

D4.1.1 – Report on use cases for service design & D4.1.4 – Experiments on solving use cases with selected tools

Research Area: Ecosystem Design and Evolution
Project Title: Ecosystem Design and Evolution (EDEN)
Document Type: I (Internal)

Document Id: FS-EDEN-D411
Document Title: **Deliverable 4.1.1: Report on use cases for service design**
Deliverable 4.1.4: Experiments on solving use cases with selected tools
Editor: Seppo Törmä
Authors: Seppo Törmä
Status / Issue: 1.0
Date Last Change: 17/12/2009 13:52:00
File: FS-EDEN-D411.doc
Delivery Date: 31.5.2009

Document History:

12.02.2009	Document created
13.02.2009	Version 0.1
25.02.2009	Version 0.2
09.03.2009	Version 0.3
20.04.2009	Version 0.4
19.05.2009	Version 0.5
31.05.2009	Version 1.0

Abstract

This report is a composition of two deliverables of EDEN WP4: D4.1.1 *Report on use cases for service design* and D4.1.4 *Experiments on solving use case with selected tools*. Deliverable D4.1.1 analyzes the use cases defined in the Flexible Services program to identify *elements of service design by end users*. It aims to answer the questions: What end user activities the use cases contain that would directly influence the evolution of a service ecosystem? The primary focus is on end user activity in creation of new services by composing them from existing services. The goal of deliverable D4.1.4 is to describe practical experiences with *capabilities of existing mashup tools* to address the end-user service design needs in Flexible Services use cases. Since the discussion in these deliverables is intertwined, they are combined into a single document.

The main use cases of Flexible Services program concern (1) mobile payments, (2) collecting and selling environmental information, and (3) collaborative organization of complex events such as weddings. The last one of these turns out to contain especially interesting examples to end-user service composition; a great part of analysis in this deliverable is therefore centered on it. The analysis reveals important technical elements in the use cases: (1) support for end-user development, (2) support for collaborative and incremental service composition, (3) identification of users, (4) social networks and groups, (5) location-awareness, and (6) management of shared data.

During the task T4.1.4 it became obvious that the existing mashup tools are able to address these technical requirements so poorly that prototype implementations would not be meaningful. While many of the mashup tools are, in principle, meant for relatively general service composition, in practice the functionality is focused on information integration and its subtasks such as data retrieval, data cleaning, data integration, and data visualization. In contrast, the use cases of Flexible Services turn out to deal with management of everyday life and its activities, much of which cannot be addressed through information integration tools. Consequently, there are almost no specific interesting Flexible Services requirements that could be solved with existing mashup tools.

TABLE OF CONTENTS

1	Introduction	5
1.1	Goals.....	5
1.2	Requirements.....	5
1.3	Terminology	6
2	Use cases of Flexible Services program.....	7
2.1	MoFS - Mobile Financial Services	7
2.2	Envitori - Market place for environmental monitoring data.....	8
2.3	LUCRE Wedding case – Collaborative organization of complex events.....	8
2.3.1	Basic ideas	8
2.3.2	Scenarios from weddings.....	9
2.3.3	Example: Shared todo-list with location-based reminders	12
2.4	Summary of requirements.....	14
3	Service composition tools versus the requirements.....	16
3.1	Existing service composition tools.....	16
3.2	Building blocks	17
3.3	Evaluation.....	18
4	Summary	20
5	Bibliography.....	21

1 Introduction

The question addressed by this report is how well the existing end user service composition tools could be used to create services to manage everyday life. The answer turns out to be: not well. This report is a composition of two deliverables of EDEN WP4: D4.1.1 *Report on use cases for service design* and D4.1.4 *Experiments on solving use case with selected tools*. Since the discussion in these deliverables is intertwined, they are combined into a single document.

1.1 Goals

Deliverable D4.1.1 *analyzes* the use cases defined in the Flexible Services program to identify *elements of service design by end users*. It aims to answer the questions: What end user service design activities the use cases contain? The focus is on creation of new services by composing them from existing services, not developing them from ground up.

The goal of deliverable D4.1.4 is to describe practical experiences with *capabilities of existing mashup tools* to address the end-user service design needs in Flexible Services use cases. The tools have been specified in the D4.1.2 *Description of available service composition tools*; there is also a broad survey of such tools in LUCRE D2.1: *Report on the state of the art – Part 1: Service composition*. There has been a lot of recent interest in developing tools to remix the contents and services available in the Internet, though the development of some of the promising tools has quite recently been terminated.

1.2 Requirements

The main use cases of Flexible Services program concern

1. Mobile payments (Project: MoFS)
2. Collecting and selling environmental information (Project: EnviTori)
3. Collaborative organization of complex events such as weddings (Project: LUCRE)

The last one of these turns out to contain especially interesting examples to end-user service composition; a great part of analysis in this deliverable is therefore centered on it. The functionalities envisioned in that use case also have clear uses in many related everyday contexts. The analysis reveals important technical elements in the use cases:

1. End-user development
2. Evolving service design created collaboratively in an incremental fashion
3. Identification and authentication of users
4. Management of social networks and groups
5. Location-awareness such as location-based messages and services
6. Management of shared data

During the task T4.1.4 it became obvious that the existing mashup tools are able to address these technical requirements so poorly that prototype implementations would not be meaningful. While many of the mashup tools are, in principle, meant for relatively general service composition, in practice the functionality is focused on *information integration* and its subtasks such as data retrieval, data cleaning, data integration, and

data visualization. In contrast, the use cases of Flexible Services turn out to deal with management of everyday life and its activities, much of which simply cannot be addressed by information integration tools. Consequently, there are almost no specific interesting Flexible Services requirements that could be solved with existing mashup tools.

1.3 Terminology

The title of EDEN WP4 is *user-driven service ecosystem evolution*. The terminology is explained in the following:

- *Service ecosystem* means an environment for service provisioning and consumption inhabited by multiple species of interacting participants - service providers, developers, users, and intermediaries. Within an ecosystem, the participants can publish and discover services. They can use and misuse them as well as interconnect them in unforeseen ways (Barros, 2006). The environment should provide mechanisms for the participants to communicate about services and the delivery process of services. It is worth noticing that the ecosystem is primarily inhabited by agents - companies, governmental and non-governmental organizations, individual people, and possibly also automated agents. The glue that keeps the elements of the ecosystem together is formed the services that the agents provide, request, or mediate between each other.
- *Service ecosystem evolution* means the change over the time in the services, participants, tools, and technologies of an ecosystem. In an ecosystem a service uses and is used by other services and simultaneously needs to compete from limited resources with other services. The change takes a form of co-evolution among these ingredients. A well-functioning service ecosystem can exhibit sustained evolution and a capability to adaptation to external changes. It is an active topic of research to identify the conditions - ecosystem infrastructure and interaction laws - that would enable sustained evolution in the ecosystem (Zambonelli, 2008).
- *User-driven ecosystem evolution* means evolution influenced by the users participating in the ecosystem - basically in contrast to the more traditional development of service offerings by service providers and developers. Users' influence can be direct or indirect. A user can influence the evolution directly by composing new services or by giving explicit recommendations - ratings or diggs - about services. Indirect influence means that different participants of an ecosystem can utilize the information gathered by monitoring the behaviour of users. The participants to adapt their behaviour, service offerings, service requests, and so on.

In this report we will concentrate on elements of use cases in which *end users directly influence the ecosystem evolution by composing new services*. In the analysis of the use cases we will also discuss the examples of indirect influence.

2 Use cases of Flexible Services program

In this section we will give an overview of the use cases that were identified in the different subprojects of Flexible Services program. In total, nine different use cases were identified:

1. MoFS - Strong authentication
2. MoFS - Mobile banking
3. MoFS - Payments and rewards
4. Envitori - Mobile air quality reporting
5. LUCRE - Collaborative organization of complex events (case weddings)
6. LUCRE – Context-sensitive customizable widgets
7. EDEN – User identification for XFormsDB
8. UDOI – Senior-living
9. CrossMedia - Supply chain management in tailored cross media production chain

EDEN use case 7 was provided merely as an example of the format to present real use cases. In addition, 6, 8, and 9 were not defined well enough to serve as a basis of further analysis. Consequently, the analysis in this report is limited to use cases 1 – 5.

In most of these cases, the services are more or less fixed and end users have a relatively passive role in utilizing those services. The only exception is number 5 – the collaborative organization of complex events defined in LUCRE. In this use case end users have an active role in composing new services. Large part of the analysis below is centered to that use case.

2.1 MoFS - Mobile Financial Services

MoFS has provided three use cases related to mobile payment mechanisms. The use cases concern strong authentication, mobile banking, and payment and rewards.

1. *Strong authentication.* The strong end-user authentication with the mobile device is about identifying and authenticating the end-user strongly with the help of his mobile device (with required hardware and software set-up). Obvious use scenarios for strong mobile integrated authentication are log-in to mobile internet bank, log-in to internet bank, authorising a payment, etc
2. *Mobile banking.* Enable a highly usable, interesting, simple, portable and secure two-way connection between financial institution and the user. The mobile banking is tackled from business and technical perspectives: proactive banking service model, mapping technical environment, mapping business models, and a case study for trials. MOFS aims to introduce a two-way contextual pro-active banking experience that uses unique features of mobile technology to bring relevant banking services to audiences when and where they are needed. The concrete realization of pro-active banking concept is a small piece of software on your mobile phone that is an opt-in channel for your bank to provide you data that you opted in to receive. This data can be for example e-invoice notifications and balance alerts along with contextual data that is relevant to the user at that particular time and situation. This contextual data can be for instance a reference to a simple few click service upgrade proposal for your credit card or micro loan offers based on your financial situation and other context information from the device.
3. *Payments and rewards.* CyberBank is an ObC (On board-Credentials) –based mobile micro-payment and rewarding system for mobile social media services. The goal is to develop easy way to integrate micro-payment and rewarding system to mobile services, run user studies in connection with the trials, and evaluate the user experience of the concept. Social media services were selected for the trial, because they will give good outlook of emerging payment needs, e.g. handling very small amounts of

money, complex payment schemes, and rewarding with both loyalty and activity -based non-fungible rewards.

The use case does not contain particular activities where end users would compose new services. However, The payment services of MoFS could serve as building blocks in many user-created services. This would also present composability requirements for those services.

2.2 Envitori - Market place for environmental monitoring data

Mobile air quality reporting is a use case in which users can report their assessment of the local air quality using a mobile device. The use case concerns a new air quality service for consumers, and a way to connect air quality information with personal health. Nowadays, there is only few locations where air quality is measured in real-time. User-provided reports would complement that data. The problem in the use case is to relate air quality with personal health. That could be in form of a new type of service for special groups like asthmatics living in an urban area, and also for average consumers that are concerned about the environment.

Users are involved to act as sensors themselves and collect subjective observations about air quality. These observations are compared with measured and modelled air quality values in order to improve models. Observations from sensors and people are presented together with model predictions on the screen. A mobile user interface enables observation gathering and the presentation of air quality data and services.

Value-adding services can be developed to alert about bad air quality or inform about good air quality in selected locations, suggest routes that avoid areas of bad air quality, and calculate what kind of air the user has been breathing during a day or a week for example.

This use case does not contain activities in which end users would create new services. However, it contains elements of user-generated content, since users act as air quality sensors. There is also a possibility that users create a new environmental measure and start a collection of observations for determination of that measure.

2.3 LUCRE Wedding case – Collaborative organization of complex events

LUCRE has develop a use case on a *wedding organizer*, a service to help participants coordinate their activities relating to a wedding. The service is designed collaboratively and in an incremental fashion by the wedding participants themselves during the wedding arrangement process. First, new wedding organizer is created, then key people are added and their roles and relationships specified, groups for different purposes formed, lists of things to do or to acquire are specified and attached to groups, and so on.

2.3.1 Basic ideas

The service allows the creation of an organizer service that will collaboratively be refined and enriched over the time by participating people. Participants can add different elementary functional components – groups, shared lists, wiki areas, media, subevents, and so on – and relations between these components. After each addition the service provides slightly different functionality also for other participants depending on their roles and access rights. The evolving service helps participants to coordinate their activities that relate to the weddings. Each modification to the service – for example, the addition of a group, a todo list, or a reminder – is a small service design action.

The key functional elements that the participants can use are (1) people including their social relations and roles in the wedding organization, (2) groups of participants, (3) shared todo lists, checklists and acquisition lists attached to groups, (4) discussion channels and wiki areas attached to groups, events, or artefacts, (5) location-based messages, (6) media content shared after the wedding with other participants.

For each wedding the resulting service that evolves during the wedding arrangement process would end up in a different composition of the elements than in other weddings. In addition, the view of each participant is personalized based on his or her roles, membership in different groups, and access rights.

Wedding is an instance of a more general problem of *collaborative organization of a complex event*. There are multiple phases or subevents such as bachelor's parties, the wedding ceremony, wedding feast, and after-party. There are many different things to arrange or coordinate: invitations, table placing, wedding dress, banqueting room, church, gifts, photography, flowers, and so on. A large group of people is involved; people are in different roles - bride, groom, bestman, maid of honor, bride's father, - and there are many subgroups - e.g. those organizing bachelor's party, those arranging program for wedding feast, and so on.

There are other examples of collaborative organization of complex activities: holiday trips with friends, fraternity parties, family reunions, rummage sales, bees, major anniversary parties, and many events related to voluntary work. In each of these there are things that need to be taken care of before the event (e.g., reservations and preparations), during the event (coordination, communication), and after the event (e.g. media sharing).

Characteristic of collaborative organization is that there is not necessarily any particular person who is in charge of everything. Much of the organization is carried out in voluntary manner and characterized by self-organization. The focus in this use case is in solutions that could support such organizational activities without (complete) centralized control.

2.3.2 Scenarios from weddings

Below we go through a set of scenarios within the composer of wedding organizers.

A. Sensitivity to time and context

Elina and Akseli are getting married. Few months before the event, maid of honour Anna uses Lucre tools to set up a service that would assist the participants before, during and after the event. There are a number of things they expect to be able to organize and Lucre system has a examples or templates for each of these:

- Before the event
 - Sending invitations
 - Planning a bachelor party
 - Coordination of gift purchases
- During the event
 - Schedules, transportation
 - Games & plays
 - Official media (e.g. by a hired photographer)
 - Photos taken by the guests
- After the event
 - Reminiscing
 - Commenting media
 - Ordering photographs
 - Sending greetings
 - Contacting new acquaintances

Requirement:

- Creating and setting up a wedding organizer
- Access to examples or templates to guide the definition of different aspects of the weddings
- Availability of features depend on time and context

B. People and their roles

People to be invited to the wedding and those that will take part in the organization of weddings can be entered into the system. Their relationships can be specified using the friend-relations as in social networking services. The roles of different participants can be specified.

Requirements:

- An invited person can sign in to the system and become a participant
- A participant can create and modify a personal profile and relations to others
- A participant can specify his or her role in the wedding

C. Groups and access rights

The bestman creates a secret group for bachelor's party and invites a number of groom's friends to the group. The bride of honor sends invitations to a bachelor party planning session, inquiring about possible times.

Participants have different access rights and customized views to the service, depending on their role in the event, for example:

- Bride and groom get to build their wish list, send invitations to the participants, and interact with venue contacts, catering services, orchestra, photographer, etc.
- Maid of honour and best man can coordinate the programs of bachelor parties, invite participants, etc.
- Guests sign-up to the event, inform the catering service about their dietary restrictions, use the gift list service to avoid duplicate gifts and to see what the couple wants, etc.
- Venue organizer provides driving/transportation instructions to the site and layouts of the interior for the couple that plan the seating order of guests.
- Catering service has access to up-to-date schedule, list of participants, and their dietary preferences
- The official photographer links her services to the wedding service..
- Participants link/mashup their personal media sites and content to the service

Requirements:

- A participant can create a group and specify whether it is
 - Public: anyone can see it and join it
 - Closed: anyone can see it but joining requires approval
 - Secret: not visible, join only by invitation

D. Shared lists

To distribute the workload related to the program, a group of organizers is formed. A shared todo list is created and attached to the group. Members can create and add tasks to the todo list. They can reserve tasks that they intend to do. After finishing a task it can be marked completed. A reserved task can be returned to the list if cannot be finished.

The bride and the broom compile a gift list, including linkages to department store and product web pages. The gift list is attached to the group of all wedding guests.

Requirements:

- Possibility to create and modify checklists, todo lists, and gift lists
- Simple management of tasks and time (Doodle-style)

E. Messages, discussions, and wiki areas

After the bachelor party planning session, the guests use the service to continue discussion on details and new ideas that have emerged.

Venue provider sends an image with layout of the dinner tables. The bride reorganizes some tables and starts to plan the seating order.

Requirements:

- User groups, including access rights for individuals and groups
- Generic discussion or messaging areas or other such tools to support unstructured activities
- Instance messaging, micro blog, or wiki-like textual interface, links to external services
- Image overlays; basic drawing operations on top of images.
- Updated/changed/approved by tracking

F. Budgeting and cost sharing

The costs of expensive items can be shared by multiple people. Guests can also add their own gifts to the list.

Requirements:

- Sharing of costs

G. Group-aware location-based messaging

At some point during the evening, selected participants receive a notification message about kidnapping of the bride. Some guests distract the groom and the bride is kidnapped..

All guests that are in the area receive a message to gather together to assist the groom through a series of tasks to free the bride. Guest who left earlier or attended only the ceremony in the church do not receive these messages.

Requirements:

- Quick group formation based on user's social network
- Messages or notifications can be sent to all or selected people inside an area.
- Different types of notifications. Obtrusiveness depending on the importance of the message.

H. Media sharing and content aggregation

During the event, the event service provides an aggregated view to media created and posted by the guests to various social media and networking sites.

The participants can also see a low-resolution preview of the photos the photos taken by the official photographer, and buy high-resolution versions and prints.

Requirements

- Photo or video publishing
- Preset accounts or easy sign-up for new users
- Tools for adding links or feeds, in and out

- Necessary search features
- Media access control: yes/no/preview/other?
- Sign up to external service, including the management of payments
- Sharing of media with members of a social network or some particular group

I. Sub-events, ad hoc coordination, and voting

After the event, some participants still feel like going out to a bar/nightclub downtown.

An invite is sent, with possible places to go to and a backup plan. Because there is no coordinated transportation, the attendees can enable location sharing to let others know where they have gone.

Requirements

- Cascading/extendable event structures
- Basic mobile editing functionalities or/and wizards
- Voting
- Network-side location sharing for a limited time, for a specific group of people.
- Text and map, combined with status line.

2.3.3 Example: Shared todo-list with location-based reminders

The scenarios A-I presented in subchapter 2.3.2 above address different aspects and subevents of weddings. Due to the breadth of scenarios presented we will now focus on one specific service that users might want to create, and work out the details of that scenario. That subscenario is called *shared todo-list with location-based reminders*. This scenario is a distillation of the key functionality of wedding organizer but, at the same time, it is also adaptable to many everyday contexts such as managing the chores within a family. After all, LUCRE belongs within the Flexible Services program into the research area of Everyday services. This is an example of the kind of service that a user should be able to create with a service composition tool from services that provide group management, todo-lists, and location-based messages.

A shared todo-list is associated with a *group* and maintains a *list of tasks* that the group needs to do. In the normal mode, the members *can see the tasks* that need to be done, and also some of the recently completed tasks. A member can sort and view the pending and completed tasks in different ways. A member of the group can independently *add new tasks* to the list and *do tasks* from the list. The execution happens in two stages: first a member *reserves* a task for execution (lock to avoid accidental concurrency) and after that either marks the task as *completed* (commit) or *releases* it back to the todo-list (abort). A task can have associated locations important for its execution. When a member of a group moves into a location he or she will get reminder of any uncompleted task not reserved to someone else related to that location. There is a whole range of other functionalities that a shared todo-list might contain, dealing with detailed properties of tasks (due dates, estimated durations, or precedence relations to other tasks), group awareness (overall status of the tasks, work-in-progress, who has done how much), multi-person tasks (reserved by two or more persons), and so on.

There are different kinds of groups:

- *Long-living groups* – typically created explicitly and have a clear identity, such as a name or description (“decoration group”). In the public/private –dimension a group can be public (free for anyone to join), closed (joining requires an approval from the group), or secret (joining only by invitation). Group has often an “owner” who represents the group towards the external parties and accepts requests to join.
- *Temporary groups* – typically created implicitly in connection of a shared activity such as people sharing a car in car pooling, people sitting in same table, and so on.

The requirements for shared todo-list with location-based reminders are the following specified for group management, todo-list, task, and location-based reminders.

Group management

Operation	Priority
Create group	High
Add a person to group	High
Remove a person from a group	High
Associate group with a todo-list	High
Add/remove owner (Explicitly created group)	Medium
Invite person (Explicitly created group)	Medium
Ask the approval to join (Closed groups)	Medium
Approve person (Closed groups)	Medium
Remove a group	Low
Merge or split groups	Low
Create subgroups or supergroups	Low

Notes:

- The semantics of operations that merge or split groups is problematic and needs to be carefully analyzed and specified.
- The people to invite to a group should be easy to specify by filtering or picking from a social network.

Todo-list

Operation	Priority
Create a list	High
Add/remove task	High
Reserve/release task	High
Confirm task completed	High
Show tasks (with a filter)	High
Sort tasks (based on a criterion)	Medium

Task

Operation	Priority
Specify a location of a task	High
Specify the urgency (e.g. due date)	Low
Specify estimated amount of work	Low
Specify the priority	Low

Location-based reminders

Operation	Priority
Follow the locations of group members	Medium
Alert a member of a task when in location specified in a task	Medium

Notes:

- The above components can be combined with group awareness widgets to display who is doing what, and how much each member has done.
- There could be generators for repetitive tasks when the shared todo-list is applied in everyday life . For example, go to market once a week, take kids to day care every morning of a working day, get kids from day care every afternoon of a working day, etc.

2.4 Summary of requirements

The technical elements identified in the analysis above can be grouped into following categories:

1. *End-user development.* The environment and available actions should be designed for end users in mind.
2. *Evolving design created collaboratively in an incremental fashion.* The design of the organizer for a particular wedding proceeds in a distributed fashion by invited participants when they add and connect new functional elements – groups, lists, discussion forums, subevents, and so on – to weddings.
3. *Identification and authentication of users.* All of the use cases analyzed in this document – from MoFS, Envitori, and LUCRE – require the identification and authentication of users. The contents of the service are not anonymous, i.e. same for every user but depend on the identity. Many actions such as creation of content or modification of shared data also require that users have proper access rights. The payment-related use cases of MoFS even require the use of strong authentication methods.
4. *Management of social networks and groups.* Management of everyday activities requires the possibility to access, create, and update one’s social context in a service. For example, the creation of groups for different purposes would be difficult if the information of each person needs to be entered to the service from scratch. In some of the wedding scenarios the creation of groups need to happen on-the-fly. Many services are best implemented as belonging to a group, e.g. with the access rights determined by a group membership. Social networks are an important basis also for the establishment of trust in services as well as the basis of recommendations presented to users.
5. *Location-awareness.* The possibility to make actions based on the locations of users is important in some of the use cases. Examples are location-based messages in wedding scenarios and location-based reminders in LUCRE scenario of shared todo-list with location-based reminders. Location-awareness is naturally primarily related to the mobile use of services. In Envitori the users’ observations of air quality happen when they move around and each observation should be automatically associated with its location.
6. *Management of shared data.* An essential requirement in shared todo-list and shared acquisition lists is the proper management of data shared by the members of a group. Members should have the access to the data, the – possibly restricted – ability to manipulate it, and even enough concurrency control so that the integrity of data would not be compromised due to concurrent access. Ultimately many services would also benefit from a proper server push –functionality; that is, when one user updates shared data, other users would immediately have their views to the data updated as well.

In addition to these main requirements there are many secondary requirements. Among others, they concern the support for the following:

- *Dynamic association* of groups with discussion channels, shared workspaces, polls, and so on.
- *Transactions*, especially in activities associated with payments.

- *Media sharing*, connected to media service providers.
- *Advanced group operations*, such as merging and splitting groups.

3 Service composition tools versus the requirements

In this chapter existing service composition tools are compared against the requirements identified in the previous chapter. They give an idea of what could be building blocks in the composite services, some relevant services available in the Web also listed.

3.1 Existing service composition tools

Existing service composition tools have been surveyed in the EDEN Deliverable 4.1.2 and LUCRE Deliverable D2.1: Report on the state-of-the-art, Part 1: Service composition. The discussion below is based on these reports.

Mashup tools can be divided based on whether the services are composed during *design-time* (in advance of execution) or at *run-time* (combining services on-the-fly). Run-time compositions are created ad hoc and the result is typically a throwaway service, not a reusable service artefact. While there are some run-time oriented tools, most notably Intel MashMaker¹ (Ennals, 2007) and Mozilla Ubiquity², most of existing mashup tools belong to the design-time category. A need for on-the-fly composition is, however, present in some of the use cases.

When new service artefacts are created (in design-time mode), the basic approaches have been classified in (Kotkaluoto, 2009) as follows:

- *Programming-by-example* is a title that is actually used from two somewhat different approaches:
 - *Programming-by-demonstration*. Users operate with services in an environment that can record their actions. The environment generalizes the actions of users and as a result creates a more generic service. Mashup tools belonging to this category are Karma (Tuchinda, 2008) and Vegemite (Lin, 2009).
 - *Example modification*. Users are able to take an existing service and modify it to better suit their needs. One recent tool in this category is d.mix (Hartmann, 2007) that helps the users to copy the behaviour of existing services through so-called site-to-service maps that tell which visual areas of the user interface have been produced by which API calls.
- *Visual programming* is a method of specifying service compositions with a graphical editor. The services are programs that are shown to the user in visual forms, e.g., consisting of components and pipes representing information flows between the components. A complex graphical editor is required to manage the structure of the program and the relationships of components. Most notable examples of such tools are Microsoft Popfly³, Yahoo Pipes⁴, and IBM Lotus Mashups⁵. In the editor the components are viewed from outside and connected with each other using a graphical notation. The user works from an

¹ <http://mashmaker.intel.com/web/informationguest.html>

² <https://wiki.mozilla.org/Labs/Ubiquity/>

³ <http://www.popflywiki.com/>

⁴ <http://pipes.yahoo.com/>

⁵ <http://www.ibm.com/software/lotus/products/mashups/>

operation-centric view and often needs to understand basic programming concepts, although the notation may be easier to work with than with ordinary programming languages.

- *Form-based creation.* The tools in this category are not generally called mashup tools. The idea is to install widgets – services with small embeddable interface components – to a dashboard. The widgets reside in the dashboard side-by-side. From the user interface perspective the result is like a personal portal or a collection of related services than one unified service. Often the widgets on a common dashboard do not even share information with each other, and it is therefore questionable if the result can be called a composite service. However, there are highly successful environments in this category, such as iGoogle⁶ and NetVibes⁷. An advantage in the approach is that widgets typically allow the manipulation of the data that they are showing.
- *Script-based creation.* Users can program services using flexible dynamic languages. Examples are Firefox Extensions⁸, Facebook Applications⁹, and Google App Engine¹⁰. While these kinds of environments may make it easier to develop applications when compared to plain old-fashioned programming, in practice these tools are out of reach of most end-users.

3.2 Building blocks

There are many useful component services that could be used as building blocks in the services envisioned in the Flexible Services use cases. Examples are:

- *Social networking.* Services to create, modify, and access personal profiles and social relationships. There are several such services available – for example, Facebook, MySpace, LinkedIn, Friendster, Twitter, and so on – but the possibilities to access the social network data can be limited. One accessible service is the OtaSizzle social networking service.
- *Group formation:* Ranging from explicit creation of long-living groups that have a definite identity to implicitly creation of short-living groups based on collaborative tasks. This is not generally supported in the existing social networking services. An example of a service that supports group formation is OtaSizzle group formation service.
- *Group awareness:* Display of the status of shared entities or lists in a easy-to-use but non-intrusive manner. For instance, showing the status of todo list in green, yellow, or red depending on whether the tasks have been taken care of, are in danger to be left undone, or have already been missed.
- *Group communication:* Flexible, task-specific, and non-intrusive communication mechanisms - message channels, blackboards, and so on. An example is OtaSizzle channel service.
- *Shared todo lists:* A list of tasks that need to be done by someone in a group. An example is the Tadalist¹¹ service.

⁶ <http://www.google.com/ig>

⁷ <http://www.netvibes.com/>

⁸ <https://addons.mozilla.org/>

⁹ <http://www.facebook.com/apps/>

¹⁰ <http://code.google.com/appengine/>

¹¹ <http://www.tadalist.com>

- *Shared acquisition lists*: A list of entities that need to be bought or rented for the event. An example is Warp¹².
- *Shared calendars*: Display of common events of a group in a calendar. An example is Google Calendar¹³.
- *Meeting scheduler*: Service to find a common meeting time. Examples are Doodle¹⁴ or Sumpli¹⁵.
- *Voting/survey*: Service for determining the group opinion about an issue.
- *Budgeting and cost sharing*: Services that allow the planning and sharing of costs.
- *Media sharing*: There are many services available, such as YouTube, Flickr, Digg, Facebook, ...

Many of these kinds of component services might be generalized later on to work for everyday family activities - e.g. managing chores with shared todo list within a family, or using shared acquisition lists as shopping lists. The main difference in the everyday activities is their recurrence. This could be addressed with mechanisms such as generators of daily or weekly activities to the todo-list or generators for the basic contents of a shopping list.

3.3 Evaluation

The question now is, how could the service design aspects of the use cases of Flexible Service be realized using the existing service composition tools. Or in concrete terms, how could the requirements enumerated in the subchapter 2.4 be implemented with tools mentioned in subchapter 3.1 using, for example, using building blocks like those mentioned in subchapter 3.2?

The primary requirement is (1) the ability of end-users to design and create new services, and more specific requirements are (2) identification and authentication of users, (3) social networks and group management, (4) location-awareness, and (5) management of shared data. They all turn out to be challenging, if not out of reach of the tools. Below we look at these in more detail.

- *End-user development*. There are different kinds of end users. At one end, anyone who is able to use computer or a mobile phone – e.g. browse a web or send text messages – can be called an end user. At the other end, office workers capable of creating spreadsheet programs – i.e., spreadsheets with non-trivial formulas – are cited as a primary example of end-user programming. However, spreadsheet programming can safely be regarded as most demanding type of end user programming; anything that is more difficult than that, such as the use of more general programming languages or program development environments – is clearly outside the realm of end user development. In that scale, the existing tools are located in different positions. Programming-by-demonstration, such as that supported by Karma or Vegemite – is accessible to most naïve users. Also the form-based creation, like in iGoogle and NetVibes, is clearly possible to large end user populations. Visual programming, in the style of Yahoo Pipes, Microsoft Popfly, or IBM Lotus Mashups – is already significantly more difficult, since it assumes basic understanding of programming concepts, like data structures, iteration, filtering, and so on. It is based on operation-centric view that is foreign to end users. Script-based creation, like the

¹² <http://www.wrapd.com/>

¹³ <http://www.google.com/calendar>

¹⁴ <http://doodle.com/>

¹⁵ <http://sumpli.com/>

service development with of Firefox Extensions, Facebook Applications, or Google App Engine, is in most cases completely incomprehensible for end users, even to those who might be able to create spreadsheet macros.

- *Evolving design created collaboratively in an incremental fashion.* This is a mode of service composition that none of the identified tools support. There are use-time mashup tools – like Intel MashMaker and Mozilla Ubiquity – designed mainly for the creation of single-use, throwaway service compositions while browsing the Web. Most mashup tools are, however, meant for use before the execution of a service to create a reusable service artifact. The incremental collaborative creation of a service is a some kind of mixture of these modes. Many of the composition operations are done at run-time but the result is persistent and immediately visible to other users. A widget-based framework that supports shared dashboards would be close to this mode of operation.
- *Identification and authentication of users.* This capability is available in complex environments such as iGoogle, Facebook, or Google App Engine. Simple tools like Karma or Vegemite do not identify users. In iGoogle the realm of single sign-on covers the widgets produced by Google itself.
- *Social networks and group management.* The access to social network data is becoming possible through interfaces like Open Social or Google Friend Connect which to provide an access to social network data maintained by services that support the interfaces. However, there are other non-technical concerns that may hinder the use of social networks in user-created services. Companies maintaining social networking services may want to protect the privacy of their users. The social network data is also considered an important asset of these services, and the commercial interests may make the companies jealous of revealing the data. For example, large-scale access to social network data may be prohibited by the services. This may make it difficult to use outside of a service in a meaningful way. Finally, the capabilities of creating groups of people in social networks – that is one of the central requirements of the wedding use case – are limited in existing social networking services.
- *Location access.* The access to client-based location is generally problematic in mashup tools. A mashup typically uses a set of fixed services found on the Web. The access to the information in the mobile client would require that the mashup is able to use a service residing in the client device it is been running in. This is not impossible but requires specific access mechanisms. Mashup tools like Karma, Vegemite, and Yahoo Pipes do not have any support for this. Another approach to location access is to use operator positioning. In that approach the Internet operator maintains the data about the current position of a mobile device and makes it available for authorized users through a specific service on the Web. The approach is indirect and requires the users to participate the particular positioning service. Some users may not be willing to do that due to privacy concerns.

4 Summary

The main use cases of Flexible Services program concern (1) mobile payments, (2) collecting and selling environmental information, and (3) collaborative organization of complex events such as weddings. In this report we identified those aspects of the use cases that require service design by end users. The wedding use case contains many examples to end-user service composition. The key requirements that a service composition tools should support are: (1) end-user development, (2) collaborative and incremental service composition, (3) identification and authentication of users, (4) social networks and groups, (5) location-awareness, and (6) management of shared data.

The existing mashup tools are not able to address most of these technical requirements, and none of them are able to address them all. While many of the mashup tools are, in principle, meant for relatively general service composition, in practice the functionality is focused on information integration and its subtasks such as data retrieval, data cleaning, data integration, and data visualization. In contrast, the use cases of Flexible Services deal with management of everyday life and its activities, much of which cannot be addressed through information integration tools. Consequently, there are almost no specific interesting Flexible Services requirements that could be solved with existing mashup tools.

There is clearly a need for more research and development for these kinds of tools. Within Flexible Services program this work takes place in LUCRE and EDEN WP4. LUCRE is developing a service composer that addresses these problems, and the EDEN WP4 develops basic technologies for connecting services together in service composer.

5 Bibliography

Barros, Alistair P. and Dumas, Marlon. „The Rise of Web Service Ecosystems.“ *IT Professional* (IEEE Computer Society) 8, Nr. 5 (2006): 31-37.

Ennals, R., and Garofalakis, M. (2007). MashMaker: Mashups for the Masses. SIGMOD'07. ACM.

Hartmann, B. and Wu, L. and Collins, K. and Klemmer, S.R. (2007) Programming by a sample: rapidly creating web applications with d. mix, Proceedings of the 20th annual ACM symposium on User interface software and technology, pp. 241--250, ACM New York, NY, USA

Kotkaluoto S., Leino J., Oulasvirta A., Peltonen P., Räihä K., and Törmä S. *Report of the state of the art – Part 1: Service composition*. Flexible Services Deliverable LUCRE D2.1, 2009.

Lin, J., Wong, J., Nichols, J., Cypher, A., and Lau, T. A. (2008). End-user programming of mashups with vegemite. In Proceedings of the 13th international Conference on intelligent User interfaces (Sanibel Island, Florida, USA, February 08 - 11, 2009). IUI '09. ACM, New York, NY, 97–106.

Pohja, Mikko. *Description of Available Service Composition Tools*. Flexible Services Deliverable EDEN D4.1.2, 2009.

Tuchinda, R. and Szekely, P. and Knoblock, C.A., Building mashups by example, Proc. 13th int'l Conf on Intelligent User Interfaces, ACM, 2008.

Zambonelli, Franco and Viroli, Mirko. „Architecture and Metaphors for Eternlly Adaptive Service Ecosystems.“ *Studies in Computational Intelligence - Intelligent Distributed Computing, Systems and Applications*, 2008.