SAlly: A Framework for Semantic Allies

Catalin David, Constantin Jucovschi, Andrea Kohlhase, Michael Kohlhase

Computer Science, Jacobs University Bremen
http://kwarc.info

Abstract. We 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, our SAlly framework follows an invasive approach: Users can profit from semantic technology without having to leave their accustomed workflows and tools. On the other hand, SAlly offers a largely application-independent way of extending existing (open API) applications with MKM technologies. The SAlly framework presented in this paper consists of three components: i.) a universal semantic interaction manager for given abstract document types, ii.) a set of thin APIs realized as invasive extensions to particular applications, and iii.) a set of pseudo-invasive renderer components for existing semantic services. We validate the SAlly approach by instantiating it with a spreadsheet-specific interaction manager, thin APIs for LibreOffice Calc 3.4 and MS Excel'10, and a browser-based renderer.

1 Introduction

A major research interest in the field of Mathematical Knowledge Management (MKM) consists in the development of semantic technologies: they can be prototyped on mathematical knowledge 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]). 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 MKM technologies are typically very specialized in the kind of service they offer and very good at that.

The main disadvantage of standalone MKM 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 that 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 the 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 MKM systems.

In our earlier research on the adoption issue (cf. Semantic Authoring Dilemma; see [Koh05]), we have argued for the creation of *invasive semantic technology*, i.e., a technology, where the semantic services are embedded in (figuratively: invade) the host application, so 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 [HHN85] and Software-Engineering-driven re-use ideas in [Aßm03]. To this end we developed the semantic LaTeX authoring environment "SiTeX" [JK10; KKL10], or the `MS Excel`'03 add-in "`SACHS`" [KK11] as a semantic extension of a spreadsheet document.

In our latest project "`SiSsI`" (Software Engineering for Spreadsheet Interaction) we now want to transfer the semantic functionalities in `SACHS` to more spreadsheet systems: Spreadsheet users are locked into different applications like `LibreOffice Calc`, `MS Excel`'10, or GoogleDocs for reasons beyond our control. But offering `SACHS` functionality as *invasive* technology, every new application would induce a development effort similar to the original one for `SACHS`, as the functionality has to be re-created in differing application contexts and programming languages. Moreover, architectural differences like the ones between desktop applications, server-based web applications or even mobile apps, pose radically different solutions for accessing the background ontology or visualizing its knowledge (e.g. in a graph viewer). Differing security issues complicate the picture even further.

The central idea to address these issues is based on a combination of the Semantic Illustration architecture in [KK09] with a new approach towards invasive design. This gives rise to an innovative framework for semantic extensions that we present 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 MKM system contribute semantic services and interactions to the host user interface, managed by a 'semantic ally'. In Section 2 we first elaborate on the combination of Semantic Illustration and invasive design, followed by a presentation and discussion of the "`SAlly`" architecture, which allows to build semantic allies, reusing MKM components and technologies to drive down the cost.

In section 3 we validate and report on our experiences with a first implementation "`Sissi`" of this `SAlly` framework. We review related work in Section 4 with respect to (semantic) extensions of document players and compare it with `SAlly`. In Section 5 we summarize `SAlly` and draft upcoming projects.

2 The SAlly Framework

First we recap the conceptual underpinnings of our Semantic Illustration approach and sketch invasive design as an efficient replacement for invasive technology. Then we combine both approaches into a framework for semantic allies: SAlly. Taking the standpoint of SAlly being a mashup enabler in Subsection 2.2, conceptual tasks and responsibilities of each component within SAlly are marked and explained.

2.1 Invasive Design via Semantic Illustration

In the **Semantic Illustration** [KK09] architecture semantic technology is made fruitful by "illustrating" existing software artifacts semantically via a mapping into a structured background ontology $\mathcal{O}$. We consider an underlying "illustrator" and interaction manager a **semantic ally**. 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**; see [KK09] for a thorough discussion of the issues involved.

We combine that with a new approach to software design we call **invasive design** to obtain the same effect as invasive technology does. We observe that a service $\mathcal{S}$ feels embedded into an application $\mathcal{A}$ if it occupies a screen-area $\mathcal{D}_\mathcal{S}$ that is part of the area $\mathcal{D}_\mathcal{A}$ originally claimed by the application itself: if the screen area $\mathcal{D}_\mathcal{S}$ 'belongs' to $\mathcal{A}$ in this way and the service $\mathcal{S}$ was requested by the user from within $\mathcal{A}$, then the user *perceives* $\mathcal{S}$ as an application-dependent service $\mathcal{S}_\mathcal{A}$, see for example Chapter 1:"*We perceive what we expect*" in [Joh10]. This perception is amplified, if a service and its request refer to the local semantic objects, we speak of "**contextualized**" services if they make use of this effect (compare with, e.g. [MDD09]).

In semantic allies the semantic link between $\mathcal{A}$ and $\mathcal{O}$ drawn on by semantic services $\mathcal{S}$ given by Semantic Illustration provides such a contextualization. In particular, $\mathcal{S}$ does not need to be implemented as application-specific invasive technology. From a technological standpoint, a *thin* client (see e.g. [NYN03]) invading $\mathcal{A}$ is sufficient to obtain the advantages of invasive technology in the eyes of the user. This means especially, that development costs can be drastically reduced. The only condition for $\mathcal{S}$ consists in being able to strictly outsource application-dependent parts. Note that this is again an instance of the never-ending quest for the separation of content and form, therefore semantic services should not have big difficulties in adhering to this premise.

Thus, the combination of Semantic Illustration and invasive design results in a **framework for semantic allies**, which we call "SAlly". In a nutshell, it has three components:

- a platform-independent semantic interaction manager "Sally" (as a semantic ally), that has access to contextualizable semantic services, and that
- partners with a set of invasive, thin, application-specific "Alex$_\mathcal{A}$", that essentially only manage user interface events in $\mathcal{A}$, and that

arch.tex 272 2012-03-04 16:35:46Z ako

– has access to a set of application-independent screen area managers "$\texttt{Theo}_n$" that can render the available services.

Note that we restrict our attention here to applications which come in the form of a "document player", i.e., 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 Web2.0 systems.

2.2 The SAlly Framework as a Mashup Enabler

We have observed above that functionalities of a semantic extension system can be realized with invasive design. The SAlly framework introduced in this paper is based on the additional observation that the pertinent semantic extensions are already largely implemented in web-based Mathematical Knowledge Management systems (wMKM), and that semantic allies can be realized by mashing up wMKM systems with the original application. But in contrast to traditional mashups, which integrate web data feeds in a broad sense into web portals, the SAlly framework mashes up the GUIs of wMKM systems and applications themselves. In this sense, the SAlly framework can be considered a **mashup enabler**, a system that transforms otherwise incompatible IT resources into a form where they can be combined.

Let us assume $\mathcal{S}$ to be a wMKM system drawing on an ontology $\mathcal{O}$.[1] In the SAlly framework, the task of the mashup enabler between $\mathcal{A}$ and $\mathcal{S}$ then is split into three parts (see Figure 1).

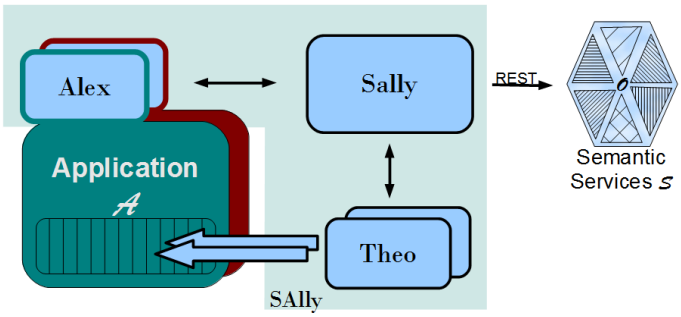

Fig. 1. SAlly as a Mashup Enabler for Semantic Allies

[1] For the sake of the exposition we assume that $\mathcal{S}$ runs as a web service and is accessed via the Internet, but nothing hinges on this; for offline situations, $\mathcal{S}$ could be started locally. However, since $\mathcal{O}$ is probably be biggest investment necessary for enabling semantic services via Sally, a shared, web-accessible mode of operations for $\mathcal{S}$ is probably the most realistic until we have distributed MKM systems.

`Sally` The main part of the mashup enabler is realized in the "`Sally`" component which integrates the functionalities of $\mathcal{A}$ and $\mathcal{S}$ into a joint user interface and interaction model and thus realizes the semantic extension functionality for $\mathcal{A}$. `Sally` can draw on an implementation of abstract document types, which on the one hand abstract from the particulars of the data structures in $\mathcal{A}$ and on the other hand tie particular interactions to (abstract) document fragments. For instance, a click on a cell in a spreadsheet should lead to a different reaction than a click to a textbox in a presentation.

`Alex` The application $\mathcal{A}$ is extended by a slim API "`Alex`"[2] that reports and executes relevant user interactions with $\mathcal{A}$ like cell clicks in spreadsheets to and from `Sally`. It allows to store a **semantic illustration mapping**, i.e., a mapping between semantic objects in $\mathcal{A}$ and concepts in $\mathcal{O}$, with the document itself. Note that any open API of an application $\mathcal{A}$ provides us with an opportunity for invasion as it allows decentralized business developments. But the 'thickness' of any `Alex` built on this correlates directly with the dependency of the `SAlly` instance to $\mathcal{A}$. This dependency may have unforeseen consequences; in the `SACHS` project, for instance, we were confronted with an automatic Windows security update that made the used sockets inoperative, so that this dependency resulted in a stableness issue for our semantic extension.

`Theo` To enable invasive design, a screen-area manager `Theo`[3] is needed. $\mathcal{S}$-supplied content is embedded as "popup" into the GUI of $\mathcal{A}$. This embedding can be implemented at two levels: *natively*, if $\mathcal{A}$'s GUI allows to embed browser layout engines or *at the operating system level* by superimposing browser windows over the GUI of $\mathcal{A}$. Given enough care the latter solution can be made very similar to a native integration from a user point of view.

Note that nowadays software applications come in three flavors: traditional desktop applications, web/browser-based applications, and mobile apps. `SAlly` can cope with all of these, if we are flexible in the deployment of the `SAlly` components:

1. For a *desktop application*, the components of `SAlly` run as local processes and use a runtime environment for a browser layout and interaction engine to provide operating-system level GUI embedding (see Figure 1).
2. For *mobile apps*, the application comes in two parts, the document and the core data structures reside on an application server $\mathcal{A}_{core}$, whereas (some of) the user interface functions ($\mathcal{A}_{UI}$) run on the mobile device in the respective app execution environment. In this situation (see Figure 2), we usually cannot assume that a local process can be started, so the `SAlly` components must run as web services on a `SAlly` server. Here, `Alex` is realized as a web service that (a) distributes the user interaction events to $\mathcal{A}_{core}$ and `Sally`,

[2] Note that the `Alex` is the only real "invasive" part of `SAlly`, we have named this system after Alexander the Great; one of the mightiest invaders in history.

[3] 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.

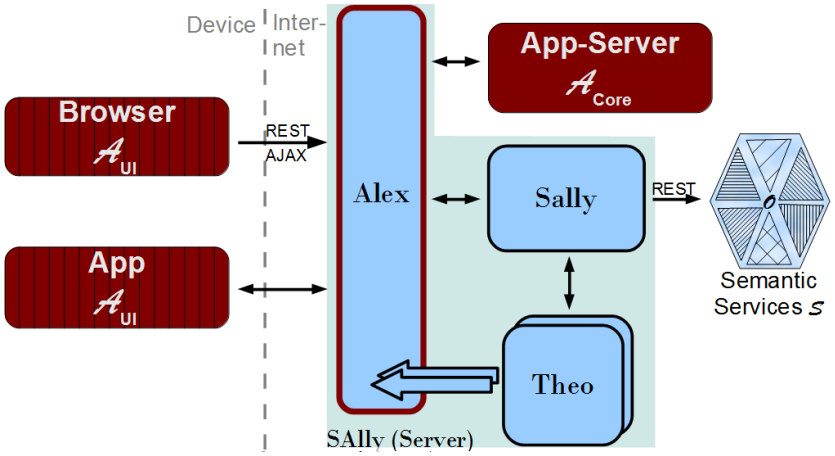

Fig. 2. `SAlly` for Mobile Apps

and (b) aggregates the contributions of $\mathcal{A}_{core}$ (HTML pages) with those from the semantic services (via `Sally` and `Theo`) into a consistent aggregated user experience.

3. For *browser-based applications*, one of the scenarios above applies: If we cannot start processes (e.g. in locked down web-kiosk-style situations) we use the "mobile apps" framework, otherwise a desktop-like deployment may have advantages. In this case, `Alex` is realized as JavaScript browser extension that embeds wMKM content directly into the web document used as GUI by the application server.

Note that in all these cases, the `SAlly` components perform the same functions and communicate with each other in essentially the same way, even if two incarnations of `SAlly` differ in the way they deliver their GUI contributions.

Observe that we need a two-way communication between `Alex` and `Sally` to realize even simple semantic interactions, whereas `Sally` and the wMKM system $\mathcal{S}$ can restrict their communication to a client-server communication, where `Sally` takes the role of a client. For the browser-based case, communication between the browser and `Alex` as well as between `Alex` and the app server is unidirectional REST (Representational State Transfer). Observe also that for the "mobile apps" case, the setup in Figure 2 is only possible, if the communication between the respective app server and app execution environment relies on open APIs or open Standards so that `Alex` can be positioned as a 'man in the middle'. This requirement is the analogue of the "open API" requirement postulated for the desktop application case.

3 A Validation of of the `SAlly` Framework

In order to validate the `SAlly` architecture, we have implemented it for spreadsheet systems within the `SiSsI` project. Concretely, we have realized `Alex`es for

LibreOffice Calc 3.4 [Lib] and MS Excel 2010 to show the feasibility of the framework, but also to get first measures on the efficiency of the SAlly setup. In order to evaluate the new aspects of the SAlly framework, our goal was to reach feature parity with and compare the actual expenditures of doing so for two essential semantic services offered by the SACHS system for MS Excel 2003 spreadsheets:

Definition Lookup & Semantic Navigation in SACHS Here, the event of a cell click triggers the semantic interaction that the user had previously chosen. If she opted for a **"Definition Lookup"**, then SACHS displays the definition from the background ontology (associated with the selected cell via the semantic illustration mapping) as a native MS Excel popup close to the chosen cell. If the found definition involves 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 resp. cell on click of the according graph's node.

3.1 Sissi: An Implementation of SAlly

In the following we describe the realized components of SAlly in the SiSsI project, which show the feasibility of SAlly.

Alex$_{\mathcal{A}}$: *Managing Interface Events*
Our Alex instance for LibreOffice "Alex$_{\text{Calc}}$" uses LibreOffice's Java open API for communication tasks as well as the OpenBasic backend for exporting and managing requested UI events. In contrast, the MS Excel 2010 Alex instance "Alex$_{\text{Excel}}$" is solely based on .NET infrastructure (here C#).

Due to SAlly's invasive design requirement the employment of semantic services is done from within the respective application, therefore Alex$_{\text{Calc}}$ and Alex$_{\text{Excel}}$ are responsible for starting and stopping Sally. Note that from the user's perspective a semantic ally is started, so the menu entry is called "Sally". To support multiple document types (like spreadsheets, presentations, or texts), Alex identifies itself and its document type to Sally during initialization. It also transmits to Sally the semantic illustration mapping stored with the spreadsheet document. Now, Sally as the semantic ally takes on responsibility for all semantic interactions. For the moment, the Alexes only

- report $\mathcal{A}$ cell click events 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), and
- move the cursor to a cell when requested from Sally.

In particular, the Alexes indeed stay very thin. Depending on future semantic interactions based on other semantic objects, the UI listeners and UI actions have to be extended. But note that the former extensions are mere expansions of the listeners, that already are in place, and that the latter will handle nothing more complicated than the resp. semantic objects.

`Sally`: *Merging Interfaces and Interactions*
`Sally` is the central interaction manager of the `SAlly` framework and can be used universally for all kinds of semantic services. As we have seen in Figures 1 and 2, `Sally` comes in two flavors, one for a desktop setting and one for a web-based application. These two only differ in the communication to their respective `Theo`s and possibly `Alex`es. Concretely, we have only implemented the desktop variant for now. As `Sally` has to be cross-platform, it is implemented in Java making use of a Socket and WebSocket server for communication. `Sally` keeps an abstract model of the documents played in the application, an abstract interpretation mapping based on this, and maintains an abstract model of the UI state. These abstractions are useful so that `Sally` can communicate with different `Alex`es/`Theo`s. To avoid hard-coding the set of services, `Sally` will query the wMKM system for the services available for the respective document types registered by the `Alex`es connected to `Sally`.

`Theo`: *Managing Screen-Area*
The main purpose of the `Theo` component is to provision interface items upon request by `Sally`. 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. For math-heavy applications, it is important that `Theo` allows HTML5 presentation of any interface item (buttons, text boxes, etc.) and text content. Note that `Theo` only concentrates on the rendering of interface elements, acting as a renderer for `Sally`, which sends content, placement and size information via Web Sockets[4]. Note that the communication channel is bi-directional: `Theo` events (e.g. clicking on a node in a dependency graph) are communicated to `Sally`, which then coordinates the appropriate reaction.

wMKM = Planetary: Provisioning Semantic Services
For `Sissi`, we (re-)use much of the Planetary system [Koh+11; Planetary] as the underlying wMKM system. The Planetary system is a Web 3.0 (or a Social Semantic Web) system for semantically annotated technical document collections based on MKM technologies. The background ontology $\mathcal{O}$ for semantic illustration is stored as a collection of documents in a versioned XML database, that indexes them by semantic functional criteria and can then perform server-side semantic services. Results of these queries (usually fragments selected/aggregated from $\mathcal{O}$) are transformed to HTML5 via a user-adaptive and context-based presentation process.

3.2 Discussion

To gain an intuition on the runtime-behavior of the `SAlly` framework let us look at the new realization of `SACHS` functionality described earlier:

[4] The natural communication via sockets is prohibited by XULrunner for security reasons; Web Sockets provide a safer abstraction that we can use in this context.

Definition Lookup & Semantic Navigation in `Sissi` In the `Sissi` implementation (see Figure 3), `Alex` responds to a click of cell [E5] by requesting a **definition lookup** window from `Sally`, which requests an HTML5 document from the wMKM system $\mathcal{S}$, on whose arrival `Theo` overlays the $\mathcal{A}$-GUI at the appropriate location with a browser window containing the requested information.

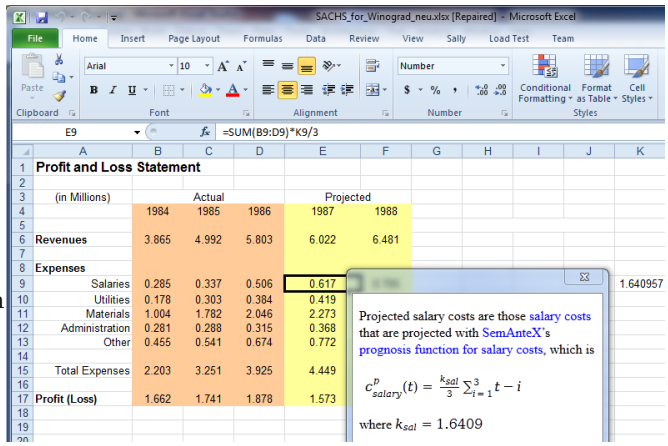

Fig. 3. Definition Lookup in `Sissi`

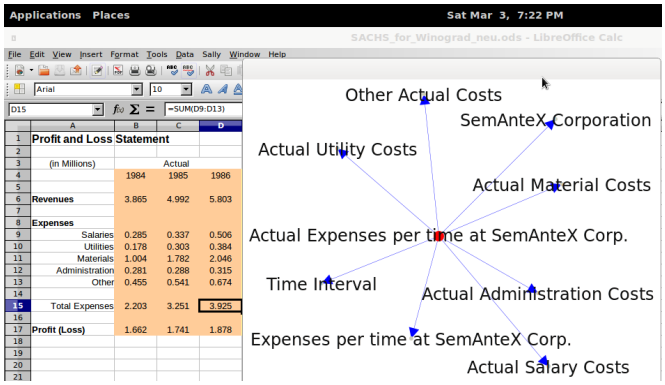

Fig. 4. Semantic Navigation in `Sissi`

Now, let us look at Figure 4, 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 illustration mapping) 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 moved the cursor to cell [D10] in the GUI of $\mathcal{A}$ (based on the evident heuristic). Note that semantic navigation works workbook wide and can be extended to go across. Figure 4 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:

1. The JOBAD interaction framework employed in Planetary allows to make the interactions semantically interactive by embedding semantical services in them as well [GLR09]. For instance, if a fragment of the definition lookup text (e.g. "salary costs" in Figure 3) 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 [B5:D5]), then semantic navigation (which was hard-coded top concept graphs in SACHS) comes for free.
2. 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 SAlly framework, we have the full power of dynamic HTML5[5] at our hands. In particular, OpenMath formulae can be nicely rendered (see Figure 3), which gives spreadsheet users visual (and navigational) support in understanding the computation (and the provenance) of the respective values.
3. 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 illustration mapping.

Note that in the SAlly 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 have access to a suitable (thin) $\texttt{Alex}_{\mathcal{A}}$.

To conclude our discussion let us see how our initial claim that the SAlly framework allows the rapid deployment of semantic services in applications holds up to the implementation experiences. Naturally, the development time of $\texttt{Alex}_{\text{Calc}}$ cannot be considered separately as it is indistinguishable from the expenditures for developing SAlly and Sissi up to this point. But we can look at $\texttt{Alex}_{\text{Excel}}$ and its development costs: Even though the author was not familiar with the MS Excel .NET backend, he succeeded building $\texttt{Alex}_{\text{Excel}}$ 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. For instance, MS Excel removes the cell focus markers, when the spreadsheet window loses focus (which it necessarily does in the SAlly framework, since the Theo supplied window gets the focus, as Theo is a separate process).

4 Related Work

We have presented the SAlly 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 SAlly's contribution by assessing related work. In particular, we review existing relevant (semantic) extensions of document players resp. documents and the frameworks used with respect to our architecture.

[5] and thus in particular HTML+CSS for text layout, MathML for formula display, SVG/Canvas for diagrams, and JavaScript for interactivity

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. But Greasemonkey as well as those other mashup enablers are limited to offer resulting web services. With the `SAlly` 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.

We can also consider `SAlly` as a *mashup builder* like "Marmite" [WH07]. This specific one aligns programs and their data flows and is realized as a proxy server, that mediates between a web browser and a webpage. Again, there seems to be an underlying assumption that mashups only live on the web, the very thing that `SAlly` extends.

When focusing on enabling semantic services as extensions of existing programs, we observe that the number of *semantic service aggregators* is on the rise. The web search engine "WATSON" [dM11], a 'gateway' for the Semantic Web, for a web example, collects, analyzes and gives access to ontologies and semantic data on the Web. But the potential semantic services can only be used via a unified UI, so that such service aggregators have to be considered standalone systems, which we argued against in the introduction. Therefore, our `SAlly` framework only contains the aggregating component `Sally`, but separates its tasks via `Alex` and `Theo`.

There are desktop variants of semantic service aggregators like the "Semantic Desktop" [Sem], which is an open-source semantic personal information management (PIM) and collaboration software system, that offers semantic services based on the relations of structured data of desktop applications. In particular, it uses desktop crawlers based on distinct document type- and application ontologies to collect metadata. Within the Semantic Desktop framework third party components can be integrated via pluggable adaptors. The services can be used by a user via a PIM-UI provided by the Semantic Desktop or via application-UIs that are extended by invasive technology. The cost and redundancy issues discussed in the introduction apply here as well, that is, the development costs for each semantic extension is very high. In the `SAlly` architecture only thin, invasive add-ons ("`Alex`"es) are necessary to provide the same user experience. Moreover, the Semantic Desktop focuses on services using automatically gathered semantic data based on ontologies of desktop applications. In contrast, `SAlly` centers around specific documents with their individual ontology given by a semantic illustration mapping.

"truenumbers" [Tru] is a technology for supporting the representation, management and copy/pasting of engineering values as semantically enhanced data. It encapsulates for instance the magnitude and precision of a number, its units, its subject, and its context. The technology is realized as a set of plug-ins for e.g. Eclipse, MS Office, or Adobe PDF. The metadata are stored in hosted or private clouds and semantic services are offered via a Web client. We consider

truenumbers to be related closely to the SAlly. But as truenumbers only targets engineering values, its semantic objects are limited, and hence, its scope.

In the SAlly framework we are elaborating the idea of "Interface Attachments" [Ols+99]. These are small interactive programs that augment the functionality of other applications with a "*minimal set of 'hooks' into those applications*" [ibd., p. 191], where the hooks exploit and manipulate the *surface representation* of an application. We target with a Theo component this manipulation of the surface representation, but with our thin Alex component we only address the mentioned minimality of hooks, as our main goal does not consist in efficient service exploitation, but in the exploitation of the underlying semantics of a document played by an application. Note that our Theo is application-independent in contrast to "Interface Attachments"' manipulation hook. Moreover, with one Sally component we make complex services available to distinct applications.

"Contextual Facets" [MDD09] are a UI technique for finding and navigating to related websites. They are built automatically based on an analysis of webpages' semi-structured data aligned to a user's short term navigation history and filter selection. They are contextual in the sense, that a user's recent and current webpage elements usage determines the context for available services (here, navigation links). In Sally, SAlly's interaction manager, we use users' interaction with semantic objects (that are given by an abstract document type) like clicking a cell as context for semantic services. The time component of user actions is not yet integrated within Sally. Note that we can consider a semantic ally as a contextual facet, so that the SAlly architecture is a transition from the "Contextual Facets" technique from a browser document extension to general document extensions via the underlying abstract document type. We also like to mention that Medynskiy et al. report for their implementation "FacetPatch" that "*participants in an exploratory user evaluation of FacetPatch were enthusiastic about contextual facets and often preferred them to an existing, familiar faceted navigation interface.*" [MDD09, p. 2013], which makes us hopeful for the adaptation of SAlly.

5 Conclusion and Future Work

We have presented the SAlly 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 her point of view, she 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 resp. application's GUI. This is possible due to an innovative task distribution in SAlly 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 for $\texttt{Alex}_{\text{Excel}}$, 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 and incorporates thus substantial MKM technology.

Our particular Sally implementation in Sissi as of now is desktop-based and realized in Java, integrating much of the semantic functionality via web-services, and our Theo is browser-based. Therefore, our setup of SAlly is fully operating-system independent. So far we have only used Sally to mediate between different types of spreadsheet programs, 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 LibreOffice 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+09]. We expect the main work to be 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.

Even though our work only indirectly contributes to the management of mathematical knowledge, we feel that it is a very attractive avenue for outreach of MKM technologies. For instance, in the SiSsI project for which we have developed the SAlly 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 can be embedded into MS Word, ..., the opportunities are endless.

The SAlly framework is licensed unter the GPL and is available at https://svn.kwarc.info/repos/sissi/trunk/.

Acknowledgements The research in the SiSsI project is supported by DFG grant KO 2428/10-1.

References

[Act] ActiveMath. URL: http://www.activemath.org (visited on 06/05/2010).

[Aßm03] Uwe Aßmann. *Invasive software composition.* Springer, 2003, pp. I–XII, 1–334. ISBN: 978-3-540-44385-8.

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

[dM11] Mathieu dAquin and Enrico Motta. "Watson, more than a Semantic Web search engine". In: *Semantic Web* 2.1 (2011), pp. 55–63.

[GLR09] 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.

[HHN85] 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.

[JK10] Constantin Jucovschi and Michael Kohlhase. "sTeXIDE: An Integrated Development Environment for sTeX Collections". In: *Intelligent Computer Mathematics*. Ed. by Serge Autexier et al. LNAI 6167. Springer Verlag, 2010. ISBN: 3642141277. URL: http://kwarc.info/kohlhase/papers/mkm10-stexide.pdf.

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

[KK09] 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.

[KK11] 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.

[KKL10] Andrea Kohlhase, Michael Kohlhase, and Christoph Lange. "sTeX – A System for Flexible Formalization of Linked Data". In: *Proceedings of the 6th International Conference on Semantic Systems (I-Semantics) and the 5th International Conference on Pragmatic Web*. Ed. by Adrian Paschke et al. ACM, 2010. ISBN: 978-1-4503-0014-8. URL: http://kwarc.info/kohlhase/papers/isem10.pdf.

[Koh+09] 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+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 Com-*

putational 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.

[Koh05] 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.

[Lib] *Home of the LibreOffice Productivity Suite*. URL: http://www.libreoffice.org (visited on 11/13/2011).

[Mata] *Mathcad: Optimize your design and engineering*. URL: http://www.ptc.com/products/mathcad (visited on 02/24/2012).

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

[MDD09] Yevgeniy Medynskiy, Mira Dontcheva, and Steven M. Drucker. "Exploring websites through contextual facets". In: *Proceedings of the 27th international conference on Human factors in computing systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 2013–2022. ISBN: 978-1-60558-246-7.

[NYN03] Jason Nieh, S. Jae Yang, and Naomi Novik. "Measuring thin-client performance using slow-motion benchmarking". In: *ACM Trans. Comput. Syst.* 21 (1 2003), pp. 87–115. ISSN: 0734-2071.

[Ols+99] Dan R. Olsen Jr. et al. "Implementing interface attachments based on surface representations". In: *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*. CHI '99. Pittsburgh, Pennsylvania, United States: ACM, 1999, pp. 191–198. ISBN: 0-201-48559-1.

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

[Planetary] *Planetary Developer Forum*. URL: http://trac.mathweb.org/planetary/ (visited on 09/08/2011).

[Sem] *Semantic Desktop*. URL: http://www.semanticdesktop.org/ (visited on 02/24/2012).

[Tru] *truenumbers*. URL: http:\\www.truenum.com (visited on 02/24/2012).

[WH07] Jeffrey Wong and Jason I. Hong. "Making mashups with marmite: towards end-user programming for the web". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. CHI '07. San Jose, California, USA: ACM, 2007, pp. 1435–1444. ISBN: 978-1-59593-593-9.

[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).