A Service-Oriented Architecture for Web Applications in e-mental health: two case studies

Frank J. Blaauw[†][*]
[†]Distributed Systems Group
Johann Bernoulli Institute for
Mathematics and Computer Science
University of Groningen
The Netherlands
Email: f.j.blaauw@umcg.nl

Ando Emerencia[*]
[*]ICPE
University Center for Psychiatry
University Medical Center Groningen
University of Groningen
The Netherlands
Email: a.c.emerencia@rug.nl

***Abstract*—E-mental health applications have gained popularity recently as a result of applying developing technologies to leverage advantages over traditional care. First, e-mental health applications are inherently scalable, whereas traditional care involves one-to-one relations between patient and clinician. Second, electronic data formats can facilitate interoperability in a way that medical data on paper cannot. Third, the flexibility of a web application warrants that improvements in the care program exposed through the application will immediately benefit all applicable users.**

For the specific domain of e-mental health, the primary source of medical data are online questionnaires, and most applications in this domain focus on viewing, filling out, and managing these questionnaires. Many distinct such applications have been developed in recent years, most of them starting from scratch and running in isolation.

In this paper, we design and propose a generic architecture for e-mental health applications from two case studies. These case studies are two e-mental health applications called HowNutsAreTheDutch and Leefplezier, and are used on a large scale with over 13,000 active users combined. By abstracting functionalities into reusable Service-Oriented Architecture interfaces, we can maximize data interoperability while minimizing application-specific code to facilitate rapid development of e-mental health applications.

***Index Terms*—Service-Oriented Architecture, Service-Oriented Computing, Mental Health, eHealth, Service Modularity, Questionnaire-based Research**

I. Introduction

eHealth is a term used to describe the use of information and communication technology (ICT) to support health-care, to perform health-care, or to carry out health-care related research. In recent years, the advantages of scalability, interoperability, and flexibility that are associated with well-designed eHealth applications have convinced many of the feasibility of eHealth in clinical practice.

While most physical ailments have objectively measurable symptoms, mental disorders are typically subjective and complex, and clinical practice has come to rely on establishing their presence through sets of validated questionnaires. The results of these questionnaires are summarized and used to inform professionals or to derive disorder classifications.

Within the field of e-mental health research, a flurry of distinct applications centered on deriving information from these questionnaires has been developed. Applications designed for research projects in particular tend to focus primarily on collecting data and not on having a clean architecture or incorporating reusable components. This neglect can be attributed to the fact that the success of a research project is usually not measured by the quality of the application but rather by the data that is collected. Furthermore, since many such research projects have predefined budgets and time limits, application development that supports reuse tends to be an afterthought. The result is that, rather than incorporating reusable components, these projects create one-off applications, each starting from scratch and learning the same lessons, and each being costly to develop.

In light of these concerns, we believe that incorporating external service components and reusable application logic in a generic architecture will reduce development time and cost compared to those of one-off projects. The present work proposes a generic architecture for e-mental health applications based on two case studies. This architecture applies Service-Oriented Architecture (SOA) and Service-Oriented Computing (SOC) paradigms for its core functionality. An architecture for both case studies is explained, and we give an overview of the pros and cons of the decisions made during the design process. The novelty of this work resides in the fact that such an architecture has, to the best of our knowledge, never been provided in the field of e-mental health research, especially with focus on SOC.

II. Related work

The importance of accessible and affordable care for mental disorders cannot be overstated. Approximately 41.2% of the people in the Dutch population experiences at least one DSM-III-R disorder in their lifespan [1]. The symptoms experienced range from feeling tired and having a lack of interest, to experiencing strong feelings of depression. Such episodes can have a great influence on the well-being one experiences.

While numerous research platforms exist in the field of e-mental health, only a few of the papers reviewed shed light on the architecture of the used research application. Griffiths and Christensen give an overview of several on-line mental health assessments and Internet based treatments [2]. The authors conclude that the e-mental health assessments appear to provide a promising means for mental health self management. Donker et al. review several mobile applications used to deliver mental health assessments [3]. They conclude that mobile phones are well suited to give basic advice on mental health.

SOA and SOC are two relatively new concepts in the field of e-mental health and health research. Research platforms are often basic and do not consider any elaborate (service-oriented) architecture. We found only a few studies that describe the use of these concepts. Kazi and Deters describe an architecture for a pain diary study, using various SOC principles. They found that the combination of web technology and a REST architecture allows for the development of a reliable and secure health information system [4], [5].

In the broader field of eHealth, SOA is more prevalent. One example is the research by Savini et al., who describe an eHealth SOA platform, designed to connect patients to their clinicians by using mobile devices [6]. They found that SOA is a useful technique for their platform, because (i) it allows them to have a centralized place for business logic and (ii) it allows them to carry out complex calculations on a specialized server, instead of on a less powerful mobile device.

Some research exists with regards to safely storing health data in a SOA environment. Fan et al. describe DACAR, a secure data storage by means of a single point of contact (SPoC) [7]. DACAR uses a SOA to support integration of eHealth services.

The described works show that SOA/SOC concepts are viable and have made an impact in certain branches of eHealth. However, SOA/SOC concepts have not seen widespread adoption for e-mental health applications focusing on questionnaires specifically. As a result, these applications are created as isolated, one-off solutions.

The present work evaluates the architectures of two of our e-mental health applications. For each of these applications, we analyze the degree of SOC applied and motivate the design decisions. Furthermore, we present a generic architecture, based on these case studies. The first of the two case studies was intended purely as a one-off project, the second case study, however, was designed to use more reusable components. Analyzing the differences and commonalities between these case studies could give us an insight in which elements could be important in a generic e-mental health platform.

III. Two case studies

The present work evaluates two real-world applications in e-mental health research: (i) *HowNutsAreTheDutch* (Dutch: HoeGekIsNL) and (ii) *Leefplezier* (translation: joys of living). Both platforms enable e-mental health research on a moderate to large scale. The first platform was designed to get an insight in the psychological well-being of the Dutch population. The second platform focuses on well-being of elderly people in the Netherlands. An overview of the architecture and degree of SOC of both applications is provided in Section III-A and Section III-B. A comparison of both architectures is performed to derive a generic architecture. In Section III-C the technical details of both platforms are described.

A. *HowNutsAreTheDutch*

HowNutsAreTheDutch (HND) is a web application created as the platform for a national Dutch mental health research project [8], [9]. The goal of HND is threefold. The goal is to show that (i) mental health is dynamic (i.e., it can change over time), (ii) mental disabilities are not binary, one can experience various degrees of them (i.e, not all-or-nothing), and (iii) mental anomalies can also have positive implications.

In the HND application, people can fill-out various psychological questionnaires and automatically retrieve feedback on these questionnaires. This feedback is a comparison with the average Dutch populations combined with personalized diagrams and figures, depending on the questionnaire. HND offers two types of studies: (i) a cross-sectional study and (ii) an individually repeated *Ecological Momentary Assessment* (EMA) study (referred to as *diary study*). In the cross-sectional study, participants can fill out a number of questionnaires from a predefined set of questionnaires. These questionnaires contain, but are not limited to, questions regarding well-being, mood, and personality. For an extensive list of questionnaires, see [9]. After filling out a questionnaire, people get automatically generated feedback with which they can compare themselves with the other HND participants.

The second study available on HND is the diary study. The diary study focuses on variance within a person by letting them fill out the same questionnaire multiple times. In the diary study, participants are measured for thirty consecutive days, by filling out a questionnaire three times per day. The questionnaire consists of 43 questions measuring certain psychological constructs, as well as somatic and physiological aspects that could influence the psychological constructs. More information about the diary study can be found in [9], [10]. The participants in this study are notified via an SMS when they should fill out their next questionnaire. The SMS contains the link to the web application, on which the questionnaire can be filled out.

Figure 1 gives an overall impression of the HND architecture. The architecture consists of an application built upon various local libraries and several external services. Most functionality in HND is built into the application. Authentication and questionnaire conduction is implemented using two off-the-shelf libraries. The questionnaire data is stored in a database separate from the authentication information and user specific data (demographical information and email addresses) in order to limit the risk of a security breach affecting both types of data. The diary study uses two SOC services. Firstly, a service is used to perform the scheduling of the questionnaires. This service schedules the notifications and does a callback to HND whenever a participant should be notified. Secondly, when a participant has completed his or her

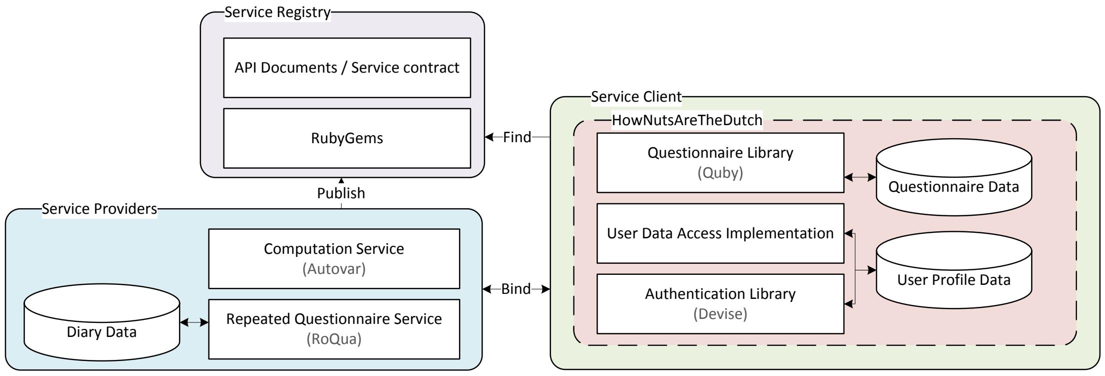

Figure 1. General overview of the HowNutsAreTheDutch architecture

diary study, personal feedback is automatically calculated. In order to calculate this feedback, an existing service known as *Autovar* is used [11]. Autovar calculates various statistics and returns the raw results to HND. Autovar calculations can be started using a web-service. The feedback based on the results returned by Autovar is rendered using client-side JavaScript.

B. Leefplezier

Leefplezier is a project that focuses on enhancing and sustaining the well-being of elderly people [12]. The notion of well-being has been an interest in gerontology for a long time [13]. Elderly people are a frail group with respect to well-being. For example, they report more longstanding illnesses (e.g., in the United Kingdom [14, p. 102]), they have to cope with the loss of freedom and mobility (e.g., the amount of physical disabilities per age group in the Netherlands [15]), and have a chance of becoming isolated [16], [17].

Like HND, the Leefplezier project is divided into two main phases, (i) a cross-sectional overview phase and (ii) a personalized analysis phase. For the second phase, a mobile application was developed. The architecture of Leefplezier is shown in Figure 2. We developed Leefplezier a year after developing HND, and in its architecture we incorporated some of the lessons learned.

Compared to HND, the Leefplezier architecture depends much more heavily on external SOC services. Both the cross-sectional part and the diary study part are implemented using SOC. Authentication of users is provided by a *single sign-on* approach. Leefplezier participants authenticate themselves to a third-party application and are provided access Leefplezier via keyed-hash message authentication code (HMAC). Storage of demographical data is provided by a separate service. The calculation of diary study results is performed using the Autovar service. However, in Leefplezier, the Autovar service (as described in Section III-A) is wrapped into a visualization service. This visualization service takes care of running Autovar and also renders the feedback of the questionnaires. The only part customized for Leefplezier is the content management system and the business logic to connect the services.

C. Technical overview of both case studies

The case studies share several commonalities. For example, communicating and authenticating with external services and their encompassing frameworks. The communication with external services is performed using a *Representational State Transfer* (REST) architecture. A REST architecture uses the Hypertext Transfer Protocol (HTTP) for accessing the external services (e.g., GET, PUT, POST and DELETE calls). These calls are sent over a Secure Socket Layer (SSL) to ensure privacy and security. Binding to these services is secured by using server-to-server authentication. Three methods of authentication are used (depending on the service): (i) HTTP Basic Authentication, (ii) Open Authorization (OAuth) and (iii) Keyed-hash message authentication code (HMAC). HTTP Basic authentication and OAuth are used to do the general communication to the service providers. HMAC is used to allow external applications to access the Leefplezier application, such as the single sign-on service.

Both case studies are implemented in the open-source web framework Ruby on Rails[1]. Ruby on Rails is a web framework that allows for fast development of web-applications, but also has numerous plug-ins available. These plug-ins (called *Gems* in Ruby parlance) are self-contained packages providing functionality that can be used by the application including them. HND and Leefplezier use Gems in order to provide a clean interface to the SOC services. The HTTP communication to the services is abstracted by the Gems, by providing a simple Application Programming Interface (API) to the developers. Gems can be distributed via standardized services, such as RubyGems[2]. RubyGems is in this case used as a service registry.

The components of both architectures that have not yet been discussed, including the applications used to provide some of the functionality (e.g., Quby and RoQua), are elaborated in the next section.

[1]Website: http://rubyonrails.org
[2]Website: http://rubygems.org

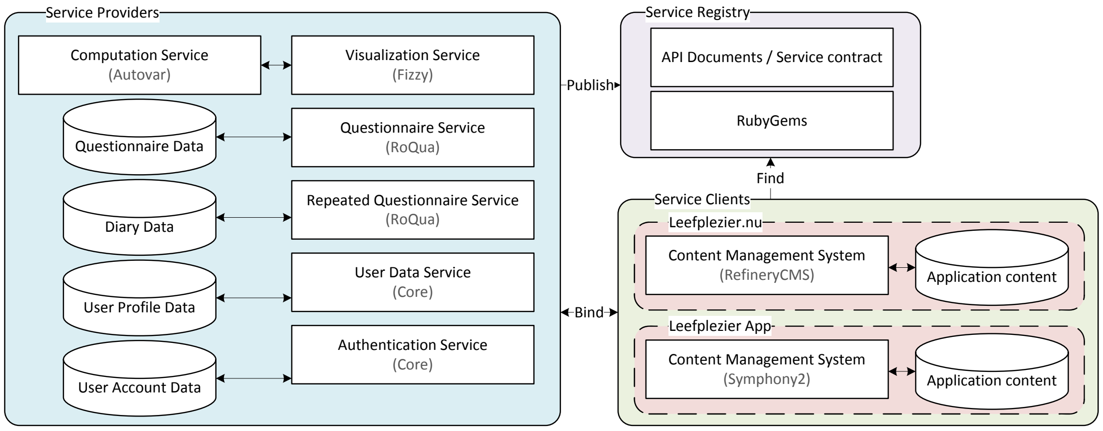
Service Providers
Computation Service (Autovar)
Visualization Service (Fizzy)
Questionnaire Data
Questionnaire Service (RoQua)
Diary Data
Repeated Questionnaire Service (RoQua)
User Profile Data
User Data Service (Core)
User Account Data
Authentication Service (Core)
Publish
Bind
Service Registry
API Documents / Service contract
RubyGems
Find
Service Clients
Leefplezier.nu
Content Management System (RefineryCMS)
Application content
Leefplezier App
Content Management System (Symphony2)
Application content

Figure 2. General overview of the Leefplezier architecture

IV. Comparison

While both applications described in Section III are used for collecting mental health data, their architectures are dissimilar, for two reasons.

The first reason is the fact that the applications were built in succession. We developed HND prior to developing Leefplezier. The experience of building and maintaining HND has influenced our decisions in designing the Leefplezier architecture.

The second reason is that the operating environments for the two applications were different. HND focused on sampling a larger percentage of the Dutch population. In order to find a large group of participants for HND, seeking publicity for HND was done using several newspaper articles, magazine articles, radio interviews, and other media attention [9]. When media attention was sought to announce new features of the application, this imposed strict deadlines on the development of HND and could cause a sudden increase in the number of participants. The important requirements for HND were therefore speed of development and scalability, often at the expense of maintainability and reusability. Leefplezier, on the other hand, is aimed at a smaller target audience, i.e., elderly people associated with certain care organizations. For Leefplezier, scalability is therefore not a primary concern. The lack of reusable code from the HND project forced us to create Leefplezier from scratch and to reevaluate the design priorities. For Leefplezier, two of the main requirements were established to be reusability and maintainability, by relying more on off-the-shelf components and existing services.

Table I shows the degree to which SOC concepts were applied in the case studies. *Custom* describes the components that were custom built for the target application. *SOC* describes the components used via external services. The remainder of this section explores, discusses, and compares the impact that the different requirements have on the architectural design of the applications.

Table I
The degree of SOC applied in the case studies

Topic	**HowNutsAreTheDutch**	**Leefplezier**
Authentication	Custom	SOC
Data management	Custom	SOC
Questionnaires	Custom	SOC
Diary questionnaires	SOC	SOC
Content management	Custom	CMS
Result calculation	SOC	SOC
Result visualization	Custom	SOC
Newsletter list management	Custom	SOC

A. Data security

E-mental health platforms may contain delicate and personal information that should not be publicly available. Keeping this information safe is a primary requirement for any e-mental health platform. An important factor for achieving data security is the data storage location. One can choose to store the data locally or outsource storage to an external service. In the latter case, one could opt for choosing an external service that is specialized in storing medical data.

In HND, the questionnaire data is stored locally. However, to reduce the impact of a security breach, the personal information (i.e., demographical data, email addresses, and authentication details) is stored separately from the questionnaire data. For both types of data a different database management system is used and the data is stored on physically different servers.

In Leefplezier too is the questionnaire data is stored separately from the personal information. Moreover, Leefplezier uses a specialized medical service known as RoQua[3] to store its data. RoQua offers different services for storing personal data and for storing anonymous questionnaire data. Authentication to Leefplezier is provided by a single sign-on service. By using single sign-on, Leefplezier is not responsible for securely storing the login information.

[3]Website: http://roqua.nl

B. *Conducting questionnaires*

Mental health research is predominantly reliant on the use of vetted questionnaires. These questionnaires are used to measure certain psychological constructs. When conducted in a larger sample, such questionnaires can give an indication of the mental health quality in that sample. In HND, we used an existing application for administering questionnaires, named *Quby*[4]. Quby is an application designed for administering questionnaires in the field of mental health. Currently, the Quby application only supports the mode where we are responsible for hosting the data. Hosting the data in a self-managed database did give us more flexibility since we had direct access to the data. The trade-off for this flexibility is having to assume responsibility for securely storing the questionnaire data.

The Leefplezier platform uses Quby as a Service. The Quby used in Leefplezier is hosted by an organization specialized in conducting medical questionnaires viz. RoQua. RoQua offers a platform that hosts Quby (also called RoQua). RoQua was used for all interactions with the hosted Quby, by means of REST web services. These web services were exposed natively through a Gem.

Although the service-oriented approach of Leefplezier for conducting questionnaires absolves us from the responsibility of having to worry about storing questionnaire data, it is not without caveats. The main issues with the Leefplezier approach are development time and flexibility. Implementing questionnaire functionality using an external web service could save time. However, since the available services were created to be very generic, they might not exactly provide the functionality needed by the platform to be implemented, and some additions might be required in order for the system to be usable. This was the case for the Leefplezier application, and these additions added up to a significant part of the development time spent. One instance of such an addition was fixing the order in which questionnaires were filled out. In the design of the Leefplezier study, it was determined that questionnaires had to be filled out in a certain order. While our external questionnaire service did provide functionality to conduct questionnaires in a certain order, the specifics of their approach did not match the requirements of the Leefplezier study exactly.

C. *Feedback generation*

The two case studies attempt to incentivize participants to fill out questionnaires by providing them with individual and useful feedback based on their questionnaire data. In particular, the diary study allows for detailed analysis of the changes in behavior of a person over time.

The feedback for the cross-sectional study phases is relatively straightforward. We show the participants a graph of their score compared to the maximum obtainable score and compared to the score of the average of the other people in the study. These graphs are calculated on the fly as they do not require lengthy statistical analysis.

[4]Source available at https://github.com/roqua/quby_engine

In contrast, the diary study feedback is more complex. We perform time series analysis using vector autoregressive (VAR) models to elucidate causal relationships between symptoms measured in the questionnaires. In order to fit a VAR model automatically, we use *Autovar* [11]. Autovar is written in the statistical programming language R, so its code cannot execute directly on the Ruby platforms. We therefore interface with Autovar over a RESTful interface, using the OpenCPU platform [18]. The OpenCPU approach is used for both the HND and Leefplezier applications and provides the added benefit of separating the statistical calculations from the application logic in a way that facilitates the reusability and scalability of the web applications.

Currently, the implementation of the Autovar service is not generic. Part of the application logic of HND and Leefplezier resides in custom functionality that was added to Autovar. A better solution would be for this functionality to be scriptable or configurable in some way (e.g., as parameters for the HTTP request calling the service), rather than to be coded into the Autovar package. Removing this specific application logic allows for easier reuse of Autovar in different projects.

D. *Feedback visualization*

Various graph types are used to visualize the questionnaire results (e.g., bar graphs, line graphs, scatter plots, pie charts, etc.). The graphs showing this feedback are generated using the *Highcharts*[5] and *Data-Driven Documents* (D3)[6] libraries. These are JavaScript libraries for rendering graphics. Both libraries are designed to run in the web-browser of the client.

HND uses both the Highcharts and the D3 library to visualize the feedback graphics. Since the only way to access HND is by using the web interface, these front-end JavaScript can be implemented directly into the application.

For the Leefplezier application, however, it is insufficient to only use client-side rendering. In Leefplezier, participants should also be able to view their feedback on a mobile application. For this feedback to be generated in a generic and abstract way, we developed a service provider called *Fizzy*. We designed Fizzy to be a reusable service that can generate graphs based on questionnaire data. Fizzy renders data using both Highcharts and D3 visualizations and exposes them as a Scalable Vector Graphic (SVG) via a RESTful API. By using graphics as a service, the Fizzy rendering capabilities can be used in multiple projects as they are separated from the rest of the application.

E. *Content management*

Not all contents of a web application are static. Some pages may require regular additions, removals, or other updates of text or media. Rather than editing the HTML files on the server by hand, applications frequently facilitate editing their contents as an integral part of the functionality of the web application. This approach is called a *Content Management System* (CMS). A CMS is designed to be easy to use for people without a

[5]Website: http://highcharts.com
[6]Website: http://d3js.org/

technical background. The main advantage of using a CMS is the fact that multiple users are able to edit the contents of the application as it is running. Without a CMS, any changes in the contents of the application falls to the responsibility of the application developers. The main disadvantage of using a CMS is that it incurs a higher up-front cost from the development team as the application needs to be adapted to integrate the CMS. An important requirement for a CMS to be useful is the dynamic nature of the content on the page. If the content is expect to change often, a CMS would be useful. If the content is likely to be static, a CMS might be less advantageous. Nevertheless, in most situations the application developers do not remain available for the length of the project to maintain the system for minor textual changes. In these cases a CMS is inevitable.

HND was implemented without the use of a CMS. All changes that were to be made to the application were given to the development team, who directly edit the HTML source of the pages. The decision for not having a CMS was made during the development phase, in order to save time on development and have high flexibility. In our experience, having a CMS, at least for the most frequently changing pages of the website, is worth the extra up-front cost. It would have been more time efficient if HND was implemented with a CMS, compared to the current approach.

The Leefplezier does integrate a CMS. The application uses *RefineryCMS*[7]. Our experience with having to maintain the HND website and being tasked with adding content throughout the year influenced our opinion in that it is good practice not only to separate contents from design but also to allow those contents to be updated directly by the users that create that content.

V. Requirements of e-mental health applications

From our two case studies, we identified not only architectural commonalities but also three main non-functional requirements that we believe are applicable to e-mental health applications in general: (i) *security and privacy*, (ii) *maintainability*, and (iii) *availability and reliability*.

1) Security and Privacy: Security and privacy are two requirements inextricably tied to any application that deals with personal health information. Law and regulations dictate the use of numerous security standards for collecting and storing medical information (e.g.,*NEN7510* in the Netherlands and *HIPAA* in the US). The delicate nature of health information can have major effects to a person when publicly accessible, and should be kept safe and secure. SOC can help achieving a secure an privacy aware system, by restricting data access for service clients. The service provider could be the only service retaining the data, being a single point of contact (SPoC) for data retrieval. The provider can be used as a security authority and restrict access to the data by only exposing specific services to specific service clients [19].

[7]Website: http://refinerycms.com

The separation of concerns in SOC also makes a system inherently more secure. A separation of for example personal data from patient health data reduces the chance of compromising all data, if one of the service providers were to contain a security breach. This does, however, not necessarily mean that using a SOC approach is the only valid solution. Some organizations have policies stating that data needs to be stored in-house. Such policies make the use of generic SOC solutions impossible. It is important to take different solutions into account, and select the solution most suitable for the organization.

2) Maintainability: Researchers in e-mental health projects are mostly concerned with starting the data collection as soon as possible and would prefer to spend little time and effort in having to develop and maintain the application itself. The concerns of these researchers can be met by combining off-the-shelf components rather than directing a one-off monolithic solution for which they are solely and wholly responsible. This workflow is accelerated by using SOC because it can reduce the lead time for developing a new project while outsourcing most of the software maintenance.

The separation of concerns imposed by SOC also increases maintainability. For example, by separating the concerns of application logic from those of the statistical data analysis, software developers and researchers can adapt the parts of the project in their field of expertise. Moreover, because of the dynamic binding between the SOC components, external services can be updated without needing to redeploy the application.

Due to the dynamic nature of SOC, service providers should devise clear and concise service contracts. These contracts can state which functions are exposed, and which parameters are expected by the service provider. The service provider should take this contract into account and notify clients when the specification of the contract changes.

3) Availability and Reliability: The punctual and time-dependent nature of many medical studies imposes high demands on system availability and reliability. These demands are magnified when the system features a patient-facing front-end. Especially in studies where patient compliance is an area of concern, additional technical difficulties that may discourage participation can be detrimental or even prohibitive to performing a study successfully.

Availability concerns can be mitigated by SOC since separation of concerns can help increasing the availability of critical parts of the system. These parts can be replicated decreasing the chance of a failure to occur.

VI. Proposed Architecture

Based on the analysis of the two case studies, we propose a generic architecture for e-mental health applications, shown in Figure 3. Based on the lessons learned from both our case studies, this architecture encompasses the elements common to applications in e-mental health research.

The architecture depicted in Figure 3 consists of five layers. From top to bottom, the layers decrease in volatility and in

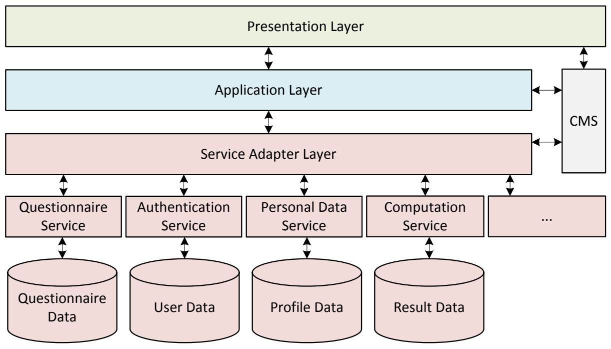

Figure 3. A layered overview of a generic architecture for e-mental health applications. The colors indicate the likelihood of an individual application having to write application-specific code in the layer (green is very likely, blue is likely, and red is very unlikely).

interactivity with the user. For example, while the presentation layer consists of mostly user-facing code, the lower layers contain the back-end of the application (hence the decrease in user interactivity). The decrease in volatility is because the top layers are deemed application specific, with layouts and graphical user interfaces that may vary from project to project, while the lower layers contain functionality that is more generic, more complex, unlikely to change frequently, and often accessed through service-oriented mechanisms.

The topmost layer is the *presentation layer*. The presentation layer houses the user-interface (e.g., layout templates and themes) of the application. We envision this layer to be the most application specific and to feature little code reuse, as each application is likely to want to use its own theme and looks. Figure 3 shows a connection between this layer and the CMS, as certain assets may be modified through a CMS, for example.

The second layer is the *application layer*. The application layer features the application logic as well as other code that lets the presentation layer interact with the other layers. Examples of application-specific components may include, e.g., the specifications for which pages are accessible to users, the diary study protocol specifications, and the specifications for which plots to include on a results page. Examples of reusable components in this layer may include, e.g., code to generate certain result plot types or account management pages.

The *service adapter layer* serves to abstract the service-specific code for interfacing with the back-end services from the application. This layer functions as an adapter in the sense that it provides the application with a non-changing interface to interchangeable external services components. This layer may host service-specific code to support certain services or service types should be restricted to this layer and should only need to change when the external components change, not when the application changes. We envision a scenario where these adapter definitions are managed by a service registry.

The fourth layer is the *service layer*, which contains the service providers. We have identifier four primary services as the common subset of services used by applications in e-mental health: (i) questionnaire services, (ii) authentication services, (iii) personal data services, and (iv) computation services. These services should operate fully agnostic of the application and are therefore often external to the application. The additional flexibility provided by the service adapter layer warrants that external services may be used as interchangeable parts, in such a way that, e.g., one can switch to a different questionnaire manager without having to rewrite any application code (for this particular example, migrating questionnaire data is a separate issue but an issue nonetheless).

The fifth layer is the *data layer*, which contains the data collected by the research application. We have two recommendations for this layer. First, we recommend (physically) separating user identifiable information (i.e., user account information and demographics) from the other data, to the best extent possible. The reason is that a data leak is more likely to happen in either one of these services than to both at the same time. Second, the process of selecting external services should weigh the availability of features for exporting data that meet the research requirements.

A separate component in the architecture is the CMS. Our recommendation is to take into account which contents of the application are likely to change, and if there are such contents, to provide a means for regular users to manage this content themselves. In practice, this often translates to integrating a CMS. Because, in theory, any form of an admin interface could also be used to manage, e.g., which questionnaires are selected for a study or which external services should be used, the CMS layer may have tie-ins to the application layer and service adapter layer. With the CMS being such an integral part of the application, it is often difficult to abstract the CMS from the rest of the application in a way that facilitates, e.g., interchangeability.

The presented architecture acknowledges the requirements listed in Section V. A short explanation for each requirement is listed below.

- **Security and Privacy:** The generic architecture absolves the application owner from being solely responsible for data security by assigning responsibility to each of the external services for managing their share of the data. Security could further be increased by using data storage services from providers that specialize in information security.
- **Maintainability:** Applications following the proposed architectural guidelines are maintainable because they (i) separate application-specific code from reusable code (e.g., the layout is separated from the back-end logic), (ii) separate code by functionality and responsibility (e.g., only the computation service is responsible for statistical computations, and for nothing else), and (iii) separate static content from dynamic content, allowing direct user edits for the latter category (i.e., by integrating a CMS).
- **Availability and Reliability:** Although our architecture does not directly influence the availability and reliability of the platform, it was designed to allow for redundant

availability between any of its individual components, providing a high level of reliability.

VII. Evaluation

One of the major advantages of our proposed architecture is the loose coupling of the service components, provided by SOC. Loose coupling allows for easy reuse of the services when implemented in a generic way. In our proposed architecture, the service adapter layer enforces the loose coupling between the application and the services it uses. Reuse of existing systems can save development time for successive studies and can eventually save costs. For medical data, one advantage of SOC could be to outsource the responsibility for keeping data secure.

The proposed architecture also has some important limitations to keep in mind. While applying SOC removes several levels of complexity, it also introduces new ones. One difficulty with SOC is the testability of the applications. For testing applications that use SOC, traditional approaches may be less suitable [20]. This may result in applications being unable to validate the functionality of external services, having to trust that their descriptions are accurate and up-to-date. Even in cases where testing external services is theoretically possible, this may be too slow and costly for practical use, especially for services that charge on a per-request basis.

Relying on external services for accessing data may have other downsides. The increase in security and maintainability comes with a decrease in flexibility and performance. Flexibility is reduced since the service client has no direct influence on the functionality exposed by the service provider. In many cases, the service client might not have access to run detailed queries on the data. Performance is reduced due to the extra layer of abstraction (i.e., the API of the service provider) between data and application. Data read requests that invoke an external service induce latency and bandwidth overhead compared to direct database connections. Especially in procedures that necessarily invoke multiple repeated queries, the performance penalty can be significant.

Finally, there are several aspects to keep in mind when implementing any kind of SOA. Firstly we noticed it is important to have a clear service contract with the service provider. The service provider should define which services they provide and which parameters they expect for these services. The service provider should stick to this contract and notify the service clients whenever an important change is introduced. Secondly, a service client depends on the availability of the service provider. Whenever a service provider is unavailable, the service client could suffer from this. In order to ensure the availability of the external services, one could establish service level agreements with the service provider, describing their availability.

VIII. Conclusion

The purpose of the present work was to devise a generic architecture for e-mental health research applications. In order to determine such architecture, we evaluated two e-mental health applications that we developed in recent years. Both applications have been implemented successfully and used on a moderate to large scale. We explored the differences and similarities between the applications and provided heuristics for implementing these architectures in practice. We proposed a generic architecture for an e-mental health research application and evaluated its merits and disadvantages.

Despite the fact that we performed only two case studies, we believe that the presented architecture and the described caveats for implementing SOC architectures can serve as guidance for developing e-mental health research applications in general. Since e-mental health applications frequently have a significant amount of overlap in terms of their main functionality (i.e., recording questionnaires and providing feedback), using a service-oriented architecture fosters the reuse of these components and meets the most important requirements of e-mental health platforms. The use of specialized services can help application owners to focus on performing research without being burdened with the implementation details of every single aspect of the application. If more applications followed these guidelines, then perhaps in the future the interfaces could be standardized, and components could be reused not only between projects within the same hospital, but between different hospitals worldwide.

Acknowledgements

The Leefplezier project is funded by Espria. Espria is a healthcare group in the Netherlands consisting of multiple companies targeted mainly at the elderly population. The HowNutsAreTheDutch project is partly funded by a VICI grant (no. 91812607), awarded to Prof. Peter de Jonge. We would like to thank all people working on the HowNutsAreTheDutch and the Leefplezier project. We would also like to thank Prof. Marco Aiello and Prof. Peter the Jonge for their supervision, and RoQua for allowing us to use their services.

References

[1] R. V. Bijl, A. Ravelli, and G. Van Zessen, "Prevalence of psychiatric disorder in the general population: Results of the Netherlands Mental Health Survey and Incidence Study (NEMESIS)," *Social Psychiatry and Psychiatric Epidemiology*, vol. 33, no. 12, pp. 587–595, 1998.

[2] K. M. Griffiths and H. Christensen, "Review of randomised controlled trials of Internet interventions for mental disorders and related conditions," *Clinical Psychologist*, vol. 10, no. 1, pp. 16–29, 2006.

[3] T. Donker, K. Petrie, J. Proudfoot, J. Clarke, M. R. Birch, and H. Christensen, "Smartphones for smarter delivery of mental health programs: A systematic review," 2013.

[4] R. Kazi and R. Deters, "RESTful dissemination of healthcare data in mobile digital ecosystem," *IEEE International Conference on Digital Ecosystems and Technologies*, pp. 78–83, 2013.

[5] R. Kazi, "A Dissemination-Based Mobile Web Application Framework for Juvenile Ideopathic Arthritis Patients," pp. 990–997, 2013.

[6] M. Savini, A. Ionas, A. Meier, C. Pop, and H. Stormer, "The eSana Framework: Mobile Services in eHealth using SOA," in *2nd European Mobile Government Conference (Euro mGov 2006)*, 2006, pp. 191–200.

[7] L. Fan, W. Buchanan, C. Thümmler, O. Lo, a. Khedim, O. Uthmani, a. Lawson, and D. Bell, "DACAR platform for eHealth services cloud," *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, pp. 219–226, 2011.

[8] F. Blaauw, L. van der Krieke, E. Bos, A. Emerencia, B. F. Jeronimus, M. Schenk, S. de Vos, R. Wanders, K. Wardenaar, J. T. W. Wigman, M. Aiello, and P. de Jonge, "HowNutsAreTheDutch: Personalized feedback on a national scale," in *AAAI Fall Symposium on Expanding the Boundaries of Health Informatics Using AI (HIAI'14): Making Personalized and Participatory Medicine A Reality*, 2014.

[9] L. van der Krieke, B. Jeronimus, F. Blaauw, H. Schenk, R. Wanders, A. Emerencia, S. de Vos, E. Snippe, M. Wichers, J. Wigman, E. Bos, K. Wardenaar, and P. de Jonge, "HowNutsAreTheDutch (HoeGekIsNL): A Crowdsourcing Study Exploring Mental Symptoms and Strengths in the General Dutch Population," submitted.

[10] L. van der Krieke, F. J. Blaauw, A. C. Emerencia, M. H. Schenk, E. H. Bos, B. F. Jeronimus, and P. de Jonge, "Temporal dynamics of health and well-being in the Dutch population: mobile assessments and personalized feedback," submitted.

[11] A. Emerencia, L. Van der Krieke, E. Bos, P. De Jonge, N. Petkov, and M. Aiello, "Automating vector autoregression on electronic patient diary data," *IEEE Journal of Biomedical and Health Informatics*, 2015, in press.

[12] F. Blaauw, L. Van der Krieke, P. de Jonge, and M. Aiello, "Leefplezier: Personalized well-being," *IEEE Intelligent Informatics Bulletin*, vol. 15, no. 1, pp. 28–29, 2014.

[13] P. J. Clarke, V. W. Marshall, C. D. Ryff, and B. Wheaton, "Measuring Psychological Well-Being in the Canadian Study of Health and Aging," *International Psychogeriatrics*, vol. 13, no. S1, pp. 79–90, Feb. 2001.

[14] Office for National Statistics, *Social Trends (correction notice)*, 2010th ed., M. Hughes and J. Church, Eds. Newport: Palgrave MacMillan, 2010, no. 40.

[15] Centraal Bureau voor de Statistiek, "Langer leven, maar meer jaren met lichte lichamelijke beperkingen," http://www.cbs.nl/nl-NL/menu/themas/gezondheid-welzijn/publicaties/artikelen/archief/2014/2014-4135-wm.htm, 2014, (Accessed: 01 May 2015).

[16] C. Victor, S. Scambler, J. Bond, and A. Bowling, "Being alone in later life: loneliness, social isolation and living alone," *Clinical Gerontology*, vol. 10, no. 4, pp. 407–417, 2000.

[17] C. R. Victor, S. J. Scambler, A. Bowling, and J. Bond, "The prevalence of, and risk factors for, loneliness in later life: a survey of older people in Great Britain," *Ageing and Society*, vol. 25, no. 3, pp. 357–375, 2005.

[18] J. Ooms, "The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns," no. 2000, pp. 1–23, 2014.

[19] L. Fan, W. J. Buchanan, O. Lo, C. Thuemmler, A. Lawson, O. Uthmani, E. Ekonomou, and A. S. Khedim, "SPoC: Protecting Patient Privacy for e-Health Services in the Cloud," in *The International Conference on eHealth, Telemedicine, and Social Medicine, eTELEMED*, no. 4, 2012, pp. 98–104.

[20] G. Canfora and M. Di Penta, "Service-oriented architectures testing: A survey," in *Software Engineering*. Springer Berlin Heidelberg, 2009, pp. 78–105.