semantic allies for CAD/CAM systems) and the respective background ontology.

The Semantic Alliance framework is built with extensionality in mind – it is not constrained that all components run on the same machine. As the Semantic Alliance framework is loosely coupled to the semantic ser-

vice provider, a multitude of semantic services can be offered. This makes the `Semantic Alliance` framework very attractive for semantic technology integration. For instance, in the `SiSsi` project for which we have developed the `Semantic Alliance` framework, we will use `Sally` to integrate verification of spreadsheet formulae against (formal) specifications in the background ontology or test them against other computational engines. In CAD/CAM systems the illustration mapping can be used to connect CAD objects to a bill of materials in the background ontology, which in turn can be used to verify physical properties (e.g. holding forces). Computational notebooks in open-API computer algebra systems like Mathematica or Maple can be illustrated with the papers that develop the mathematical theory. Theorem provers can be embedded into MS Word ... — the opportunities seem endless.

References

[AbrErw06] Robin Abraham and Martin Erwig. "Inferring templates from spreadsheets". In: *ICSE '06: Proceedings of the 28th international conference on Software engineering*. Shanghai, China: ACM, 2006, pp. 182–191. ISBN: 1-59593-375-1. DOI: `http://doi.acm.org/10.1145/1134285.1134312`.

[Act] ACTIVEMATH. URL: `http://www.activemath.org`.

[Ama03] Uwe Aßmann. *Invasive software composition*. Springer, 2003, I–XII, 1–334. ISBN: 978-3-540-44385-8.

[AndNov08] Oscar D. Andrade and David G. Novick. "Expressing Help at Appropriate Levels". In: *Proceedings of the 26th annual ACM international conference on Design of communication*. Ed. by Carlos J. Costa et al. ACM Special Interest Group for Design of Communication. ACM Press, 2008, pp. 125–130.

[Aus+10a] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 3.0*. Tech. rep. World Wide Web Consortium (W3C), 2010. URL: `http://www.w3.org/TR/MathML3`.

[Aus+10b] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 3.0*. W3C Recommendation. World Wide Web Consortium (W3C), 2010. URL: `http://www.w3.org/TR/MathML3`.

[BozMesDeu07] E. Bozdag, A. Mesbah, and A. van Deursen. "A Comparison of Push and Pull Techniques for AJAX". In: *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on*. Oct. 2007, pp. 15 –22. DOI: `10.1109/WSE.2007.4380239`.

[BraPet08] Richard Brath and Michael Peters. "Spreadsheet Validation and Analysis through Content Visualization". In: *CoRR* abs/0803.0166 (2008).

[Bus+03] Stephen Buswell et al. *The* OpenMath *Standard, Version 2.0 Public Draft 4*. Tech. rep. The OpenMath Society, 2003. URL: `http://www.openmath.org/standard/om20-2003-11-24/`.

[Car+09] Jacques Carette et al., eds. *MKM/Calculemus Proceedings*. LNAI 5625. Springer Verlag, July 2009. ISBN: 978-3-642-02613-3.

[Cin] *Cinderella: Interactive Geometry Software*. URL: `http:\\www.cinderella.de` (visited on 02/24/2012).

[CirGinLan11] Mihai Cîrlănaru, Deyan Ginev, and Christoph Lange. "Authoring and Publishing of Units and Quantities in Semantic Documents". In: *The Semantic Web: ESWC 2011 Workshops*. Workshops at the 8th Extended Semantic Web Conference (ESWC) (Hersonissos, Crete, Greece, May 29–30, 2011). Ed. by Raúl García Castro, Dieter Fensel, and Grigoris Antoniou. Lecture Notes in Computer Science 7117. Heidelberg: Springer Verlag, 2011, pp. 202–216.

[Cis] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011-2016*. URL: `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf` (visited on 05/15/2012).

[Dav+10] Catalin David et al. "JOBAD/MMT – Interactive Mathematics". In: *AI Mashup Challenge*. Ed. by Adrian Giurca et al. June 2010. URL: `http://sites.google.com/a/fh-hannover.de/aimashup/home/jobad`.

[Dav10] Catalin David. "Interactive Documents as Interfaces to Computer Algebra Systems: JOBAD and Wolfram—Alpha". B. Sc. Thesis. Jacobs University Bremen, 2010. URL: `https://svn.eecs.jacobs-university.de/svn/eecs/archive/bsc-2010/cdavid.pdf`.

[Dav+12] Catalin David et al. "`Semantic Alliance`: A Framework for Semantic Allies". In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics (CICM) (Bremen, Germany, July 9–14, 2012). Ed. by Johan Jeuring et al. LNAI 7362. Springer Verlag, 2012. URL: `http://kwarc.info/kohlhase/submit/mkm12-SAlly.pdf`. In press.

[Deu] *Deutsche Forschungsinstitut Für Künstliche Intelligenz*. seen March 2007. URL: http://www.informatik.uni-bremen.de/dfki-sks/.

[Din09] Matthew Dinmore. "Documenting Problem-Solving Knowledge: Proposed Annotation Design Guidelines and their Application to Spreadsheet Tools". In: *CoRR* abs/0908.1192 (2009).

[Dub] *Dublin Core Metadata Initiative*. web page at http://www.dublincore.org. seen March 2008. URL: http://www.dublincore.org.

[EUS10] EUSPRIG. *European Spreadsheet Risks Interest Group*. home page at http://www.eusprig.org. 2010. URL: http://www.eusprig.org.

[GicLanRab09] Jana Giceva, Christoph Lange, and Florian Rabe. "Integrating Web Services into Active Mathematical Documents". In: *MKM/Calculemus Proceedings*. Ed. by Jacques Carette et al. LNAI 5625. Springer Verlag, July 2009, pp. 279–293. ISBN: 978-3-642-02613-3. URL: https://svn.omdoc.org/repos/jomdoc/doc/pubs/mkm09/jobad/jobad-server.pdf.

[Gin09] Deyan Ginev. "An Architecture for Recovering Meaning in a LaTeX to OMDoc Conversion". Bachelor's Thesis. Jacobs University Bremen, 2009. URL: https://svn.eecs.jacobs-university.de/svn/eecs/archive/bsc-2009/dginev.pdf.

[Gooa] *Google Apps Script Project*. URL: https://developers.google.com/apps-script/ (visited on 05/30/2012).

[Goob] *Google Knowledge Graph*. URL: http://www.google.com/insidesearch/features/search/knowledge.html (visited on 05/23/2012).

[Gooc] *Google Search for "height of Eiffel Tower"*. URL: https://www.google.com/search?q=height+of+eiffel+tower (visited on 05/23/2012).

[HarJan09] Melanie Hartmann and Frederik Janssen. *LWA 2009; Workshop-Woche: Lernen – Wissen – Adaptivität*. Tech. rep. Universität Darmstadt, Sept. 2009.

[Hea+] Tom Heath et al. *Linked Data – Connect Distributed Data across the Web*. URL: http://linkeddata.org (visited on 06/11/2010).

[HilKohSta06] Eberhard Hilf, Michael Kohlhase, and Heinrich Stamerjohanns. "Capturing the Content of Physics: Systems, Observables, and Experiments". In: *Mathematical Knowledge Management (MKM)*. Ed. by Jon Borwein and William M. Farmer. LNAI 4108. Springer Verlag, 2006, pp. 165–178. URL: `http://kwarc.info/kohlhase/papers/mkm06physml.pdf`.

[HodMit08] Karin Hodnigg and Roland T. Mittermeir. "Metrics-Based Spreadsheet Visualization: Support for Focused Maintenance". In: *CoRR* abs/0809.3009 (2008).

[Hom] *Home of the LibreOffice Productivity Suite*. `http://www.libreoffice.org`. seen Feb. 2012. URL: `http://www.libreoffice.org`.

[HutHolNor85] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. "Direct manipulation interfaces". In: *Hum.-Comput. Interact.* 1.4 (Dec. 1985), pp. 311–338. ISSN: 0737-0024.

[Inc] AppJet Inc. *Etherpad*. URL: `http://www.etherpad.com` (visited on 09/08/2011).

[JOBAD] *JOBAD Framework – JavaScript API for OMDoc-based active documents*. URL: `http://jobad.omdoc.org`.

[Joh10] Jeff Johnson. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules*. Morgan Kaufmann Publishers, 2010.

[Koh] Michael Kohlhase. *OMDoc Version 1.3*. URL: `http://trac.omdoc.org/OMDoc/wiki/OMDoc1.3` (visited on 12/09/2010).

[Koh04] Michael Kohlhase. "Semantic Markup for TEX/LATEX". In: *Mathematical User Interfaces Workshop 2004*. Ed. by Paul Libbrecht. 2004. URL: `http://www.activemath.org/~paul/MathUI04`.

[Koh05a] Andrea Kohlhase. "Overcoming Proprietary Hurdles: CPoint as Invasive Editor". In: *Open Source for Education in Europe: Research and Practise*. Ed. by Fred de Vries et al. Proceedings at `http://hdl.handle.net/1820/483`. Open Universiteit Nederland. Heerlen, The Netherlands: Open Universiteit Nederland, Nov. 2005, pp. 51–56. URL: `http://hdl.handle.net/1820/483`.

[Koh05b] Andrea Kohlhase. "Overcoming Proprietary Hurdles: CPoint as Invasive Editor". In: *Open Source for Education in Europe: Research and Practise*. Ed. by Fred de Vries et al. Proceedings at `http://hdl.handle.net/1820/483`. Open Universiteit Nederland. Heerlen, The Netherlands: Open Universiteit Nederland, Nov. 2005, pp. 51–56. URL: `http://hdl.handle.net/1820/483`.

[Koh06a] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: `http://omdoc.org/pubs/omdoc1.2.pdf`.

[Koh06b] Michael Kohlhase. "sTEX: A LATEX-Based Workflow for OMDoc". In: OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. Chap. 26.15. URL: `http://omdoc.org/pubs/omdoc1.2.pdf`.

[Koh07] Andrea Kohlhase. "Semantic PowerPoint: Content and Semantic Technology for Educational Added-Value Services in MS PowerPoint". In: *Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications 2007 (ED-MEDIA'07)*. Ed. by Craig Montgomerie and Jane Seale. AACE, June 2007, pp. 3576–3583. URL: `http://go.editlib.org/p/25890`.

[Koh+09a] Michael Kohlhase et al. "Formal Management of CAD/CAM Processes". In: *16th International Symposium on Formal Methods (FM 2009)*. Ed. by Ana Cavalcanti and Dennis Dams. LNCS 5850. Springer Verlag, 2009, pp. 223–238. URL: `http://kwarc.info/kohlhase/papers/fm09.pdf`.

[Koh+09b] Michael Kohlhase et al. "JOBAD – Interactive Mathematical Documents". In: *AI Mashup Challenge*. Ed. by Brigitte Endres-Niggemeyer, Valentin Zacharias, and Pascal Hitzler. Sept. 2009. URL: `https://svn.omdoc.org/repos/jomdoc/doc/pubs/ai-mashup09/jobad.pdf`.

[Koh10] Andrea Kohlhase. "Towards User Assistance for Documents via Interactional Semantic Technology". In: *Proceedings of the 33.rd Annual German Conference on Artificial Intelligence KI'10*. Ed. by Rüdiger Dillmann et al. LNAI 6359. Karlsruhe, Germany, 2010, pp. 107–115.

[Koh+11] Michael Kohlhase et al. "The Planetary System: Web 3.0 & Active Documents for STEM". In: *Procedia Computer Science* 4 (2011): *Special issue: Proceedings of the International Conference on Computational Science (ICCS)*.

Ed. by Mitsuhisa Sato et al. Finalist at the Executable Papers Challenge, pp. 598–607. DOI: 10.1016/j.procs.2011.04.063. URL: https://svn.mathweb.org/repos/planetary/doc/epc11/paper.pdf.

[KohKoh08] Andrea Kohlhase and Michael Kohlhase. "Semantic Knowledge Management for Education". In: *Proceedings of the IEEE; Special Issue on Educational Technology* 96.6 (June 2008), pp. 970–989. URL: http://kwarc.info/kohlhase/papers/semkm4ed.pdf.

[KohKoh09a] Andrea Kohlhase and Michael Kohlhase. "Compensating the Computational Bias of Spreadsheets". In: *Festschrift in Honour of Bernd Krieg-Brückner's 60th Birthday*. Ed. by Berthold Hoffmann et al. DFKI, 2009, pp. 184–200.

[KohKoh09b] Andrea Kohlhase and Michael Kohlhase. "Compensating the Computational Bias of Spreadsheets with MKM Techniques". In: *MKM/Calculemus Proceedings*. Ed. by Jacques Carette et al. LNAI 5625. Springer Verlag, July 2009, pp. 357–372. ISBN: 978-3-642-02613-3. URL: http://kwarc.info/kohlhase/papers/mkm09-sachs.pdf.

[KohKoh09c] Andrea Kohlhase and Michael Kohlhase. "Modeling Task Experience in User Assistance Systems". In: *Proceedings of the 27th annual ACM international conference on Design of communication (SIGDOC)*. (Bloomington, Indiana, USA, 2009). Ed. by Brad Mehlenbacher et al. ACM Special Interest Group for Design of Communication. New York, NY, USA: ACM Press, 2009, pp. 135–142. DOI: 10.1145/1621995.1622013. URL: http://kwarc.info/kohlhase/papers/sigdoc09-taskexperience.pdf.

[KohKoh09d] Andrea Kohlhase and Michael Kohlhase. "Semantic Transparency in User Assistance Systems". In: *Proceedings of the 27th annual ACM international conference on Design of communication (SIGDOC)*. (Bloomington, Indiana, USA, 2009). Ed. by Brad Mehlenbacher et al. ACM Special Interest Group for Design of Communication. New York, NY, USA: ACM Press, 2009, pp. 89–96. DOI: 10.1145/1621995.1622013. URL: http://kwarc.info/kohlhase/papers/sigdoc09-semtrans.pdf.

[KohKoh09e] Andrea Kohlhase and Michael Kohlhase. "What you get is what you understand: Assessment in SACHS". In: *Wissens- und Erfahrungsmanagement (Knowledge and Experience Management), FGWM*. Ed. by Christoph Lange and Jochen Reutelshöfer. Workshop at LWA 2009, published as part

of [HarJan09]. Sept. 2009, pp. 22–29. URL: http://www.kwarc.info/kohlhase/papers/lwa09-sachs.pdf.

[KohKoh10] Andrea Kohlhase and Michael Kohlhase. "What we understand is what we get: Assessment in Spreadsheets". In: *Symp. of the European Spreadsheet Risks Interest Group (EuSpRIG 2010)*. Ed. by Simon Thorne. European Spreadsheet Risk Interest Group, 2010, pp. 111–121. ISBN: 978-1-905404-50-6. URL: http://www.kwarc.info/kohlhase/papers/eusprig10-coverage.pdf.

[KohKoh11] Andrea Kohlhase and Michael Kohlhase. "Spreadsheets with a Semantic Layer". In: *Electronic Communications of the EASST: Specification, Transformation, Navigation – Special Issue dedicated to Bernd Krieg-Brückner on the Occasion of his 60th Birthday* (2011). Ed. by Till Mossakowski, Markus Roggenbach, and Lutz Schröder. accepted. URL: http://kwarc.info/kohlhase/papers/easst11.pdf.

[KohSuc06] Michael Kohlhase and Ioan Şucan. "A Search Engine for Mathematical Formulae". In: *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*. Ed. by Tetsuo Ida, Jacques Calmet, and Dongming Wang. LNAI 4120. Springer Verlag, 2006, pp. 241–253. URL: http://kwarc.info/kohlhase/papers/aisc06.pdf.

[Lan07] Christoph Lange. "Towards Scientific Collaboration in a Semantic Wiki". In: *Bridging the Gap between Semantic Web and Web 2.0 (SemNet)*. Ed. by Andreas Hotho and Bettina Hoser. June 2007.

[LanKoh06] Christoph Lange and Michael Kohlhase. "A Semantic Wiki for Mathematical Knowledge Management". In: *Proceedings of the 1st Workshop on Semantic Wikis, European Semantic Web Conference*. (Budva, Montenegro, June 12, 2006). Ed. by Max Völkel, Sebastian Schaffert, and Stefan Decker. CEUR Workshop Proceedings 206. Aachen, 2006. URL: http://ceur-ws.org/Vol-206/.

[Log] *Logosphere: a Formal Digital Library*. web page at http://www.logosphere.org. seen November 2006. URL: http://www.logosphere.org/.

[Mata] *Mathcad: Optimize your design and engineering*. URL: http://www.ptc.com/products/mathcad (visited on 02/14/2012).

[Matb] *Mathematica*. URL: http://www.wolfram.com/products/mathematica/ (visited on 06/05/2010).

[McGHar04] Deborah L. McGuinness and Frank van Harmelen. *OWL Web Ontology Language Overview*. W3C Recommendation. World Wide Web Consortium (W3C), Feb. 10, 2004. URL: http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[Meh+09] Brad Mehlenbacher et al., eds. *Proceedings of the 27th annual ACM international conference on Design of communication (SIGDOC)*. (Bloomington, Indiana, USA, 2009). ACM Special Interest Group for Design of Communication. New York, NY, USA: ACM Press, 2009. DOI: 10.1145/1621995.1622013.

[Mel+03] E. Melis et al. "Knowledge Representation and Management in ActiveMath". In: *International Journal on Artificial Intelligence and Mathematics, Special Issue on Management of Mathematical Knowledge* 38.1–3 (2003), pp. 47–64.

[MMT] Florian Rabe. *MMT – A Module system for Mathematical Theories*. URL: http://trac.kwarc.info/MMT/ (visited on 08/18/2010).

[MSD] *MSDN Social: Cell absolute screen position*. URL: http://social.msdn.microsoft.com/Forums/en-US/exceldev/thread/50dae716-598c-4b16-bd57-4e09b859c83e (visited on 05/23/2012).

[Mul06] Normen Müller. "OMDoc as a Data Format for VeriFun". In: OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. Chap. 26.20, pp. 329–332. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[NieYanNov03] Jason Nieh, S. Jae Yang, and Naomi Novik. "Measuring thin-client performance using slow-motion benchmarking". In: *ACM Trans. Comput. Syst.* 21 (1 Feb. 2003), pp. 87–115. ISSN: 0734-2071.

[OMDoc] Michael Kohlhase. OMDoc*: An open markup format for mathematical documents (latest released version)*. Specification, http://www.omdoc.org/pubs/spec.pdf. URL: http://www.omdoc.org/pubs/spec.pdf.

[Ove] *Over 5 billion mobile phone connections worldwide*. URL: http://www.bbc.co.uk/news/10569081 (visited on 05/15/2012).

[Pan00] Raymond R. Panko. "Spreadsheet Errors: What We Know. What We Think We Can Do." In: *Symp. of the European Spreadsheet Risks Interest Group (EuSpRIG 2000)*. 2000.

[Pil05] Mark Pilgrim. *Greasemonkey Hacks: Tips & Tools for Remixing the Web with Firefox (Hacks)*. O'Reilly Media, Inc., 2005. ISBN: 0596101651.

[Plaa] *Planetary Developer Forum*. URL: `http://planetary.mathweb.org/` (visited on 05/06/2012).

[Plab] *PlanetMath.org – Math for the people, by the people*. URL: `http://planetmath.org` (visited on 09/08/2011).

[PowLawBak08] Stephen G. Powell, Barry Lawson, and Kenneth R. Baker. "Impact of Errors in Operational Spreadsheets". In: *CoRR* abs/0801.0715 (2008).

[RDF] *RDFa*. URL: `http://rdfa.info` (visited on 11/08/2011).

[RDF10] *RDFa Tools*. Aug. 3, 2010. URL: `http://rdfa.info/wiki/?title=Tools&oldid=1162` (visited on 08/27/2010).

[Rig] *RightField*. URL: `http://www.sysmo-db.org/rightfield` (visited on 05/29/2012).

[SiSsi] *Software Engineering for Spreadsheet Interaction*. Project Homepage. URL: `http://trac.kwarc.info/sissi/`.

[Thea] *The Mobile Internet Report*. URL: `http://www.morganstanley.com/institutional/techresearch/mobile_internet_report122009.html` (visited on 05/15/2012).

[Theb] *The Document Foundation*. URL: `http://www.documentfoundation.org/` (visited on 05/23/2012).

[TNT] *TNTBase TRAC*. `http://tntbase.org`. URL: `http://tntbase.org`.

[Vri+05] Fred de Vries et al., eds. *Open Source for Education in Europe: Research and Practise*. Proceedings at `http://hdl.handle.net/1820/483`. Open Universiteit Nederland. Heerlen, The Netherlands: Open Universiteit Nederland, Nov. 2005. URL: `http://hdl.handle.net/1820/483`.

[W3C08] World Wide Web Consortium (W3C), ed. *RDFa Primer*. `http://www.w3.org/TR/xhtml-rdfa-primer/`. 2008. URL: `http://www.w3.org/TR/xhtml-rdfa-primer/`.

[Win06] Terry Winograd. "The Spreadsheet". In: *Bringing Design to Software*. Ed. by Terry Winograd et al. Addison-Wesley, 2006, pp. 228–231.

[Wol] *Wolfram—Alpha*. URL: http://www.wolframalpha.com (visited on 05/05/2011).

[XULa] *XUL language*. URL: https://developer.mozilla.org/en/XUL (visited on 01/30/2012).

[XULb] *XULRunner Runtime Environment*. URL: https://developer.mozilla.org/en/XULRunner (visited on 02/29/2012).

[ZhoKoh09] Vyacheslav Zholudev and Michael Kohlhase. "TNTBase: a Versioned Storage for XML". In: *Proceedings of Balisage: The Markup Conference*. Vol. 3. Balisage Series on Markup Technologies. Mulberry Technologies, Inc., 2009. DOI: 10.4242/BalisageVol3.Zholudev01.