Natural End-User Development of Web Mashups

Saeed Aghaee, Cesare Pautasso
Faculty of Informatics, University of Lugano (USI), Switzerland
saeed.aghaee@usi.ch, c.pautasso@ieee.org

Antonella De Angeli
DISI, University of Trento, Italy
antonella.deangeli@unitn.it

***Abstract*—End-User Development (EUD) can be exploited on the Web, where users are disposed to create niche "Web Mashup" applications out of the composition of many existing Web APIs to address their long tail of situational needs in different domains of application. In this paper, we describe the design of *NaturalMash*, an EUD system that enables the rapid, on-the-fly development of mashups, thus increasing their added-value through decreasing their development costs. *NaturalMash* provides a high level of expressive power while ensuring its usability by non-professional users. This clearly distinguishes *NaturalMash* from existing EUD system for mashups that are either too limited or highly specialized for inexperienced users. *NaturalMash* is based on an efficient combination of end-user programming techniques including natural language programming, What You See Is What You Get (WYSIWYG), and Programming by Demonstration (PbD). The paper describes how a formative evaluation approach is driving the design of *NaturalMash* and discusses the implications of the findings from our user studies. The results obtained with both users and experts are promising and suggest that the proposed system has a short and gentle learning curve so that even non-professional users can use it to rapidly build useful mashups.**

***Keywords*—Mashups, End-User Development, Mashup Tools, Natural Language Programming.**

I. Introduction

With the proliferation of the Web APIs (i.e., reusable software components published on the Web), the Web [1] has become a highly programmable platform. A lightweight form of Web applications that can be developed on this platform is called mashup. Mashups are usually built in an ad-hoc fashion by composing Web APIs and content [2]. As a result, mashups provide users with the opportunity of rapidly satisfying their situational needs in various domains of application [3], [4], ranging from daily utilities of Web users (i.e., consumer market) to specialized domains, such as e-learning [5], bioinformatics [6], health care [7], as well as enterprise mashups [3]. To fully exploit this opportunity, End-User Development (EUD) must be enabled to empower non-professional users to create and modify mashups [8].

In this paper, we present an innovative EUD system for mashup development (or a mashup tool [9]) called *NaturalMash*. *NaturalMash* provides adequate expressive power to create non-trivial, feature-rich, and interactive mashups out of the composition of Web APIs provided through different technologies (ranging from REST and SOAP services to JavaScript and HTML5 widgets). *NaturalMash* is designed to be usable by non-professional users by ensuring it is easy to understand and easy to learn with a gently-sloped learning curve (thanks to a highly interactive, live programming environment, featuring immediate feedback and autocompletion). Many mashup tools with the same level of expressive power (e.g., IBM Mashup Center (http://www.ibm.com/software/info/mashup-center), and JackBe Presto (http://www.jackbe.com/) are, however, designed in a way that is too specialized for non-professional users. On the other hand, mashup tools explicitly targeting non-technical users, such as `IFTTT` (https://ifttt.com) and ServFace [10], do not provide adequate expressive power to freely compose any type of Web APIs.

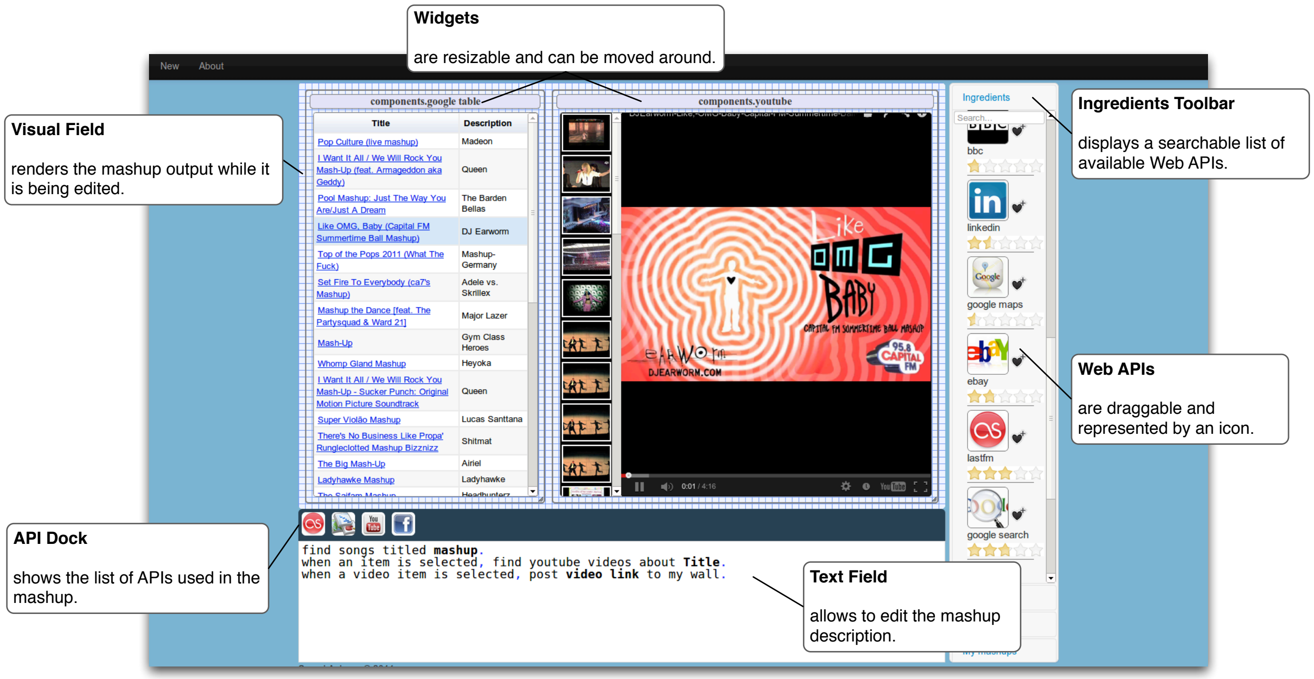
This paper also contributes a novel, hybrid end-user programming technique [11] based on natural language programming [12], WYSIWYG [13] (What You See Is What You Get), and Programming by Demonstration [14] (PbD). *NaturalMash* is one of the first live mashup tools [15] that combines natural language processing techniques [16] with model-driven engineering [17] in order to provide immediate feedback to the users and show them the resulting mashup as they are typing up its description.

A formative evaluation approach enabled us to collect early feedback on the system by two groups of users differing in their computer science knowledge: programmers and non programmers. Programmers were PhD students in Computer Science with excellent coding skills. Non-programmers were users without any computing skills. The evaluation was conducted to better focus the design and avoid gaps between user expectations and the delivered system. Initial findings indicate that users with little or no programming experience can become productive and successfully build mashups, confirming the validity of some of the design decisions behind *NaturalMash*.

The rest of the paper is organized as follows. Section II presents the rationale and goals behind the design of *NaturalMash*. Natural language programming in *NaturalMash* is explained in Section III. Section IV describes the graphical user interface environment of the system. The architecture of the system is discussed in Section V. Section VI reports on the formative evaluation of the system and discusses the impact of users' feedback in terms of usability assessment and suggested areas to improve. Related work is discussed in VII, and conclusions are drawn in Section VIII.

II. Design Rationale

NaturalMash (Figure 1) was designed to keep the mashup composition environment as simple and easy-to-use as possible. The user interface is a Single Page Application (SPA) composed of four main components: (i) *text field* providing advanced support for typing in the description of a mashup integration logic, (ii) *visual field* implementing the WYSIWYG interface for both the design and preview of the mashup user interface being created, (iii) *API dock* graphically representing


Widgets
are resizable and can be moved around.
Visual Field
renders the mashup output while it is being edited.
Ingredients Toolbar
displays a searchable list of available Web APIs.
Web APIs
are draggable and represented by an icon.
API Dock
shows the list of APIs used in the mashup.
Text Field
allows to edit the mashup description.
New
About
components.google table
components.youtube
Ingredients
bbc
linkedin
google maps
ebay
lastfm
google search

Title	Description
Pop Culture (live mashup)	Madeon
I Want It All / We Will Rock You Mash-Up (feat. Armageddon aka Geddy)	Queen
Pool Mashup: Just The Way You Are/Just A Dream	The Barden Bellas
Like OMG, Baby (Capital FM Summertime Ball Mashup)	DJ Earworm
Top of the Pops 2011 (What The Fuck)	Mashup-Germany
Set Fire To Everybody (ca7's Mashup)	Adele vs. Skrillex
Mashup the Dance [feat. The Partysquad & Ward 21]	Major Lazer
Mash-Up	Gym Class Heroes
Whomp Gland Mashup	Heyoka
I Want It All / We Will Rock You Mash-Up - Sucker Punch: Original Motion Picture Soundtrack	Queen
Super Violão Mashup	Lucas Santtana
There's No Business Like Propa' Rungleclotted Mashup Bizznizz	Shitmat
The Big Mash-Up	Airiel
Ladyhawke Mashup	Ladyhawke
The Saifam Mashup	Headhunterz

```
find songs titled mashup.
when an item is selected, find youtube videos about Title.
when a video item is selected, post video link to my wall.
```

Fig. 1. *NaturalMash* environment: users type the description of the mashup in the text field and immediately see the output in the visual field. The output contains interactive widgets that can be resized and relocated. The ingredients toolbar helps with API discovery, while the dock gives a summary of the APIs used in the current mashup. Web APIs are abstracted away from the technologies they use and are represented as icon.

the APIs used by the mashup, and (iv) *ingredients toolbar* containing all the mashups created by the users and a searchable list of Web APIs.

The four components of the environment are meant to be used together as follows. The ingredients toolbar gives a searchable overview of the available APIs that are available to be mashed up. Users can drag-and-drop APIs from the ingredients toolbar into the visual field to build their desired mashups. Alternatively, they can use the text field to describe the mashup using natural language. The text field is equipped with advanced features like *autocomplete* suggesting matching API descriptions as the user types text fragments. The visual field also enables the use of PbD: once users start interacting with the widgets, some suggestions on how to describe their interaction are proposed in the text field. The interactive API dock allows users to highlight or remove APIs.

NaturalMash is a WYSIWYG environment based on the live programming paradigm [18], [19], in which the normal edit/compile/run development lifecycle is fully automated by the system. As a result, users can more easily bridge the gulf of evaluation (the degree of difficulty of assessing and understanding the state of the system [20]). This in turn leads towards an improved learning experience [21].

NaturalMash combines three techniques of end-user programming as follows. Natural language programing is enabled through a *Controlled Natural Language* (CNL) — a subset of a natural language (e.g., English) restricted in terms of vocabulary and grammar. The visual field provides the visible and live output of the mashup being created and facilitates natural language programming through visual demonstration and interactions with widgets (e.g., clicking a map widget adds the corresponding natural language description to the text field, being, for instance, "when the map is clicked"). From the expressive power point of view, the *NaturalMash* CNL empowers users to describe relatively complex process orchestration and data integration logic as well as the composition of widgets (all at a very abstract level), whereas the visual field provides a direct way to manipulate the user interface (WYSIWYG), and partially the application logic (through PbD), of the mashup being created. As a result, the user interface becomes much more intuitive because it supports both direct manipulation (visual field) and descriptive representation (text field) of the mashup being created.

Overall, we expect our design to empower non-professional users (e.g., non-programmers) to create useful mashups with minimal prior knowledge.

III. NaturalMash Controlled Natural Language

Natural language programming in *NaturalMash* is enabled by a CNL that is an abstract, executable language for modeling the presentation integration, process integration, and data integration layers of mashups. For example, Listing 1 conforms to the *NaturalMash* CNL and describes an enhanced music video search mashup that employs Last.fm (http://www.last.fm/api) to first search for a song and then uses the results to accurately search for the corresponding music videos of the song in YouTube (https://developers.google.com/youtube/).

```
Find songs titled mashup. When an item is selected,
search YouTube videos about title.
```

Listing 1. An enhanced music video search mashup. "mashup" originally refers to a type of song created by mixing two or more songs

The underlying implementation of the CNL accommodates an abstract component model that: (i) gives a unified technology-neutral description of Web APIs, and (ii) models them in an abstract textual form (natural language description). The abstract component model distinguishes two types of functionality provided by Web APIs, namely, *Task* – a passive atomic operation, and *Event* – an active source of control. More in detail, an Event describes a condition which if satisfied, produces a message that may be read from the Event's output parameters. In the above example, selecting an item in the songs table is an Event of the table widget. A Task, on the other hand, takes some input data, does some processing, and then produces output data. Finding songs, given a keyword, exemplifies a Task behavior of the Last.fm API. The data consumed and produced by Tasks and Events of Web APIs is modeled as, respectively, one or more input and output *parameters*. Each parameter has a meaningful, unique (only within the API scope) name; its syntax and semantic descriptions are defined but not shown to the end-users.

To give a natural language representation of APIs, each of its Tasks and Events is associated with a specific natural language description. For instance, "`find songs titled [keyword]`" describes the song-searching Task of the Last.fm API. The input parameter name `keyword` is enclosed within square brackets, creating a placeholder for the object of the verb used in the description. The object may be a parameter name referring to the output of previous tasks, a constant value, or an anaphora — in linguistic, an anaphora (e.g., "`that`") is defined as an expression linking two elements in a document — pointing to a specific part of the mashup description text. "`an item is selected`" exemplifies the description of an Event. Note that an Event does not receive input parameters, and thus its description does not contain any placeholder.

To describe how to compose together APIs, the CNL imposes specific grammatical constraints, which limit the types of sentences that can be constructed. We distinguish:

- **Imperative sentences:** They are composed of multiple imperative mood clauses. Each clause is built from the description of a Task by replacing its placeholders with appropriate objects. For example, given the description "`find songs titled [keyword]`" the corresponding clause can be "`find songs titled mashup`", where the object is replaced with a constant value ("`mashup`").

- **Causal sentences:** They are written in causal form in which the time conjunction "`when`" introduces a passive clause (the description of an Event) followed by a set of imperative mood clauses (like an imperative sentence). The passive clause describes the cause of the event; the imperative mood clauses represent the effect to be realized when the cause of the event happens. Consider the causal sentence in the example (Listing 1) "`When an item is selected, search youtube videos about title.`". In this sentence, when the Event described as "`an item is selected`" happens, the Task ("`search youtube videos about title`") is executed.

IV. NaturalMash Composition Environment

The *NaturalMash* environment is designed to provide an innovative selection of features that are meant to enhance the user experience and the ability of users to build sophisticated mashups. We first describe each feature individually and later show in a usage scenario how they are used in conjunction to build a mashup.

- **Inline Search:** To enhance API discovery, the *NaturalMash* environment provides an *inline search* feature (Figure 2) in the text field that allows users to either (i) directly type in the text editor the (approximate) name of the API they are looking for, which results in the user getting a list of descriptions associated with the API matching or approximating the given keyword, or (ii) type what the API is supposed to do (in case they do not know or cannot guess the name of the API), by doing which the input text will be matched against the natural language descriptions of all the APIs in the library. In the latter case, the mechanism of searching descriptions is based on (i) exact match, (ii) word synonym (e.g., "`search`" and "`find`"), or (iii) word semantics (e.g., "`location`" and "`map`").

- **Semi-structured Text Editor:** The *NaturalMash* CNL is based on a restricted grammar meaning that not all possible input combinations are acceptable. Consequently, the CNL acts as a learning barrier as the users need to master the grammar and syntax of the language. To support the users' learning experience, the text field provides a semi-structured text editor ensuring that user input will not cause syntax errors, while still allowing a high degree of freestyle editing. To be specific, the text field: (i) restricts input characters to avoid accidental syntax errors (for instance the new line characters are disabled while typing an object in a placeholder), (ii) automatically inserts the separators ("`,`", "`and`", and "`and,`") if the cursor is positioned before and after clauses (manual insertion of the separators is also possible), and (iii) streamlines selecting and moving text objects (clauses) via, respectively, double-click and drag-and-drop.

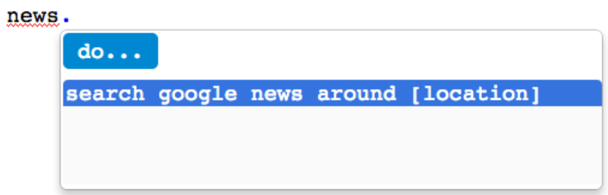


Fig. 2. Typing "news" (inline search) results in the autocomplete list showing the Google News API description as a suggestion.

- **Autocompletion:** The autocomplete feature (Figure 2) is also utilized to lower the CNL learning barrier. Based on what users type in the text field, a list automatically appears and shows suggestions for Task/Event descriptions (to support API discovery and reuse), and data flow (i.e., referencing suitable objects within Task descriptions).

- **Data Flow Highlighting:** In the text field, objects indicating flow of data are displayed in boldface (Figure 3). Moving the cursor on the text representing an object results in the highlighting of the text describing its source Task or Event. This way, users can discover the source of an object both when browsing the data flow suggestions in the autocomplete list but also after a data flow suggestion has been entered in the text.

• **Error Highlighting:** If there is an ambiguity in the mashup description (e.g., the input text does not match any Task or Event description), the compiler produces an error that is reported to the user as the text is being entered (Figure 2). Similar to many "spell-checking text editors", the error is shown by highlighting (in red color) the text that produced it. An autocomplete list containing possible suggestions to disambiguate the description is shown whenever the user moves the cursor (or click) on the highlighted erroneous text, and thus offering the opportunity to the user to quickly correct the mistake.

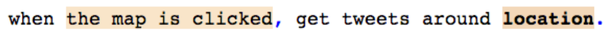

when the map is clicked, get tweets around location.

Fig. 3. Data flow highlighting feature visualizes the source of an object.

• **Drag-and-Drop:** The ingredients toolbar gives a visual overview of the available APIs. From there, users can drag-and-drop an API into the text field, visual field, or API dock. If the API is a widget, it will be displayed in the visual field. Also, the autocomplete list will appear containing the corresponding Task/Event descriptions.

• **Programming by Demonstration:** Interacting with widgets in the visual field results in appending the corresponding Event description to the text field. For instance, clicking on Google Maps widgets results in showing the text "`when the map is clicked`" in the text field. To grab the attention of users the text corresponding to the event is highlighted.

• **Synchronized Multi-perspective Modeling:** The three main interaction components of the environment (API dock, text field, and visual field) are all kept synchronized during every user interaction: (i) editing text in the field updates the visual field and the API dock; (ii) selecting a widget from the visual field or a API from the API dock results in highlighting its corresponding text in the text field and vice versa (moving the caret through a portion of the text highlights its associated widgets and API icons); (iii) deleting an icon from the dock or a widget from the visual field results in the removal of its corresponding text.

A. *Usage Scenario*

The following illustrates a common and complete usage scenario of *NaturalMash*, whereby a user builds the mashup example described in Listing 1.

The first step is to discover the right APIs for finding songs. This step is facilitated by the inline search feature which enables the users to type what the API he is looking for is supposed to do. For instance, the user can start by typing "`search musics`", which results in the text field providing an autocomplete list of descriptions that contain the input words or synonyms for the words. Once the autocomplete list is displayed, the user can select a proper suggestion (in this case, "`find songs titled [keyword]`") by either pressing the Enter key or by pointing with the mouse and clicking.

After selecting a suggestion from the autocomplete list, (i) the completed natural language description of the API is inserted in the text field, (ii) ambiguity is resolved, in case there are one or more similar descriptions, (iii) the mashup is rebuilt and executed, (iv) another autocomplete list containing dataflow suggestions for the description is displayed. For the input parameter ("`[keyword]`"), the user may type a constant string like "`[mashup]`" resulting in a mashup that uses Last.fm API to search for songs matching the input constant, and automatically shows the results in a table widget.

The output mashup is interactive and support PbD in a way that, for instance, clicking an item in the table widget results in appending the corresponding Event description (i.e., "`[when an item is selected]`") to the text field as well as setting the focus in a way that makes it easier for the user to add some Task descriptions to complete the causal sentence. For example, the user may type "`[video]`" in the text field or, alternatively, search YouTube API in the ingredients toolbar and then drag-and-drop the API icon to the text field, both of which result in displaying an autocomplete list containing the YouTube API description. Immediately after selecting the suggestion, another autocomplete list containing data flow suggestions is shown to the user. The user can select the output parameter "`title`" from this list, or type an anaphora pointing to the item such as "`it`", both referencing the click event description of the table widget. The data flow highlighting feature helps users to figure out the source of a suggestion. Leaving the object placeholder empty results in the compiler error that is shown by a clickable wavy red line. Clicking on the wavy red lines causes the autocomplete list associated with the placeholder to appear again.

While typing the mashup description, the user may modify the mashup user interface layout in the visual field. The final mashup can then be deployed in production, with a single click. Even after a mashup has been published, it still remains modifiable and can be redeployed at any time.

It should be noted that the user may follow a different (or a wrong) path to complete the above example. Therefore, the two-dimensionality of text (vs. multi-dimensionality of visual editors) might be a hinder. To address this problem, the semi-structure editor facilitates relocating text objects in a drag-and-drop fashion.

V. Architecture

NaturalMash is designed as a live mashup tool, which completely automates the repetitive task of compiling mashup descriptions and running them. Internally, mashup descriptions go through a compilation pipeline that transforms them in executable models of Web service compositions that are executed by the JOpera engine [22]. In the following we briefly describe the main steps of the *NaturalMash* compilation process.

The process revolves around a mashup representation that initially contains the input mashup description text, but later is augmented with a list of components used by the mashup, the specifications of the layout of the mashup user interface (e.g., the position of each widget), data flow information (i.e., source and destination of objects), and, if necessary, disambiguation information (e.g., when there are multiple APIs with the same Task/Event description). The mashup representation is thus recycled with each round of compilation and is regularly updated as the mashup is being developed.

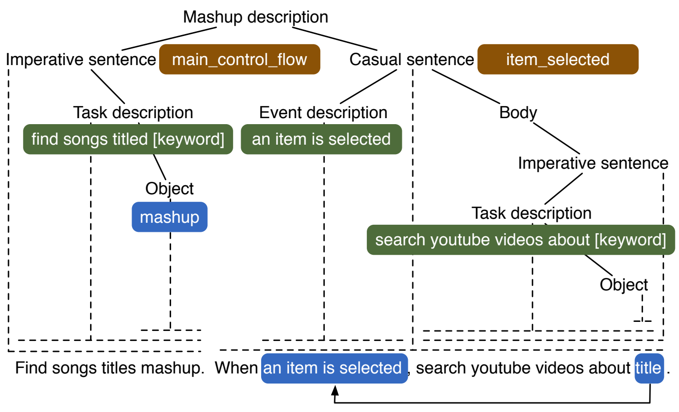


Fig. 4. The annotated syntax tree corresponding to the mashup description of Listing 1.

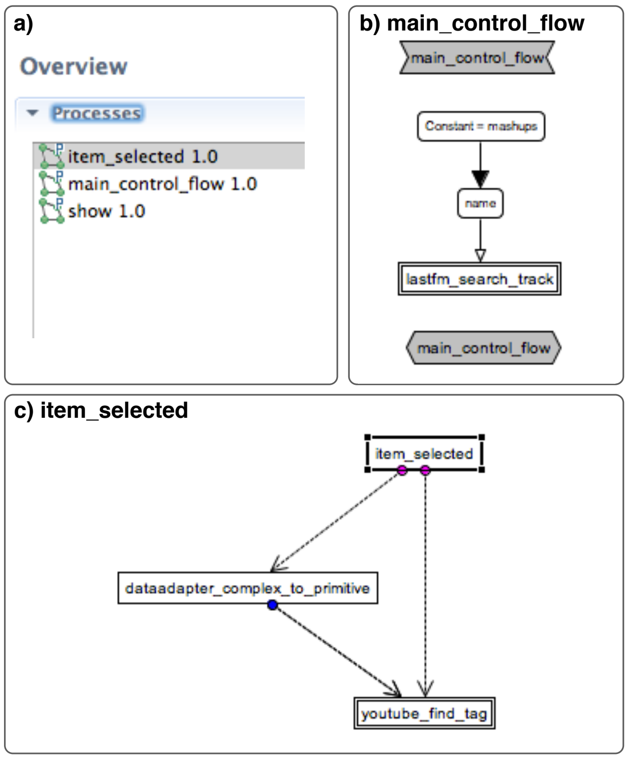


Fig. 5. The generated JOpera visual composition code [22] for Listing 1: (a) list of processes created for the mashup: `main_control_flow` implements the control flow for the imperative sentence, `item_selected` implements the control flow associated with the casual sentence (item select event), and `show` is the process responsible for creating the user interface of the mashup, (b) dataFlow corresponding to the Task description "`find songs titled mashups`" (mashups is a constant passed to the `lastfm_search_track` task), and (c) `item_selected` control flow is triggered whenever an item is selected in the table.

- **Step 1: Natural Language Parsing** The input mashup description text is parsed and its linguistic information (e.g., parsing tree, anaphora resolution, etc.) is extracted. This step is implemented using the Stanford CoreNLP library (http://nlp.stanford.edu/software/corenlp.shtml), which enables to tokenize the input text and split it into sentences, parse the text and assign a part-of-speech tag (verb, noun, etc.) to each word, process grammatical dependencies, and create the anaphora resolution graph.

- **Step 2: Constrained Natural Language Parsing** A syntax tree based on the *NaturalMash* CNL grammar is produced. In this step, we use a formal lexer and parser (implemented using ANTLR, http://www.antlr.org/) to extract and identify sentence types as well as to extract their chunks (i.e., imperative or passive clauses).

- **Step 3: API Binding** The output of Steps 1 and 2 is consumed to build a mapping between the text chunks (i.e., clauses and phrases) extracted in Step 2 and their corresponding Event/Task description. To attain this mapping, we first gather all the descriptions associated with the Tasks and Events of the APIs registered within the *NaturalMash* library. These are matched against the text chunks by ignoring the parameter placeholders. The result is a mapping between each text chunk and the corresponding Task or Event. In this step, ambiguity may occur when more than one Task/Event description match the same text chunk. Assuming that multiple APIs sharing the same description are equivalent, the ambiguity can be resolved automatically based on well known QoS-driven dynamic binding techniques [23]. Manual intervention through the tool's autocompletion feature is required only if there is an aliasing problem.

- **Step 4: Data Flow Resolution** The mapping generated from Step 3 is used to extract objects references and complete the syntax tree (Figure 4). To do so, the placeholders found within the Task descriptions representing input data are bound to the output data referenced from the actual text. Using the results of the linguistic analysis (Step 1) also the anaphoric objects are resolved. The data flow operations (e.g., matching and conversion) are delegated to the *NaturalMash* semantic framework for data integration. The framework defines a schema for input and output parameters of APIs. The schema contains metadata such as data type (primitive or complex), MIME type (e.g., application/xml and application/json), and ontology-based semantic annotation. Explaining the semantic data integration framework in detail is out of the scope of this paper. In case of ambiguity, we use the autocomplete feature to let the user specify the correct data flow references.

- **Step 5: Intermediate Model** The disambiguated syntax tree is consumed to generate an intermediate model that includes control flow and data flow graphs representing the algorithmic structure of the mashup. The nodes of these graphs also store a mapping between the executable and data elements of the CNL (technology-neutral) and the target executable model of JOpera (technology-specific). The relationship is established in advance and does not need to be exposed to the end-users, who are presented with natural language descriptions and icons of the APIs.

- **Step 6: Emitter** The intermediate model is transformed into the target composition code (Figure 5), which is directly executable by JOpera mashup engine, which further transforms it internally to Java bytecode for efficient execution.The mashup execution is controlled by *NaturalMash* through a REST API, which also allows to retrieve and display its results. By replacing the emitter it is possible to target other mashup runtime platforms.

VI. Formative Evaluation

NaturalMash evolved following a one year user-centered design approach, iterating design and evaluation activities. The evaluation provided rich users feedback at different stages of development and ensured that users were kept central in the design so to avoid as much as possible mismatches between users expectations versus system behavior. So far, we have completed two design-evaluation iteration cycles, in which the results of the evaluation conducted in each iteration informed the next iteration design. In this section, we only present the evaluation results of the last iteration (second iteration).

A. User Study: Second Iteration

At the end of the second iteration, we conducted a formative evaluation on a relatively large group of participants (22 persons). The formative evaluation mainly aimed at identifying any potential usability problems and informing the design process. In the process, due to the appropriate size of the participants, we also attempted to partially assess the goal of the design being to empower non-professional users to effectively and efficiently create useful mashups (Section II).

1) Users: NaturalMash is meant to be used by a wide variety of users on the Web in different domains of applications. In general, we classify the potential users of our system into programmers and non-programmers. We designed a background assessment questionnaire that helped us distinguish the participants accordingly.

Non-Programmers They neither have programming skills nor exposure to programming, but they use Web 2.0 services like social networks, blogs, and Web feeds. They may also have experience in advanced end-user programming tools such as spreadsheets, wikis, and visual Website builders.

Programmers They are either professional programmers (with at least three years of programming experience), or at least are in the process of learning how to program and may be familiar with some markup or scripting languages.

We recruited a total of 22 participants from young university staff and students volunteers both at the University of Lugano and at the University of Trento. In terms of programming skills, they were equally divided into programmers and non-programmers.

2) Method: The participants were given four tasks of growing complexity (in terms of the number of APIs to be mashed up), after receiving a short tutorial (5 minutes) in the form of a warm-up task (with the minimum complexity of two APIs):

Task 1 Search Flickr images with location from Google Maps (two APIs).

Task 2 Show upcoming events in a selected location on the map. Get information about each event from Google (three APIs).

Task 3 Find slides about "Web APIs". For each slide found, show relevant videos, tweets, and images (four APIs).

Task 4 Create a mashup on your own (open task).

During the study we recorded the user sessions (video, audio, and screen) and asked the users to think aloud about their activities. The recordings were complemented

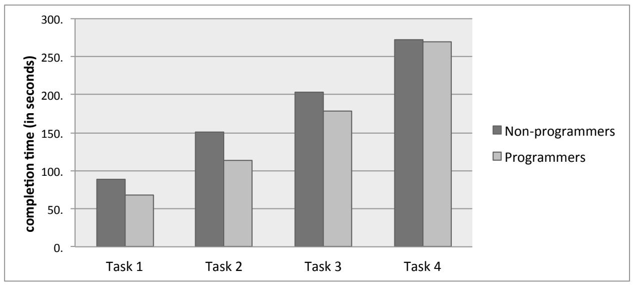


Fig. 6. The completion time grows with the complexity of the task at hand. Programmers have a slightly shorter completion time than non-programmers.

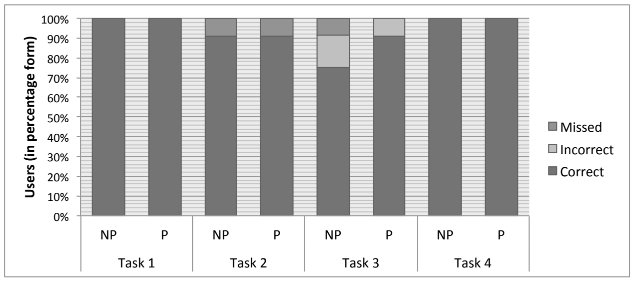


Fig. 7. The majority of tasks were accomplished successfully (correctly). In terms of accuracy, however, there is no major difference between programmers and non-programmers.

by an informal interview as well as an exit questionnaire (containing Likert scale questions) asking them about their overall reaction and satisfaction with the tool. The aim of the extended post-study interviews was to have a deeper, but informal discussion with each participant with the opportunity to reflect on what was not captured by the questionnaires and to further discuss the rationale behind some answers. For instance, we asked "`Do you have any personal mashup/use-case of the tool?`", "`Why do you feel comfortable with the tool?`", and the like.

3) Results: In terms of accuracy and efficiency (Figure 6 and 7), the majority of participants completed all the tasks correctly and in a very short time (around 3 minutes on average), with a slightly better performance on the programmers' side. Out of the total of 88 tasks, 11 programmers produced 86 correct tasks, while the 11 non-programmers achieved 84 correct tasks. Moreover, by the end of each user study session, the majority of participants felt confident about their mastery of the tool and reported on a high level of satisfaction in the exit survey (72% felt satisfy with the tool, 89% were interested in continuing using it, and 87% wanted to suggest it to their friends).

Our recorded observations together with the feedback from participants through both the exit questionnaire and informal discussion, reported positively on the individual features of the *NaturalMash* environment (Section IV). More in detail, the following percentage of participants reported that the features were helpful or very helpful for the completion of the given tasks: autocomplete (92%), inline search (91%), live execution (86%), ingredients toolbar (82%), and PbD (73%). The ingredients toolbar was more frequently used for component discovery and selection than the inline search with the text field (in average, 84% of component discovery and selection tasks were done using the ingredients toolbar). Instead, users employed the inline search feature when they were looking for a specific operation that could be perfectly described verbally.

Overall, the participants were engaged with the tool (77% felt the tools was stimulating or very stimulating). In the open task, all created distinct, useful, and non-trivial mashups. One example was a mashup that finds an audio album in eBay and plays it in YouTube by first searching and finding the exact name of the album using the Last.fm API. Another example was a mashup that shows news, Flickr images, and tweets all related to a selected location on the map, and then allows to share the results on Facebook. Indeed, some of the mashups created in the open task were actually meant to address a real pressing need of the participants. For instance, one of the participants created a mashup to automate the analysis of online presence within the tourism domain. The mashup searches tweets for a specific tourism-related keyword, and then for each tweet found, it searches for the Facebook profile given the name of the author of the tweets.

4) Lessons Learned: The results collected so far appear to support our design rationale and goal. Nonetheless, the main goal of the formative evaluation was to identify usability problems so as to inform the next iteration design and development. We observed that some users – especially non-programmers who lack algorithmic thinking abilities – would benefit from receiving suggestions not only for individual Event/Task descriptions but also for hints on how to compose them together in the right order.

Another major usability problem was with the way PbD is applied in the visual field, i.e., interacting with widges results in the corresponding Event description being added to the text field. However, many participants confused capturing the general behavior of a widgets (i.e., the event) with the recording of the concrete action on the widget they just triggered (e.g., a specific location they have clicked). For example, clicking on the map would add the map-click event description (“`when the map is clicked`”) to the text. This is meant to be completed by appending Task descriptions (e.g., “`show upcoming events around location`”) that form the body of the causal sentence. Indeed, we observed – in the same map example – the users correctly generating the event text using PbD and completing it with an imperative clause, expected to immediately see the results from the location they had originally selected (demonstrated) to create the causal sentence, as opposed to having to click again on the map to obtain the results. In other words, they did not realize they had created a parametric mashup that shows events for any location on the Map. A similar problem occurred with other widgets supporting PbD, such as the table.

VII. Related Work

In recent years, a number of mashup tools have been designed in both academia and industry. Notable industrial mashup tools are Yahoo! Pipes (http://pipes.yahoo.com/), IBM Mashup Center, JackBe Presto, and MyCocktail (http://www.ict-romulus.eu/web/mycocktail). These mashup tools, regardless of their relatively good level of expressive power, do not usually target non-professional users. We believe the full potential of the programmable Web can be exploited only when everyone is empowered to create mashups.

Recently, there have been a number of mashup tools designed specifically for non-professional users. Examples are Omelette [24], DashMash [25], ServFace [10], and RoofTop [26] that enable users to create mashups by providing a set of connectable configurable boxes whose visual layout at design time reflects the one at run time. Nevertheless, the integration logic of a mashup makes use of data filtering, correlation and conversion operations that are usually not visible in the graphical user interface, and therefore are not directly accessible for modification using a pure WYSIWYG tool. Another non-professional mashup tool is `IFTTT` which even though it is based on natural language, restricts the user’s input using a structured visual editor. Also, IFTTT only allows to create mashups based on a specific control-flow pattern (if this then that) using a predefined list of components. Existing tools designed for non-professional users are limited in terms of expressive power, as a result of which the diversity of the useful mashups they can produce is restricted.

In this paper, we showed that *NaturalMash* is not only capable of abstracting and composing a wide range of Web API technologies, but also, at the same time, is usable by non-professional users. This is clearly a major step forward in the design of next-generation mashup tools.

A. Natural Language Programming

As a research topic, natural language programming has a long history. As an example of recent work in this area, the system presented in [27] allows to use natural language to express formal rules defined in the RoboCup coach language. System English (http://www.system-english.com/) enables to refer to MATLAB function calls using regular sentences. CoScripter [28] is a natural language based scripting environment for automating Web browsing activities. The idea of sloppy programming utilized by CoScripter is closely related to the natural language programming style of *NaturalMash*. In the area of personal information management, [29] presented *Automate* that uses a simplified CNL for context-sensitive personal automation. More recently, [30] proposed a controlled natural language interface to simplify the development of semantic mediawikis.

Even if natural language is considered a natural way for humans to command computers ([31], [32], [33]), it cannot be efficiently used to design integrated user interfaces, a fundamental mashup development activity [34]. *NaturalMash* not only enables the development of relatively complex mashups through supporting various programming constructs (i.e., events) in a very abstract manner, but also, at the same time, supports the user interface design by means of direct manipulation through a drag-and-drop WYSIWYG interface.

Moreover, it could be argued that learning the restrictions of a controlled natural language may be more cumbersome than learning the syntax of an artificial language from scratch. To address this problem, we introduced the controlled natural language within an environment that enables end-users to immediately use the language, without prior training, by means of features such as: autocompletion (for API discovery and data flow resolution), error and text highlighting (to visualize data flow and and displays errors), and immediate feedback (to shorten the gulf of evaluation).

VIII. Conclusion and Outlook

In this paper we presented *NaturalMash*, a "natural" tool for end-user mashup development. *NaturalMash* is based on a novel hybrid composition technique combining a controlled natural language tuned for mashup development with an interactive WYSIWYG and drag-and-drop interface allowing the direct manipulation and live execution preview of the resulting mashup user interface. The design of *NaturalMash* has adopted an incremental, user-driven approach in which iterative formative evaluations inform the next steps to be taken to improve the usability of the tool. The results of our formative evaluations helped us to identify several usability problems and gather ideas on how to address these problems. Also, the results provided positive feedback about the tool design, demonstrated its usability by non-professional users as well as its adequate expressive power to let users create useful and non-trivial mashups.

For the future, we plan to (i) enable autocompletion of mashup compositions based on semantic and syntactic matching to assist non-programmers with limited algorithmic thinking capabilities, (ii) boost the use of programming by demonstration to generate output in addition to behavior, and (iii) enhance the live execution feature by saving and restoring the current state of the user interface across compilation cycles.

Acknowledgements

We are grateful for the support and the expertise of Monica Landoni with the preparations and the analysis of the formative evaluation results. This work is partially supported by the Swiss National Science Foundation with the SOSOA project (SINERGIA grant nr. CRSI22 127386).

References

[1] T. O'Reilly, "What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software," *Communications & Strategies*, pp. 17+, 2007.

[2] D. Benslimane, S. Dustdar, and A. Sheth, "Services Mashups: The New Generation of Web Applications," *IEEE Internet Computing*, vol. 12, pp. 13–15, 2008.

[3] A. Jhingran, "Enterprise information mashups: integrating information, simply," in *Proc. of VLDB 2006*, 2006.

[4] C. Anderson, *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[5] M. Eisenstadt, "Does elearning have to be so awful? (time to mashup or shutup)," in *Proc. of ICALT 2007*, 2007.

[6] C. Goble and R. Stevens, "State of the nation in data integration for bioinformatics," *J. of Biomedical Informatics*, vol. 41, pp. 687–693, 2008.

[7] M. N. K. Boulos and S. Wheeler, "The Emerging Web 2.0 Social Software: An Enabling Suite of Sociable Technologies in Health and Health Care Education," *Health Information & Libraries Journal*, vol. 24, pp. 2–23, 2007.

[8] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-User Development: An Emerging Paradigm," in *End User Development*. Springer Netherlands, 2006.

[9] S. Aghaee, M. Nowak, and C. Pautasso, "Reusable decision space for mashup tool design," in *Proc. of EICS 2012*, 2012.

[10] T. Nestler, M. Feldmann, G. Hubsch, A. Preussner, and U. Jugel, "The ServFace builder - a wysiwyg approach for building service-based applications," in *Proc. of ICWE 2010*, 2010.

[11] B. A. Nardi, *A small matter of programming: perspectives on end user computing*. MIT Press, 1993.

[12] L. Miller, "Natural language programming: styles, strategies, and contrasts," *IBM Syst. J.*, vol. 20, pp. 184–215, June 1981.

[13] J. Rode and M. B. Rosson, "Programming at runtime: requirements and paradigms for nonprogrammer web application development," in *HCC*, 2003, pp. 23–30.

[14] A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, Eds., *Watch what I do: programming by demonstration*. MIT Press, 1993.

[15] S. Aghaee and C. Pautasso, "Live Mashup Tool: Challenges and Opportunities," in *Proc. of 1st ICSE Workshop on Live Programming*, 2013.

[16] R. Mihalcea, H. Liu, and H. Lieberman, "NLP (Natural Language Processing) for NLP (Natural Language Programming)," in *Computational Linguistics and Intelligent Text Processing*. Springer, 2006.

[17] S. Casteleyn, F. Daniel, P. Dolog, and M. Matera, *Engineering Web Applications*. Springer, 2009.

[18] N. Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live Coding in Laptop Performance," *Org. Sound*, vol. 8, pp. 321–330, 2003.

[19] S. L. Tanimoto, "VIVA: A visual language for image processing," *J. Vis. Lang. Comput.*, vol. 1, pp. 127–139, 1990.

[20] D. A. Norman and S. W. Draper, *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., 1986.

[21] A. Repenning and A. Ioannidou, *What Makes End-User Development Tick? 13 Design Guidelines*. Springer Verlag, 2006, pp. 51–85.

[22] C. Pautasso and G. Alonso, "The JOpera visual composition language," *Journal of Visual Languages and Computing*, vol. 16, pp. 119 – 152, 2005.

[23] A. Strunk, "Qos-aware service composition: A survey," in *Proc. of ECOWS 2010*, 2010.

[24] O. Chudnovskyy, T. Nestler, M. Gaedke, F. Daniel, J. I. Fernández-Villamor, V. Chepegin, J. A. Fornas, S. Wilson, C. Kögler, and H. Chang, "End-user-oriented Telco Mashups: The OMELETTE Approach," in *Proc. of WWW 2012*, 2012.

[25] C. Cappiello, M. Matera, M. Picozzi, G. Sprega, D. Barbagallo, and C. Francalanci, "DashMash: A Mashup Environment for End User Development," in *Proc. of ICWE 2011*, 2011.

[26] V. Hoyer, F. Gilles, T. Janner, and K. Stanoevska-Slabeva, "SAP Research RoofTop Marketplace: Putting a Face on Service-Oriented Architectures," in *Proc. of SERVICES*, 2009.

[27] R. J. Kate, Y. W. Wong, and R. J. Mooney, "Learning to transform natural to formal languages," in *Proc. of AAAI 2005*, 2005.

[28] G. Leshed, E. M. Haber, T. Matthews, and T. Lau, "Coscripter: automating & sharing how-to knowledge in the enterprise," in *Proc. of CHI 2008*, 2008.

[29] M. Van Kleek, B. Moore, D. R. Karger, P. André, and m. schraefel, "Atomate It! End-user Context-Sensitive Automation using Heterogeneous Information Sources on the Web," in *Proc. of WWW 2010*, 2010.

[30] P. R. Smart, J. Bao, D. Braines, and N. R. Shadbolt, "Development of a controlled natural language interface for semantic mediawiki," in *Proc. of CNL 2009*, 2010.

[31] C. Green, "A summary of the psi program synthesis system," in *Proc. of ICAI 1977*, 1977.

[32] G. E. Heidorn, "Automatic programming through natural language dialogue: a survey," *IBM J. Res. Dev.*, vol. 20, pp. 302–313, July 1976.

[33] E. Kaufmann and A. Bernstein, "How useful are natural language interfaces to the semantic web for casual end-users?" in *Proc. of ISWC'07/ASWC 2007*, 2007.

[34] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, "A framework for rapid integration of presentation components," in *Proc. of WWW 2007*, 2007.