KasaDaka: a sustainable voice-service platform

Developing a Voice Service Development Kit

André Baart
VU Amsterdam
andre@andrebaart.nl
Master's thesis supervisor: Victor de Boer

ABSTRACT

In the developing world the adoption rate of the Internet is still relatively low, causing a *digital divide*. Barriers to adoption include a lack of relevant content, low literacy and a lack of resources and infrastructure. Innovative voice-services that can be accessed through the highly adopted simple mobile phones of the local population, are able to offer information services like those found online. This research focuses on lowering the barrier to entry of voice-service development, by simplifying the development process and reducing the knowledge required. This allows the sourcing of local voice-service developers, which improves sustainability by eliminating the dependency on foreign (expensive) developers. A Voice Service Development Kit (VSDK) is developed, which allows web-based voice-service development by applying and customizing voice interaction *building-blocks*. The VSDK has been evaluated by students of the ICT4D[1] course at the Vrije Universiteit Amsterdam, who have developed applications for several ICT4D use-cases using the VSDK. From the evaluation is concluded that the *building-block* approach to voice-service development is successful in reducing the complexity and allows inexperienced users to develop simple voice-services in a short time.

Keywords

voice-based services, sub-Saharan Africa, ICT4D, slot-and-filler, low-resource hardware, digital divide, low literacy, development frameworks

1. INTRODUCTION

In sub-Saharan Africa the usage of modern technologies such as the Internet is still very low compared to highly-developed countries. Mobile phones (so-called 'dumbphones') however are widely used, even in the rural areas. Previous research has shown the feasibility and possible applications for voice-based services offered over the mobile telephone network, in the rural sub-Saharan African context. (de Boer et al., 2012) Recent work has made it possible to host voice-services on low-resource hardware (e.g. a Raspberry Pi computer), which can be owned by local (farmer) communities. (Baart, 2016b)

This research will look into how the process of voice-service development and maintenance can be made less complex, with the end-goal of improving the (financial) sustainability of voice-services in sub-Saharan Africa. This document describes the process and results of the building of a so-called *Voice-Service Development Kit* (based on the existing KasaDaka platform), which simplifies large parts of the development and maintenance processes of voice-services.

[1]Information and Communications Technology for Development

1.1 Context

Nowadays in highly developed countries, a connection to the Internet is available nearly everywhere and a large part of the population is always online. The Internet has become the main source of information and enables new ways of working and communicating. Information and Communication Technologies (ICT) have revolutionized the way of living and doing business, placing these countries into the era of the *Information Society*. (Webster, 2014) The economies of these countries have become very dependent on ICTs.

In so called 'developing countries' the situation could not be more different. The Internet penetration rate in Africa (in 2016 25%) is far below that of developed regions such as Europe. (79% ,world average 47%) (ITU, 2016) In areas such as sub-Saharan Africa, a stable electricity supply can be hard to come by, let alone a stable connection to the Internet. Issues including poor infrastructure, lack of resources, poor education (low literacy), a lack of relevant content and a lack of experience in the usage of modern technologies, prevent the large scale adoption of ICT in this context. (Chapman et al., 2002)

A gap in wealth and quality of life exists between people that have access to ICTs, and those that do not. This *digital divide* (Fuchs et al., 2008) is especially relevant in the rural communities. Urban areas will usually see a faster adoption of new technologies, as the necessary infrastructure is usually constructed in these areas first. This causes an increase in the inequality between urban and rural areas, in addition to the already existing *digital divide*.

1.1.1 ICT4D

The field of ICT for Development (ICT4D) aims to close the digital divide in the developing world. By applying the opportunities of computers, the Internet and mobile phones in the context of development. The goal is to use these technologies to increase well-being and reduce the problem of global poverty. The promoting of usage of 'traditional' ICTs (such as computers and the Internet) was the initial focus of ICT4D. These efforts were not very successful, as they turned out to be unsustainable in the developing world context. While these technologies thrive in highly developed countries, they do not necessarily satisfy the needs of a developing country (Toyama, 2010). The focus of ICT4D has since largely shifted to the usage and implementation

of technologies that are already available and have a large user base in the developing world. (Heeks, 2008) This new approach is referred to as ICT4D 2.0.

Recently an approach to ICT4D is suggested (that is to be called ICT4D 3.0), that focuses primarily on the needs of the users. This approach takes the user's goals and objectives as the starting point of a project and collaborates closely with the end-users during the co-creation process of an information service. The technologies that are used are fully adapted to the local context and use existing local infrastructure. (Bon, 2016)

1.2 Motivation

One of the technologies that has become widely adopted in Africa is the mobile phone. Mobile phones have become a successful means of communication in sub-Saharan Africa, having become the main means of telecommunication. The coverage of networks in rural Africa is good, covering a large part of the population. "mobile phone subscriptions on the continent have risen from over 16 million in 2000 to 376 million in 2008 –or one third of sub-Saharan Africa's population". (Aker et al., 2010) The subscriber penetration rate in the ECOWAS[2] states in 2015 was 47%, with a projected growth to 55% in 2020. (GSMA, 2016)

This widespread adoption creates many opportunities for the application of voice services. The network coverage of remote areas offers opportunities for rural communities and a possibility of (partially) closing the *digital divide*. The usage of existing (and adopted) technologies for new use-cases fits well within the scope of ICT4D 2.0.

Past research has proven the feasibility of the deployment of *web-like* voice-services in the context of sub-Saharan Africa. The KasaDaka platform[3] was created, allowing for locally hosted voice-services over the local available mobile telephony network. Voice-services cater exceptionally well to this context, as they are usable by the illiterate and they can be accessed by phones already owned by a large amount of the population.

Voice services for the KasaDaka can be developed in little time, allowing for a *rapid-prototyping* workflow in which a voice-service prototype can be developed in a very small time (less than an hour). (Baart, 2016b) Use-cases include a market platform, local journalism platform and a livestock vaccination scheduling system. Over the past years, several small scale experiments have been done with these use-cases (hosted on the KasaDaka) in sub-Saharan Africa. (Gyan et al., 2013; Chhetri, 2013; Mantel, 2014; Gyan, 2016)

1.2.1 Regreening

The KasaDaka project builds on information and experience acquired by the Web alliance for Regreening in Africa[4] (W4RA). The goal of the W4RA is to support farmer-managed regreening activities by developing and deploying innovative ICT applications in order to deliver relevant services to rural communities in sub-Saharan Africa.

Regreening is a process where farmers manage naturally regenerating trees on their land, instead of cutting them down. These trees help in restoring degraded lands, providing many benefits such as increased crop yields and recharging groundwater. These effects improve food security and help to lessen the influence of climate change on agriculture. (Sendzimir et al., 2011)

Figure 1: Ousséni Kindo (farmer in Burkina Faso) showing his land and explaining the agricultural techniques he practices. Photo credit: Anna Bon

The practice of regreening techniques is mainly centered in the Sahel region, which is the southern border area of the Sahara desert. Regreening allows for land that was unsuitable for agriculture due to desertification (mainly lack of ingress of rainwater into the soil) to become fertile again, increasing the area of available land for agriculture. (Reij et al., 2009)

Regreening is a very relevant practice in sub-Saharan Africa, as the expected population growth in the region (260% projected growth) will cause a rapid rise in demand for food. This demand will rise even more because of the changing of lifestyles and wealth growth in the area. In order to match this rising demand, rapid growth of crop yield is required as well as a rapid increase in the amount of land used for agriculture. (Ittersum et al., 2016)

W4RA partners with local farmers and organizations that promote regreening in rural communities. The goal is to improve local agriculture by providing relevant information services to rural farmers. This improves opportunities and living conditions of local farmers, increasing their competitiveness in the market and promoting sustainable agriculture growth.

The KasaDaka platform enables rural farmers to use locally hosted voice-services. This is achieved by hosting the services on low-resource hardware, such as a Raspberry Pi with a GSM dongle for connection to the mobile telephony network. By using this relatively inexpensive hardware and free/open-source software the platform is 'affordable' for rural sub-Saharan farmers. This enables them to use independent and community hosted information services over their already owned mobile phone, helping them in their agricultural activities and allowing them to disseminate their innovative regreening techniques.

1.3 Problem description

In order to enable further research of (and eventually permanently roll out) locally hosted voice-services in sub-

[2]Economic Community of West African States: http://www.ecowas.int/

[3]http://kasadaka.com/

[4]http://w4ra.org/w4ra/

Saharan Africa, a long term deployment (pilot) is required. There have been previous long term deployments of voice-services by the W4RA, however these were on an older platform[5] which requires significant infrastructure and is no longer maintained. (de Boer et al., 2012; Gyan et al., 2013; Boer et al., 2015) This new deployment can become the basis for further research on voice services and the KasaDaka platform. A small-scale but long-term experiment is also one of the first steps towards a future roll out on a larger scale.

While previous work has made the KasaDaka into a *rapid-prototyping platform*, enabling voice-service prototypes to be created in a short time frame, there is still significant knowledge and skills required to do so. A deployed voice-service is currently not maintainable by the (local) owner/maintainer of the KasaDaka (assumed to be farmer NGOs in rural sub-Saharan Africa). These local partners (NGOs) have limited computer knowledge, let alone programming skills. Remote management is hard to implement, as reliable connections are hard to come by and expensive. This implies that whenever there are problems, there is a very strong dependency on the team that maintains the software (currently located at the Vrije Universiteit Amsterdam). Realistically this means that whenever there is a problem the voice services are unusable for some time, until a specialist is flown in to fix the problem. As long-distance communication on these specialized subjects is often difficult, it usually takes a long time for the problem to be resolved. This makes a long-term roll-out expensive and unrealistic until these problems are addressed.

1.3.1 Sustainability

In previous research concerning the RadioMarché project of Gyan (2016), the conclusion has been drawn that a single model on the sustainability of voice-services is difficult to define. Many of the factors are external to the platform itself, such as ensuring ample funding to pay for the hosting of the service and the amount of value provided by the service.

However, some measures to improve sustainability and transferability were taken in the RadioMarché project. Local ownership is seen as an important aspect, as it removes the dependence on the support of (commercial) companies and resources that might not be stable in the long term. Physical ownership is only half of 'actual ownership' in this regard, as a dependency on (foreign) corporations limits the freedoms of the local community to the vision of these corporations, which often do not correspond to the needs and desires of the local population. It is important that the local community is able to host and maintain the voice services, as well as be able to develop new voice services independently.

Gyan mentions local training initiatives to be a possible solution to this problem and the VOICES[6] (VOice-based Community-cEntric mobile Services) project has had experiences with setting up *mobile training labs* with the goal of sparking local tech-entrepreneurship. (Papeschi et al., 2011) The general idea is to create a local community of enthusiasts, functioning as a technical development and maintenance team for voice-services. Training sessions should be organized in order to provide the community with the required technical knowledge for voice-service development. By allowing the local population to develop applications on the KasaDaka platform and giving them total freedom to use and modify the system to fit their community's needs, voice-service technology will inevitably be used in many new and creative ways. Ali et al. (2007) argue that this improvised use of and tinkering with existing and available technology, cause unintended applications that often become very successful, an activity which is referred to as *bricolage*. Accepting that the usage of technology cannot be directed and that successful innovations often come from unexpected directions, can be a determining factor of success in the field of ICT4D. By explicitly granting the general population the freedom to use the technology in any way they see fit, practicing *bricolage* is facilitated and the available technology is likely to (eventually) be applied in a way that is most relevant and innovative to the local context.

The usage of open standards allows for the usage of open-data in voice-services, which can bring the advantages of open-data to the poor, who are often excluded from the benefits of open-data. The other way around, data generated from voice-services can be used to improve local policy and other development efforts. (Davies et al., 2012)

1.3.2 Towards a more sustainable voice-services platform

In conclusion: in order to make the KasaDaka into a platform that can be truly *owned* by the local population, it is of importance that a local community can emerge that is able to develop and maintain voice-services. This problem should be approached in multiple ways. This research will focus on one approach, which is to lower the skill-set needed to develop voice-services on the platform. The goal of this strategy is to reduce the required prerequisite knowledge for the voice-service development process. This will in turn lower the barrier of voice-service development, enabling a larger amount of people to develop voice-services for the KasaDaka platform.

If the KasaDaka platform succeeds in this, an international movement could emerge that develops voice-services that solve information needs of rural communities around the world. By not focusing on the development of services that address a limited set of use-cases, but instead offering a platform that enables the development and hosting of voice-service for a low investment; The KasaDaka platform could contribute to the reduction of poverty and improve the living conditions of the global poor.

2. RELATED WORK

This section covers existing efforts in ICT4D and the design and development of voice-services. The voice-service development solutions described allow the development of voice-services through a Graphical User Interface (GUI), which is a logical step towards reducing the complexity of application development. We will mostly focus on applications that help in the development of VoiceXML[7] (a XML based language for audio dialogs, see Section 5.2.2) based voice-services. This choice is made because the goal is that the resulting system is to be integrated into the existing KasaDaka platform,which is mostly based on existing open-source software and standards, in which VoiceXML plays a

[5]This was the Emerginov platform by Orange. https://emerginov.ow2.org/

[6]http://mvoices.eu/

[7]https://www.w3.org/TR/voicexml20/

large role.

2.1 KasaDaka voice services platform

Previous work on the KasaDaka platform has enabled rapid development of voice services. The KasaDaka platform consists of a Raspberry Pi combined with a GSM dongle. The dongle provides a voice connection to the local GSM/3G network, connecting the calls to the software running on the Raspberry Pi. Many of the software components used are open-source, making the platform easily extensible and less reliant on commercial offerings. (Baart, 2016b)

While rapid development of voice services is possible, extensive knowledge of VoiceXML, Asterisk[8] (an open-source telephony exchange) and Python[9] is required. Currently there is no GUI-based application for the development of voice services for the platform and the generating of VoiceXML is done with Flask[10], a Python web-framework. Developing voice-applications for the KasaDaka platform thus requires the developer to have knowledge and experience in several standards and services including: Python, VoiceXML, Asterisk and the usage of POSIX-based operating systems[11]. This severely limits the accessibility of voice service development of sub-Saharan locals, extensive training and education would be required to start a local voice-service development community.

2.2 Sugar Desktop Environment

Sugar[12] is an open-source graphical environment developed for the One Laptop Per Child[13] (OLPC) project. The goal of Sugar is to provide a simple and intuitive interface for school-going children, with the aim of improving the education in developing countries. The interface of Sugar mainly consists of icons and graphical interfaces, with text based-interfaces offered in more advanced applications, such as browsing the internet (if there is an internet connection) or writing a document. The OLPC/Sugar also included peer-to-peer networking for data exchange in situations where there is no internet connection. The development of applications for the Sugar environment uses Python, which makes the development of applications out of reach for the intended users of the OLPC. The OLPC project is an example of the ICT4D 1.0 mindset, in which large numbers of computers are shipped to developing areas with the expectation that the local population will educate itself on the usage of these computers. In reality this did not happen due to a lack of usage in education programs (educators did not receive training on the OLPC), hardware and infrastructure problems (lack of electricity), a lack of adoption (OLPC's were not affordable enough for developing countries) and a lack of usefulness in the local context. (Warschauer et al., 2010)

2.3 DataWinners

DataWinners[14] is a data collection platform that is developed by Human Network International[15] (HNI). DataWinners enables the development of SMS and smartphone-based data surveys. These surveys are primarily aimed at the context of NGOs that need to retrieve data from their extension workers. By using SMS data can be collected without a need for an Internet connection. In the DataWinners web-based environment, new data surveys can be created in a graphical interface. As DataWinners is based on the usage of SMS, it is not usable by the low literate population.

2.4 RapidSMS

RapidSMS[16] is a toolset that allows for the development of SMS-based services for data collection and other workflows. RapidSMS is developed by UNICEF and has been used for various use cases, including remote health diagnostics and nutrition surveillance. RapidSMS is open-source and very scalable to suit large deployments, but can also run on a low-end server with a GSM modem. The data collected through SMS can be published online automatically, and Internet-accessible information can be provided to the user through SMS.

2.5 Aspect CXP/Designer (Voxeo)

A commercial and well-known platform for voice-service development is made by Aspect[17] (former Voxeo) and is called the CXP[18] (Customer Experience Platform). A component of CXP is called the Developer, which has a graphical interface for designing voice services. (see Figure 4) The focus of the CXP software lies in multi-modal applications, i.e. designing an application for multiple channels such as: voice, SMS, video and the mobile web. Earlier versions of CXP designer were built on the Eclipse[19] IDE (Integrated Development Environment) platform.[20].

Another software by Aspect is the Designer, which "makes it easy for developers and non-developers alike to create and deploy simple commercial-grade IVR applications in a Visio-like environment"[21]. The designer generates VoiceXML files that can be used in an existing or Voxeo based hosting solution. Designer is integrated in the (commercial) Voxeo Prophecy platform. Screenshots of the Designer software are included in Figures 5 & 6.

The graphical interface offered by Aspect's products is an example of how the development of voice-services can be implemented in a graphical manner, by combining a graphical overview of the call-flow, combined with templates of interactions that can be used to build the service. A limitation for use in the ICT4D context, is the intensive usage of TTS and ASR technologies in the interactions offered. Both of these technologies are not available in the under-resourced languages that are spoken in sub-Sahara Africa. (see Section 4.1.1)

[8] http://www.asterisk.org/
[9] https://www.python.org/
[10] http://flask.pocoo.org/
[11] Portable Operating-System Interface: https://nl.wikipedia.org/wiki/POSIX
[12] http://wiki.laptop.org/go/Sugar
[13] http://one.laptop.org/
[14] https://www.datawinners.com/
[15] http://hni.org/
[16] https://www.rapidsms.org/
[17] https://www.aspect.com/
[18] https://www.aspect.com/nl/solutions/self-service/customer-experience-platform-cxp
[19] https://eclipse.org/
[20] https://marketplace.eclipse.org/content/voxeo-cxp-developer
[21] https://evolution.voxeo.com/docs/platforms.jsp
[22] Source:http://wiki.laptop.org/go/Sugar
[23] Source:http://hni.org/what-we-do/data-collection/datawinners/
[24] Source:http://help.voxeo.com/go/help/vo.cxp16.designguide.dialogflow

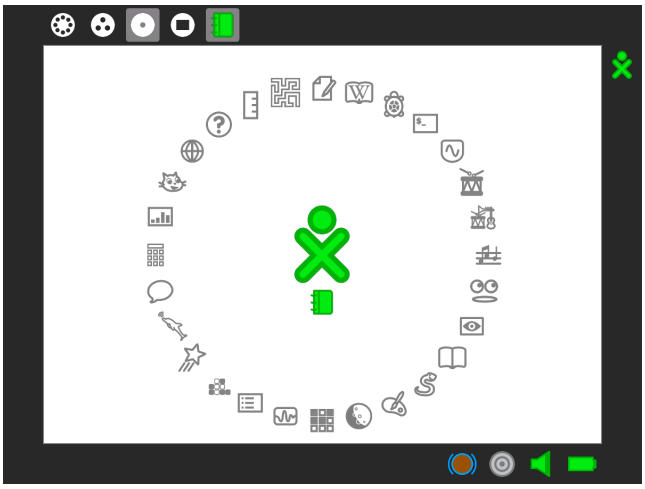

Figure 2: Screenshot of the Sugar desktop environment.[22]

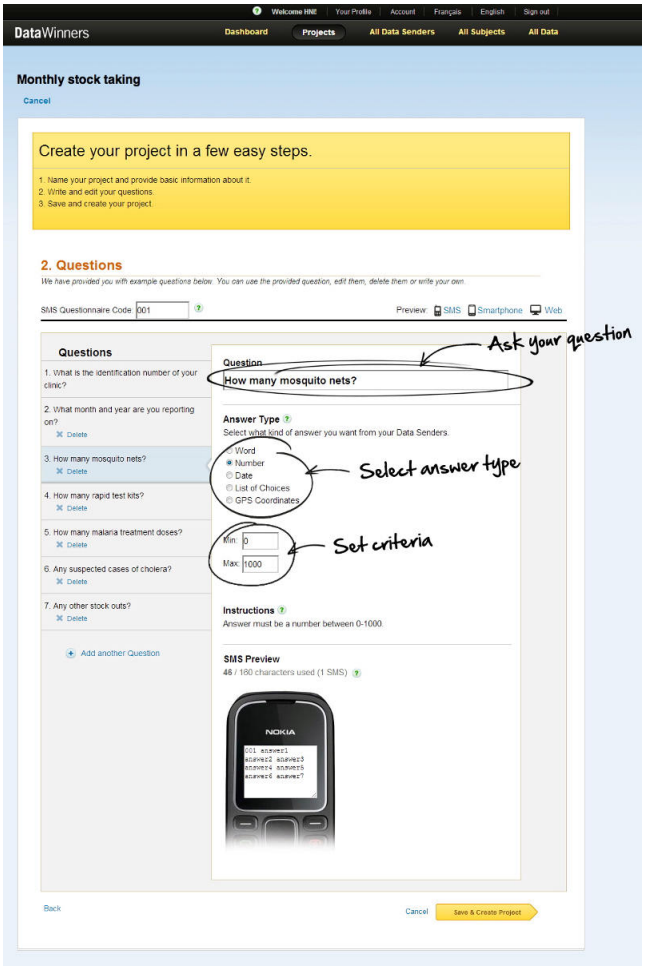

Figure 3: Screenshot of the DataWinners interface for the development of SMS-based data surveys.[23]

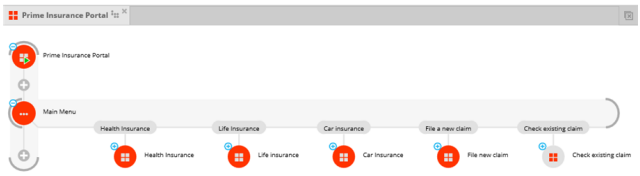

Figure 4: Screenshot of the Aspect CXP Developer interface for call flow development.[24]

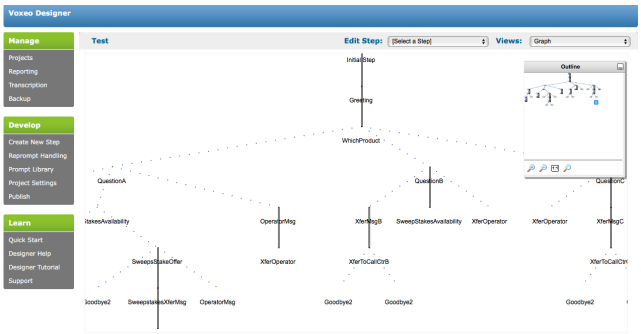

Figure 5: Screenshot of the Voxeo Designer interface call flow visualization.[25]

2.6 Blueworx toolkit

Another commercial VoiceXML development solution is the Blueworx toolkit[29] (formerly known as IBM WebSphere Voice Response[30]). The toolkit is (just like former versions of Voxeo CXP) built upon the Eclipse IDE platform and produces VoiceXML that is intended to be deployed on the WebSphere/Blueworx platform. The toolkit includes a visual call flow designing interface, in a different style than that of Voxeo/Aspect.

The graphical development interface offered by Blueworx bears a general resemblance to the interface offered by the Scratch[31] programming language. However changing the settings of each of the elements in the service still requires significant knowledge of the inner workings of the underlying mechanisms. Furthermore, as Blueworx is based on the Eclipse platform, installation of software on the developers computer is required and the developer needs to gain experience in working with Eclipse.

2.7 Twilio Studio

While Twilio Studio[32] does not use VoiceXML but Twilio's proprietary language TwiML[33], it does provide a easy to use interface for building voice-services. Twilio Studio allows

[25]Source: Voxeo Prophecy software, available for download from:https://www.aspect.com/voxeo
[26]Source: Voxeo Prophecy software, available for download from:https://www.aspect.com/voxeo
[27]Source: https://www.ibm.com/support/knowledgecenter/SSKNG6_6.1.0/com.ibm.wvraix.geninf.doc/i1671052.html
[28]Source: https://www.twilio.com/docs/api/studio
[29]http://www.blueworx.com/use-the-blueworx-toolkit/
[30]http://www.waterfieldtechnologies.com/wti-acquires-ibm-websphere-voice-response/
[31]https://scratch.mit.edu/
[32]https://www.twilio.com/docs/api/studio
[33]https://www.twilio.com/docs/api/twiml

Figure 6: Screenshot of the Voxeo Designer voice service functionality options.[26]

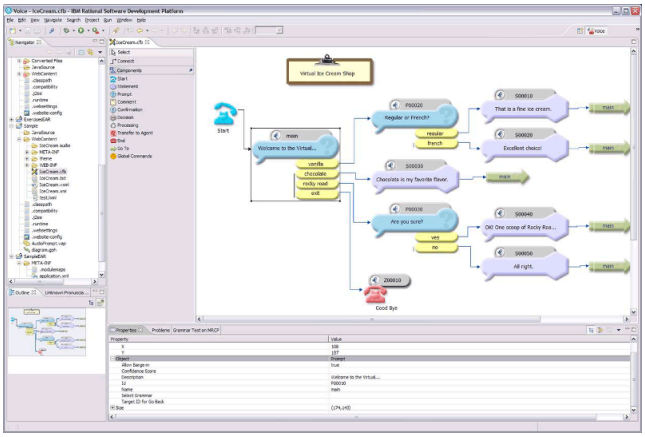

Figure 7: Screenshot of the IBM WebSphere Voice Response Communication Flow Builder (version 6.1).[27]

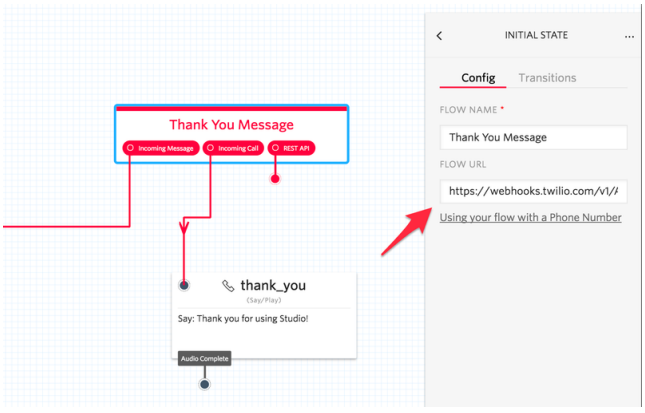

Figure 8: Screenshot of the interface of Twilio Studio.[28]

voice-service development by dragging *widgets* into the call flow, which are the components in a voice service. Through an inspector panel these widgets can be configured in more detail. Twilio Studio is entirely web-based and is hosted by Twilio.

The graphical development methodology offered by Twilio Studio seems to be very user-friendly and simple to use. However the usability of voice-services created in Twilio Studio is limited to the Twilio platform, which does not allow the purchase of phone numbers in many of the countries where ICT4D voice-services are relevant. Twilio Studio seems to not be usable without an Internet connection, which can not be assumed to be available in the ICT4D context. Furthermore, just like the previous examples Twilio makes intensive usage of TTS and ASR technologies, which are not available in the ICT4D context.

2.8 Suitability of related work in the ICT4D context

The conclusion that can be drawn from the discussed examples of voice-service development environments, is that there are serveral products that provide the functionality of visual voice-service development. However all of the examples are commercial products, which severely limits their usability in the ICT4D context. (see Section 4.1.2) While many of the examples use VoiceXML and thus produce platform independent voice-applications, actually integrating them in a voice-service hosting platform suitable for the ICT4D context (i.e. the KasaDaka platform) is no trivial task. Besides these constraints there are several other issues limit the usability in the ICT4D context, such as the use of TTS/ASR, requirement of Internet connection and limiting methods for data management.

The methodologies offered for the development of voice-services in the discussed examples succeed in reducing the complexity of voice-service development. While some of the implementations of graphical development are more advanced than others, the general methodology used in the environments is the building of voice-services by applying provided templates for interactions. These templates provide a basic interaction type, such as a choice, the playback of a message or requesting and recording user input. (see Figure 6) By applying these interaction templates as *building-blocks* for the development of a voice-service, it becomes possible to build a voice-service without interacting with the underlying (VoiceXML) code. This methodology of voice-service development is comparable to component-oriented software development. (Nierstrasz et al., 1992)

How the resulting system that was built in this research fits into the bigger picture of voice-service development environments, is discussed in Section 8.

3. HYPOTHESIS AND METHODOLOGY

The goal of this research is to develop methods that allow voice-service development by users that do not have experience in the underlying technologies. These methods have the goal of simplifying the development process, which reduces the barrier of entry into voice-service development. In the ICT4D context this simplification reduces the (knowledge) requirements of voice-service developers and maintainers, making it possible for sub-Saharan agents (e.g. NGO workers) to develop and maintain local voice-services. This significantly reduces the cost involved with voice-service de-

velopment and maintenance, as well as reducing the dependency on external support and foreign corporations. Both of these effects play a large role in improving the (financial) sustainability of voice-services in the ICT4D context of sub-Saharan Africa. The preferred approach to simplifying voice-service development and maintenance is by developing a graphical application for voice-service development, which facilitates the development of voice-services without requiring the user to write code. By providing the user with a set of possible interaction templates (which can be considered to be voice-service *building blocks*), the user can select, customize and deploy these templates to create a simple voice-application. This application will be referred to as the Voice-Service Development Kit, or **VSDK** in short.

These above defined goals lead to the following main hypothesis:

The interactions found in voice-based applications in the ICT4D context can be generalized to a small set of interaction types. By providing building-blocks for these interactions inexperienced users can build simple voice-applications[34] *by deploying and customizing these building-blocks.*

This hypothesis leads to the following research questions:

1. What are the requirements of a VSDK that enables the target user group to develop voice-applications? What does the sub-Saharan African ICT4D context contribute to these?
2. Which technologies are suitable to be used in the VSDK?
3. Does the resulting VSDK result in an improved ease of use in voice service development?

3.1 Methodology

3.1.1 Defining requirements

The first step is to analyze and gain insights on the context of ICT4D in sub-Saharan Africa and voice-services in particular. This insight is acquired by reading and analyzing relevant literature on this subject, as well as doing field trips to the area. During research projects of the W4RA the researcher has visited Burkina Faso and Mali. Due to the security situation in Mali, the visit to Mali did not include actual field visits but did include visits to local farmer and rural radio organizations. (Baart, 2016a) The visit to Burkina Faso included a workshop with local farmer-innovators, where the researcher developed a demo application in the local language. (Baart, 2017b)

The VOICES project (in which the W4RA participated) provided several trip reports and valuable experiences in the development and deployment of voice-services in the ICT4D and regreening contexts. (Gyan et al., 2013; Akkermans et al., 2013; Gyan, 2016) The experiences in this project have contributed extensively to the development of the KasaDaka platform, allowing for the hosting of voice-services on low-resource hardware. The VSDK is aimed to be an extension to the KasaDaka platform, and thus inherits many of the requirements of the KasaDaka platform. (Baart, 2016b) After the context and requirements of the VSDK have become clear, appropriate technologies have to be selected to form the VSDK architecture.

3.1.2 Development

With the preparatory work of defining requirements and choosing a development methodology done, the development of the VSDK can commence. The development phase will consist of building a system that allows for the development of voice-services, reducing the complexity of this task and the knowledge required to do so. The system will be integrated in to the existing KasaDaka platform, allowing for it to be used in the ICT4D context of sub-Saharan Africa.

3.1.3 Evaluation

After the development phase has ended (and a working version is available) the VSDK will be evaluated. This evaluation will be performed by groups of students that follow the ICT4D Master's course at the VU. The ICT4D course contains a practical part, in which the students develop their own voice-application for a specific ICT4D use-case. The students have previously used Voxeo Evolution and the KasaDaka platform, and during the 2017 course the students will use the VSDK to develop their voice-applications. The achievements and resulting voice-services of the students will be analyzed to determine the usability and capabilities of the VSDK, by developing applications for several ICT4D use-cases. (these use-cases are described in Appendix A) After the students have completed the practical part, they will be asked to fill in a survey containing relevant questions about their experience with the VSDK. The data from this survey will also be used to further evaluate the VSDK and determine changes and improvements for a next development iteration. Additional development iterations will be likely be necessary in order for the VSDK to mature and progress towards a pilot of the VSDK in the Sahel, but will fall outside of the scope of this research (due to time constraints).

4. REQUIREMENTS

This section will describe the requirements of the VSDK, which result from the goal of designing and building a development kit for voice services in the ICT4D context of sub-Saharan Africa.

4.1 Voice-services in sub-Saharan Africa

The process of developing and hosting a voice-service differs greatly depending on which part of the world it takes place in. While in highly developed countries (such as the United States and most of the countries in western Europe) the (digital) infrastructure, that is required for the development and hosting of software applications, is well developed and reliable; The opposite is the case in many developing countries where there is little infrastructure, and infrastructure usage is unreliable and (very) expensive. Besides infrastructure there are also economic and societal aspects that require consideration when designing and developing software. These constraints pose 'unusual' requirements on applications, but which are essential to the success of an application in the ICT4D context.

4.1.1 Societal challenges

Literacy rates in some countries of sub-Saharan Africa are relatively low compared to the rest of the world, in Burkina

[34]Voice-applications that rely on DTMF for user-interaction, do not use TTS or ASR and that do not involve data processing.

Figure 9: An example of the targeted usage environment of ICT4D voice-applications. Photo taken outside Ouahigouya, Burkina Faso.

Faso, Mali and Niger literacy rates are below 40%. (UNESCO, 2011) One of the consequences of this is that an information service targeted at this population must not use written text, but instead use for instance icons, voice or video to convey information.

While most of (sub-Saharan) Africa's countries are former colonies and thus have European languages as an official language, large parts of the population do not speak these languages. Rather, the local population speaks their own indigenous language, which is tied to their local region. Africa has around 2000 local languages, which each often have local dialects. (Heine et al., 2000) The majority of these languages are *spoken languages*, meaning that there is almost no literature in these languages. Furthermore, due to the population speaking these language is poor, these populations do not provide a profitable market for the development of TTS and ASR technologies in these languages. Taking into account these restrictions, these languages can be considered to be under-resourced languages. (Berment, 2004)

Most of the recently developed voice platforms that offer complex information services (e.g. Apple's Siri, Amazon Alexa, etc) make extensive use of Text-to-Speech, Automatic Speech Recognition and Natural Language Processing technologies. While these technologies are in wildspread use around the world, they require significant research and a lot of work in order to support a lanugage at a level that is sufficient for the usage in voice services. (Bagshaw et al., 2011; Black et al., 2000; Farrugia, 2005; McTear et al., 2016a) The number of languages that has well developed speech technologies is rising, but do (almost) not include any of the indigenous languages found in (sub-Saharan) Africa. When developing a voice-service, either these technologies have to be devloped to support these under-resourced languages, or the usage of these technologies is not feasible. While there are methodologies for the development of speech technologies in under-resourced languages, they are very resource intensive and are (especially in the case of spoken languages) very hard to automate. (Besacier et al., 2014; Vries et al., 2014) To solve this problem the VSDK should support the usage of pre-recorded audio fragments and thus function without the use of TTS, ASR or NLP. It should also be able to support many different languages simultaneously.

Previous experience from field visits has shown that another challenge of the sub-Saharan target user group is the little experience these users have with using technology and information systems. (Baart, 2017b) When developing a voice service it is thus important to make the interactions as simple as possible, taking in to account the low level of experience and guiding the user in the usage of the application. This low level of experience also makes it difficult for end-users to a priori state their needs in terms of functionalities. This makes the agile development methodology a good fit, as creating a voice-service using the traditional methodologies (such as the waterfall model) will almost certainly not produce the service the end-user 'wanted'. Thus it is likely that iterations during development will be required in order to achieve the desired functionality of the service. The VSDK should be able to support the agile development methodology, reducing the work required to make changes to a voice service to a minimum.

4.1.2 Resource and infrastructure constraints

The countries in sub-Sahara Africa are some of the poorest in the world, the majority of the population in Mali and Burkina Faso (the targeted countries in this research) lives on less than €2 per day.[35] In order for a voice-service to be of use to the general population, the cost of accessing and using it thus have to be very low. This implies that the users should be able to access the service without having to purchase a new device or service, but rather using a device they already own or have access to (e.g. a simple mobile phone, see section 1.2). The initial and running costs of a voice service should also be low enough to be affordable (and to provide sufficient return on interest) for the rural sub-Saharan population. The VSDK should allow the use of information services by simple mobile phones, which does not require the end-users to purchase a new device.

The (digital) infrastructure in these countries is unreliable and expensive, especially in the rural areas. While some villages have access to electricity, it is often unreliable. The majority of the population does not have (direct) access to electricity.[36] Access to internet is slowly becoming more common, (Poushter, 2016) but is very expensive and unreliable[37], due to a lack of local hosting and limited international (sea fiber) connections. Mobile networks have a good level of coverage, but are mostly 2G (voice and SMS only) in rural areas. While rates are not 'affordable', the usage of simple mobile phones is very widespread. (GSMA, 2016) Most people either own a (simple) mobile phone or share a mobile phone with family. In order for a service to be successful in this context, it should be able to run without relying on a stable power supply and internet connectivity and be accessible through existing mobile phones. The VSDK should be able to facilitate voice-service hosting and development without an internet connection.

Because of the above constraints, a very small amount of the population owns (or has access to) a computer. As

[35] https://data.worldbank.org/indicator/NY.GDP.PCAP.CD?locations=ZF

[36] https://data.worldbank.org/indicator/EG.ELC.ACCS.ZS?end=2014&locations=ML&start=2014&view=map

[37] http://100mega.ml/

such, there are only very few local technicians that are able to maintain local infrastructure and systems. The amount of technicians that have experience with voice-services will thus likely be extremely low. The implication of this is that in practice, maintenance will have to be done remotely or by flying in maintenance personnel, or by training local personnel. Because the infrastructure is often insufficient for remote maintenance (and the hardware/software providing the connection may fail as well), maintenance will probably require physical access to the device running the service. In order to keep maintenance costs to a minimum, the VSDK should be very reliable and allow for easy testing and problem diagnosis, reducing the amount of problems and the work required to fix them.

4.2 Sustainability

The long-term functioning of voice-services should be ensured for long after an initial deployment (and funding), if a long-term value to the development of the local community is desired. (Gyan, 2016) In order to achieve this, voice-services should be financially self-supporting. As the target user group is very poor, the amount that they are able to pay for a service will not be high and thus is unlikely to change. This implies that the elements that influence the costs (both initial and recurring) of running a service are essential to the financial sustainability of a service, i.e. these costs should be as low as possible. The costs of a service are determined by several factors, which are described in Table 1.

These costs vary depending on the complexity of the application and the relevant context. As this research mainly focuses on relatively small deployments of services, targeting small-scale farmers in sub-Sahara Africa, the costs of development and maintenance are relatively high. As services are highly customized to the local community, it requires a relatively high amount of work in relation to the amount of users and calls the service handles. This makes a service expensive, as the development costs are spread over a low number of users. There are many strategies that can be used to reduce the costs of a service, a few of which are described in Table 1.

4.2.1 Enabling local development and maintenance

As stated previously, the cost of the labor in development and maintenance of the service represents the majority of the costs involved with a voice service. A factor that causes these costs to (relatively) be even higher, is the difference in labor costs between the sub-Saharan countries and highly developed countries such as Europe (not to mention the travel expenses). It is thus of importance to speed up the development process in order to save costs. To reduce these costs even further, there should be sufficient local technical personnel to develop and support voice-services, which also eliminates the dependency on foreign developers. However previous field visits have shown that because of the low computer-literacy, local developers are hard to come by in many sub-Saharan African countries.[38] Because of this, creating a local 'developer community' for voice services currently seems to be far into the future. To at least achieve some of the cost savings, the underlying system that is used to develop voice-services should be as very easy, ideally not requiring programming experience. This should allow local workers with intermediate computer-literacy to perform simple maintenance tasks, after having received adequate training, addressing the lack of local developers. (Gyan, 2016)

[38]There are no figures of the amount of developers in sub-Sahara Africa, but the usage of GitHub could be considered to be an indicator of the amount of (open-source) developers: https://hackernoon.com/the-map-i-got-for-africa-8c8a958c686d, https://blog.ona.io/general/2015/01/01/github-africa-2015.html

4.3 Enabling voice-service development

Besides catering to the targeted usage context, the VSDK has to reduce the complexity and required knowledge for developing voice-applications. In order to achieve this simplification, it should be possible to develop a voice-service (prototype) within a GUI-based application, that does not require the developer to write VoiceXML or other code. The VSDK has to run on the existing KasaDaka platform and not require any additional installation on the developer's computer (or other device). The VSDK should (together with the documentation) guide the developer through the process of developing their first voice-application, reducing the risk of getting stuck by (often made) mistakes.

The process of application development that the VSDK provides should be based on component-oriented software development: providing the user with pre-packaged components that can be applied to build an application. (Nierstrasz et al., 1992) In this paper we will refer to this approach as the *building-block* approach, which can be implemented in the context of voice-services by providing templates for different types of interactions that are commonly used in voice-services. The developer should be able to build voice-services by applying and customizing these templates and build a (simple) voice service from a graphical user interface.

The applications that can be built by the VSDK should be able to support all languages and not require the use of Text-to-Speech technology. The VSDK should be able to record and store the end-user's language preference, and recognize users by their phone number (caller ID).

Interaction (without the use of TTS) takes place in the form of key presses using the user's phone keypad (DTMF[39]) to choose options in a menu of choices. In addition to key presses, voice-applications should also be able to record (and store) the user's voice, for inputs that are too complex for DTMF-based interactions. These inputs do not have to be processed further by the VSDK.

The VSDK has to include basic logging functionality, allowing for the analysis of the usage of a voice-service. The VSDK should be able to capture and store information about the call and the 'path' the user took through the application, as well as any inputs the VSDK received.

4.4 Summary of requirements

In order to get an overview of the (non-)explicit requirements posed in the previous sections, this is a summary of the requirements for the VSDK. The requirements are categorized in functional and non-functional requirements.

4.4.1 Functional requirements

- Should be able to support the development and hosting of DTMF-based voice-services.

[39]Dual Tone Multi-Frequency

Table 1: Costs of a voice-service

Category	Description	Strategy to reduce costs
Development	Costs of initial development of the voice-service. These costs mainly consist of developer salaries and software license costs.	Reduce and simplify the work required to develop a voice service. Use software components that are free and preferably open-source. (Nissilä, 2016)
Hardware	The costs of buying hardware for hosting the service.	Use hardware that is cheap to purchase and adapts well to the context (see section 4.1.2).
Infrastructure	The costs of using the telephony network (and internet connectivity). From the point of the voice-service this excludes the costs that users incur when calling into the service.	Ensure a good negotiating position versus telephone companies (telcos). This implies that the service platform should be independent of the underlying telephone infrastructure and switching between telcos should be simple.
Hardware maintenance	The cost of diagnosing problems and repairing or replacing hardware.	Use hardware that is reliable and easy to replace with low replacement costs.
Software maintenance	Costs of developers to fix problems and adapt the service to changing needs and contexts.	Simplify the maintenance process and the process of adapting a service to fit changing contexts. Using software components in a modular fashion allows for easy switching out components.

- Needs to be able to support the development of voice-services in all languages, this includes under-resourced languages. New languages can be added with minimal effort.
- Facilitate voice-service development from within a GUI.
- Provide *building-blocks* for the development of voice-services.
- Detect and notify the developer of design errors in the voice-application.
- Allow for the recording and storage of the user's input.
- User detection by caller ID, keeping track of the user's (language) preference(s).
- Include basic logging functionality, keeping track of the handled calls.

4.4.2 Non-functional requirements

- Should function reliably with minimal infrastructure, e.g. without an internet connection.
- Should not require TTS, ASR and NLP technologies.
- Needs to support an agile development workflow.
- Does not require the installation of software on the developer's computer.
- Needs to run on the existing KasaDaka platform (Raspberry Pi).
- Should include documentation and guide the user when possible.

5. SYSTEM ARCHITECTURE

This section will explain the architecture of the KasaDaka and the role of the VSDK in the overall architecture (visualized in Figure 10).

The inner workings and architecture of the VSDK will be discussed in Section 6.

We will start by covering all hardware and software components involved in the KasaDaka platform, which enable the system to serve information services over a phone connection. The functionality of each component will be described, as well as the rationale behind choosing for this specific component in relation to the previously described requirements.

5.1 Hardware

The hardware forming the foundation of the KasaDaka platform is the Raspberry Pi, which is a low-resource computer based on an ARM processor (like found in many smartphones). The main advantages of the Raspberry Pi are it's low power consumption (and subsequently low need for cooling), good on-board connectivity and the low price[40]. (and thus also a low replacement cost, see Table 1) The Raspberry Pi is a very popular product for experimentation and many projects, and is thus widely available. This makes it easy to replace should problems arise. The Raspberry Pi runs several distributions of Linux, of which Raspbian is a popular Debian based distribution. Almost all open-source software is available for the Raspberry Pi through Raspbian's package manager.

To provide the Raspberry Pi with connectivity to the local mobile phone network, a USB 3G modem is used. The exact make and model of this modem does not make much of a difference, as long as it is on the supported hardware list[41] of the `chan_dongle` Asterisk extension. (see Section 5.2.1)

5.2 Software

As mentioned when introducing the Raspberry Pi, the operating system running on the Raspberry Pi is Raspbian, which is based on Debian (a popular Linux distribution). On top of the Operating System run several applications that work together to provide the voice-service functionality.

5.2.1 Telephone exchange software: Asterisk

Asterisk is a very popular open-source Private Branch Exchange (PBX) telephony application. It is able to route calls from an incoming connection to it's destination using Voice-over-IP technologies. In the use-case of the KasaDaka platform Asterisk provides the connection between the phone network (3G dongle) and the VoiceXML interpreter. To enable Asterisk to interface with the 3G dongle an extension is required. `chan_dongle`[42] is an open-source Asterisk exten-

[40] A Raspberry Pi 3 (including case, power supply and SD card) costs around €60 at the time of writing.

[41] https://github.com/bg111/asterisk-chan-dongle/wiki/Requirements-and-Limitations

[42] https://github.com/bg111/asterisk-chan-dongle

Table 2: Description of flow through Figure 10

Label	Description
	Text in italics describes events that only take place when the call is first initiated.
A1	*An user (in this case a West-African farmer) calls the number of the SIM that is in the GSM modem of the KasaDaka. This call is routed through the local mobile-phone network and arrives at the KasaDaka's modem.* The user presses a key on the mobile phone, expressing a choice in the voice-interface, or speaks some input. The phone sends a DTMF signal on the phone connection, and/or sends the speech from the microphone over the connection.
A2	*After the GSM modem has received the incoming call from the GSM network, the* `chan_dongle` *Asterisk extension accepts the call.* The modem receives the user input (DTMF and/or speech) from the network and the input is processed by `chan_dongle`.
A3	The call is routed by Asterisk to the VoiceXML browser (VXI). This routing is configured in the Asterisk extensions configuration file. *This file also contains the URL of the VoiceXML document that VXI should load when accepting the call.*
A4	*VXI loads the VoiceXML URL defined in the configuration. This URL points to the initial VoiceXML document of the Voice-Service, which is hosted locally on the Raspberry Pi. The web-server (Apache2) handles this request.* VXI determines what action should be performed based on the user input. These possible actions include: loading a VoiceXML document (including the user's input in the HTTP request) and presenting another element in the already loaded VoiceXML document. If the latter is the case, VXI does not load a new VoiceXML document. (this skips A4 to A11, flow continues with A12)
A5	The web-server accepts the HTTP request of the VoiceXML file. The URL points to a voice-service hosted on the VSDK. The VSDK generates the VoiceXML files *on-the-fly* based on dynamic information stored in it's database. Any user input contained in the HTTP request is processed by the VSDK.
A6	The VSDK runs the necessary queries on the SQL database to generate the VoiceXML file, as well as store any new data from this call in the database. The database contains the structure of the voice-application, references to audio-fragments, as well as user-generated data and call logs.
A7	The VSDK returns the generated VoiceXML file to the web-server.
A8	The web-server serves the VoiceXML file through the HTTP connection to the VoiceXML interpreter.
A9	The VoiceXML interpreter parses the VoiceXML file. The VoiceXML file contains many references to audio files, which contain the spoken information. (see Section 6.3) In order to be able to play these audio files to the user, the VoiceXML interpreter proceeds by requesting each of the referenced audio files.
A10	The web-server receives the requests for the audio files. As these files are static they are retrieved from the file system.
A11	The web-server serves the audio files to the VoiceXML interpreter, which now has all necessary elements to present the VoiceXML file over the phone connection to the user.
A12	The VoiceXML interpreter 'displays' the interaction to the user by playing back the audio fragments referenced in the VoiceXML file generated by the VSDK.
A13	`chan_dongle` receives the audio on the phone connection generated by the VoiceXML interpreter and sends it to the GSM modem, which in turn sends it to the GSM network.
A14	The user hears the audio from the application through his/her mobile phone.
	The above described cycle repeats until the user has performed all desired tasks and either the VSDK generated VoiceXML document or the user terminates the connection.
B1	The user connects a device (smartphone or laptop) to the local wireless network, which is broadcast by the Raspberry Pi. The user accesses the administrator interface through the browser.
B2	The HTTP request arrives at the web-server. This request may include input from the user.
B3	The request is forwarded to the VSDK, which processes the request (and included user input) and generates the responding HTML web-page.
B4	The VSDK retrieves and stores/changes necessary data in the database.
B5	The VSDK returns the generated web-page to the web-server.
B6	The web-server serves the VSDK administrator interface page to the user's browser.
B7	The web-page is rendered and presented to the user.

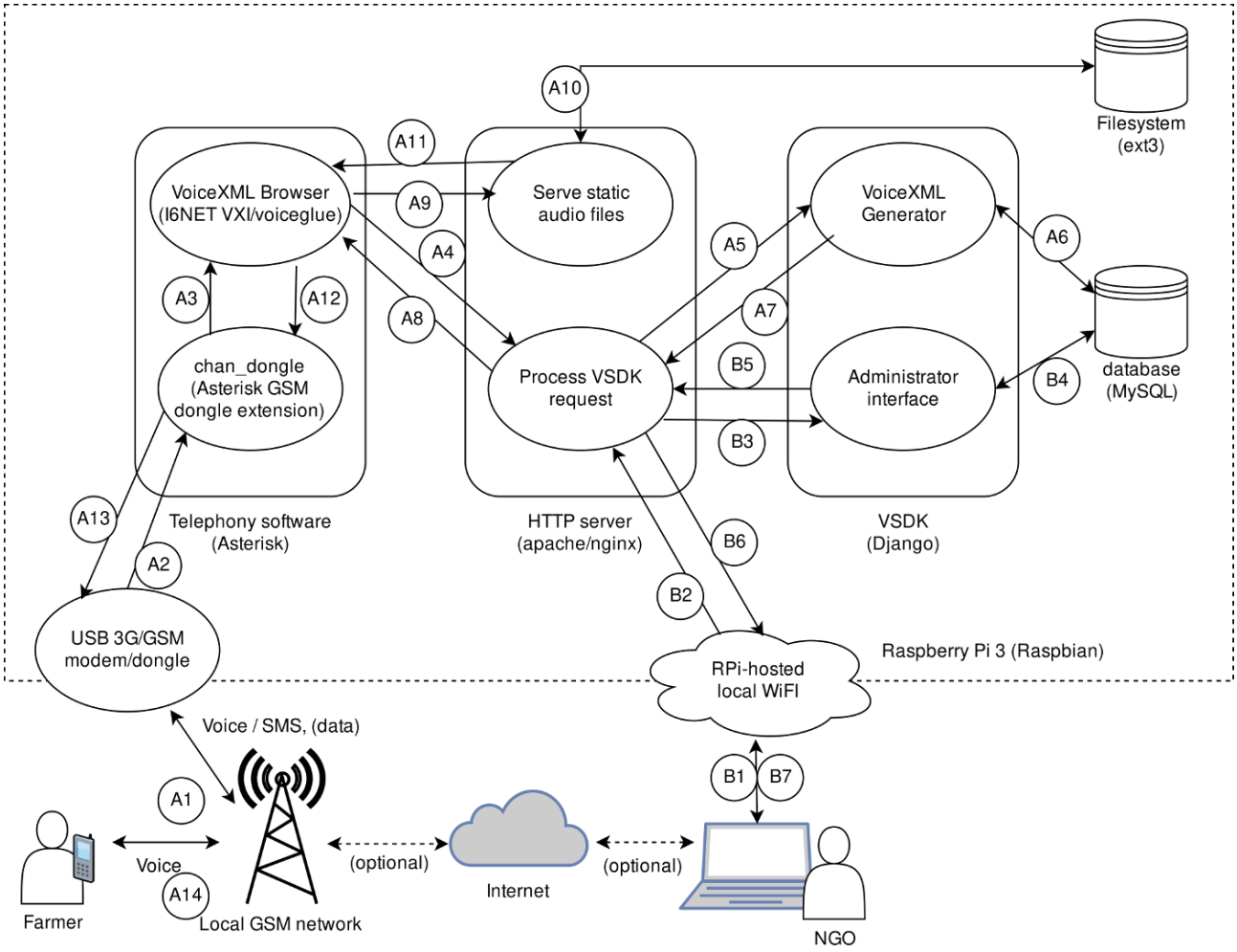

Figure 10: Overview of the KasaDaka system architecture

sion that provides connectivity between GSM/3G modems and Asterisk. It enables Asterisk to receive and place calls using the connected modem, as well as send and receive SMS messages.

5.2.2 *Voice application document standard: VoiceXML*

VoiceXML[43] is a document standard for voice applications, based on XML. It is a standard designed by the World Wide Web Consortium and is used for creating documents that describe voice-based interactions. It supports interactive voice dialogues between the computer and the user and usually contains written text that is later processed by a TTS engine. Responses by the user can happen through pressing a number on the phones keypad of by speaking (for this ASR needs to be available). As the voice applications that use the KasaDaka framework mainly focus on under-resourced languages, the use of TTS and ASR is not possible. Fortunately VoiceXML also supports the playback of audio files, much alike embedding an image in an HTML page. This allows the use of pre-recorded fragments to build up the voice services, but restricts the way of interaction to using the phone's keypad. A VoiceXML document is 'rendered' for the user in a way that is comparable to the rendering of a HTML file in a web-browser, but in this case is done by a voice browser.

5.2.3 *VoiceXML interpreter: VXI*

The software component that is used for 'rendering' VoiceXML files is VXI[44], a closed-source VoiceXML interpreter built by the company I6NET[45]. VXI connects with Asterisk as an end-point for incoming calls. When a call is redirected to VoiceXML a pre-configured URL is passed on to VXI, which it loads and 'displays' to the user as initial voice interaction. Normally this is the principal document belonging to a voice-service.

VXI currently is the only closed-source component used in the KasaDaka platform. While the goal is to use only open-source software, there are little open-source VoiceXML interpreter projects. A large project that would have the required functionalities, called voiceglue has been unmaintained for a long time and only runs on very old versions

[43]https://www.w3.org/TR/voicexml21/

[44]http://www.i6net.com/technology/voicexml-ivr/

[45]http://www.i6net.com/

of Ubuntu[46]. Committing to using voiceglue would thus restrict the framework to running on very old software, which makes it a bad choice. Furthermore, it is unknown whether voiceglue will run on ARM-architecture based systems such as the Raspberry Pi. Fortunately VXI includes a testing license, which allows VXI to be used free of charge, with the limitation of supporting only one concurrent call. For the use-case of the KasaDaka this is not (yet) a problem, as the current use-cases do not require a concurrent calls.

5.2.4 HTTP server: Apache

VXI loads the VoiceXML files it interprets over a HTTP connection, just like loading a HTML page on the web, but locally. In order to serve these files (and the audio files that are referenced in the VoiceXML files), a web server is required. There are many open-source web-servers, one of the most used is Apache 2.[47] As the web-server fulfills a relatively simple role in the platform, the web-server used does not matter very much.

5.2.5 Database: MariaDB

As a data-store MariaDB[48] is used as a SQL server. MariaDB is an open-source fork of the well known MySQL database server software.

SQL was chosen to use for the VSDK instead of noSQL (linked-data was used previously in the KasaDaka framework) for the VSDK, because MySQL is more common and well-known by developers. Additionally it is supported out-of-the-box by many frameworks, such as Django. This lowers the barrier of entry to developing custom voice-services using the VSDK, as in most cases developers do not need to learn a new database concept.

5.2.6 VSDK development framework: Django (Python)

In order to make the VSDK easy to extend by developers, Python was the programming language of choice as it is a popular language that is well supported and has several popular web-frameworks. As VoiceXML documents are comparable to HTML documents, most web-frameworks can also be used to generate VoiceXML files. Django[49] was chosen as the Python web-framework, as it has very good and extensive documentation, is well-supported and follows a Model-View-Controller methodology, giving structure for inexperienced developers. (Krasner et al., 1988) Django is open-source and has a rich collection of projects and libraries that can be used to extend it's functionality. The MVC methodology used by Django includes automatic database-structure generations (migrations) based on the models that are defined. Furthermore Django also has a built-in administrator interface which allows the management of the data within the application. Based on the defined models, appropriate management interfaces are generated automatically, which can be extended or changed if necessary. These features greatly reduce the amount of work the developer has to do when developing a new application, which makes Django a good framework to enable the VSDK to be used for *rapid-prototyping*. Django has a good implementation of internationalization functionalities, which enable the interface of the administrator interface to be translated to different languages. This functionality is of great importance when developing applications in an international context.

In Section 6 the role of Django in the VSDK is explained in more detail.

[46]A popular distribution of Linux
[47]https://httpd.apache.org/
[48]https://mariadb.org/
[49]https://www.djangoproject.com/

5.3 Interaction diagram of software components

In Figure 10, the previously described components of the architecture of the KasaDaka are shown. In order to gain a better understanding of the role that these components fulfill and how they work together, the flow through these components are described for the two main use-cases of the KasaDaka:

(A) An end-user (farmer) calling a voice-service hosted on the KasaDaka In this use case is the main purpose of the KasaDaka platform: offer voice-based information services.

(B) A voice-service maintainer/developer accesses the VSDK to monitor or change a voice-service

A detailed description of these flows and the interactions between the components in the architecture are described in Table 2.

6. VSDK FEATURES AND DESIGN DECISIONS

This section describes the features, as well as the challenges and corresponding solutions that were implemented in the VSDK.

6.1 Design and development of voice services

The main goal of the VSDK is to support the development of voice-services in the ICT4D context. As the voice services are hosted on a Raspberry Pi and internet connectivity is not to be expected, the development of voice services has to happen offline, but connected to the Raspberry Pi that will be hosting the application. Using a web-based interface is preferable to running a development environment on a computer because it solves problems with compatibility (different devices, operating systems) and reduces complexity (does not require installation of software). Another advantage of this approach is that the development and hosting of the service are integrated, allowing for instantaneous results (and testing) of changes made to the application.

The Django framework provides a built-in administrator web-interface, which allows management of the data in the application. From the data models used in the application, Django generates simple but appropriate management interfaces that can be extended or overridden to adapt to more advanced workflows. Using this built-in functionality saves a lot of effort in not having to build an interface for the data stored in the application.

The structure of the voice-application is stored in the database, using Django's model functionality. When an element in the voice-application is requested by the user in a phone call, the VoiceXML interpreter (VXI) requests the element through HTTP. Django then retrieves the information about this element from the database, and uses a view to 'render' the element in VoiceXML. The VoiceXML interpreter then interprets this VoiceXML file and 'displays' it to the user.

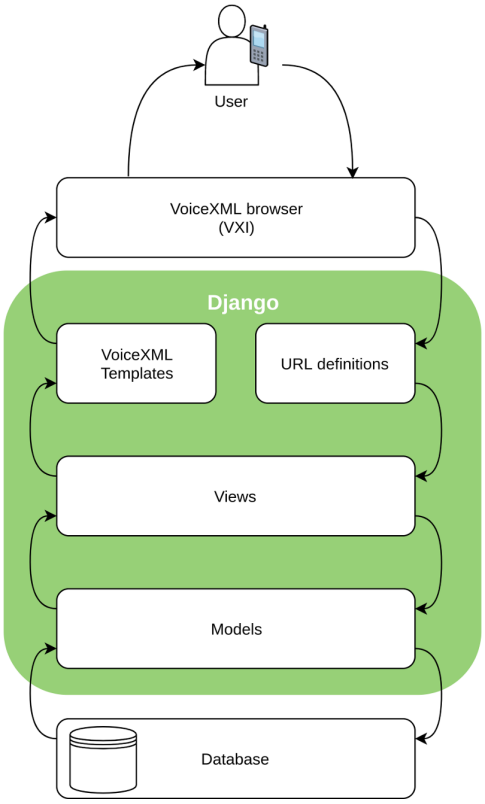

Figure 11: A visualization of the Django internal architecture.

6.1.1 *Voice-application structures*

While the interactions in voice services are always different, most of them can be generalized to a small set of interaction types, such as making a choice, playing back an audio message, or recording (voice) input of the user. The VSDK provides a set of these *building-blocks*, which consist of a VoiceXML template, view and an administrator interface to use and customize them. The current set (which will be expanded in the future, see Section 10.1) consists of a menu-based interaction Recording of user voice and the playback of a message. This set is sufficient for the development of simple voice-services.

6.1.2 *VoiceXML generation*

As mentioned in Section 5.2.6, Django is based on the MVC principle. The views define which data is presented to the user, and when a view is called (by a request from a browser) this data is processed by a template (which places the data in a HTML page). However as VoiceXML documents are much alike HTML (web-)pages, the template can also define the file structure of a VoiceXML file. In this way the VoiceXML files for voice-services are dynamically generated, in the same way as web-applications would generate HTML pages. The internal architecture of Django as implemented in the VSDK is shown in Figure 11. A more detailed explanation of the inner workings of the Django framework can be found in the Django documentation[50].

The VSDK stores the elements that comprise a voice-service in the database. Each voice-service element is of a certain type, e.g. a choice, a message that is played back or a recording of the caller's voice. These interaction types each consist of several properties which are stored in a model (and thus in the database), and corresponding views and templates. New instances of these interaction components can be created through the administrator interface, after which the properties of the newly created element can be entered. (see Figure 12) When the VoiceXML browser requests the element, Django retrieves the necessary information from the database and renders this information using the appropriate template, producing a VoiceXML file. This VoiceXML file is then parsed by the VoiceXML browser and presented to the user. A more detailed explanation of the VoiceXML generation implemented in the VSDK can be found in the VSDK documentation[51].

6.2 User registration and recognition

The configuration of Asterisk (and subsequently VXI) is set to request a voice-service from the VSDK over HTTP. In this HTTP GET request, the phone number of the incoming call (Caller ID) is included. The VSDK uses this caller ID to recognize users that have been registered in the system before and uses their preferences (e.g. the language to use). This user recognition does not require any effort from the user's perspective and can (although not very secure) be used as an authentication mechanism, for instance for determining an user's role and access level.

If the phone number is not in the system, the user is first requested to choose a language in which to continue. Afterwards (if the voice-service is configured to do so) the user is asked to speak their name, which can later be used to identify the user in the application's functionality. The recording of the user's name can for instance be used by an administrator (such as an local NGO worker) to give the user a written name in the system.

6.3 Speech and languages

As discussed in section 4.1.1 voice-services in the ICT4D context have to support under-resourced languages, for which there are no speech technologies available. The VSDK supports different languages in voice services by utilizing pre-recorded audio fragments that are relevant for the use-case domain. During the development of the service, all the necessary voice-fragments are recorded in the different languages in which the service has to be accessible.

6.3.1 *Slot-and-filler TTS*

The method of producing speech used in the VSDK is based on the use of pre-recorded fragments of speech that refer to an element in a voice-service, giving each element a *voice label*. This is a basic implementation of a *slot-and-filler* TTS system that supports a limited domain. (Black et al., 2000; Jůzová et al., 2014) This works by dividing each sentence used in a voice-service into reusable parts, and playing back the relevant ones during usage of the voice-service, when speech has to be generated. The boundaries of these parts are defined by whether the fragment is static (e.g. the welcome message) or a dynamic element (e.g. the days of the week). While this is a very simple technique compared to the many sophisticated TTS implementations, it works for all languages and requires relatively little resources to set

[50] https://docs.djangoproject.com/

[51] http://kasadaka-vsdk.readthedocs.io/en/latest/

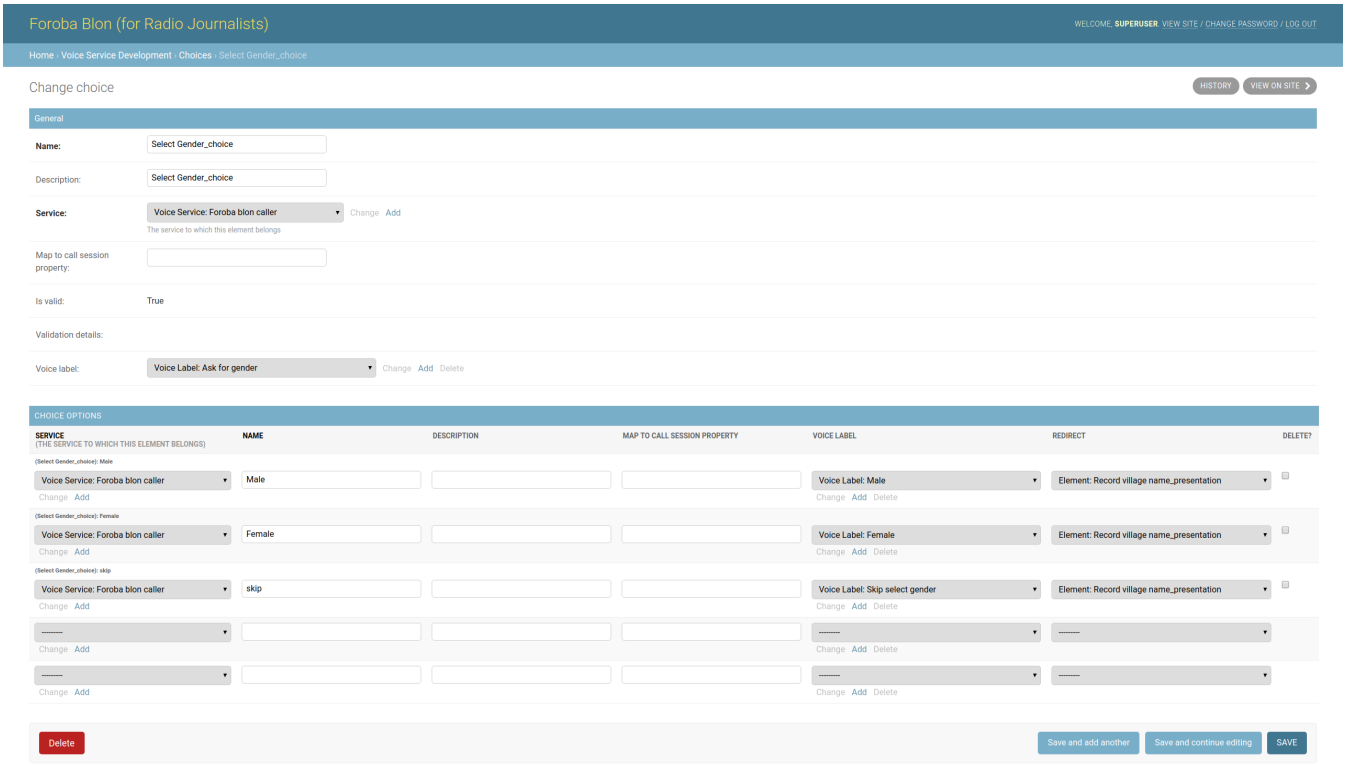

Figure 12: A screenshot of the configuration page of a Choice element in the VSDK.

up. The main limitation of this system is that the scope of usage is limited to the domain for which the fragments have been recorded. As every word and (partial) sentence has to be pre-recorded, it is not possible to generate speech for all circumstances and contexts; Thus *slot-and-filler* is only useful in applications that have a limited domain, which produces a set of required fragments that is recordable in a reasonable amount of time.

Even though recent TTS systems produce very good results, systems based on pre-recorded fragments are still used (for languages that have well developed TTS systems) extensively in certain contexts. (McTear et al., 2016b) The main area where these systems are used are in automated announcement systems, such as the announcements of the railways and airports[52]. These applications have a clearly defined domain and are thus very suitable for the use of pre-recorded fragment based systems. The advantage of these systems is the clarity of the announcements, as the voice is not synthesized but from a selected speaker with a clear and well-understandable voice that sounds more natural. (Lewis, 2010) Characteristics that make these kinds of systems recognizable include short pauses between fragments and a slightly unnatural emphasis or flow between fragments, combined with an extremely natural sounding voice.

In the context of a voice-application, the structure of the application is known and defined before hand. Usually when considering a menu structure inside a voice-application (where the user is given several options, each of which can be chosen by pressing a number on the phone's keypad), there is a predictable pattern of voice-fragments that is required. These fragments include: the initial question, a description of each of the options in the menu and several supporting fragments: the numbers 0 to 9 and small fragments instructing the user to press a number to make a choice. As each call follows the same pre-defined call flow structure, these fragments can be reused on every call. The size of the voice fragments is determined by the element in the voice-service to which it belongs. In practice this means that the fragments are usually the size of either (parts of) sentences or several words (describing an object).

During the VOICES project a more sophisticated implementation of a *slot-and-filler* system was developed, which may be implemented into the VSDK in the future; This system will be described in more detail in Section 10.1.2. (Kleczar, 2017)

6.3.2 *Voicelabel recording*

To create a voice-service in the VSDK, every audio fragment needs to be recorded and present in the VSDK. In the administrator interface, the developer can upload audio fragments or record them inside the development interface. The VSDK then stores these fragments and connects them to the voice-labels, which are in turn referenced by elements in the voice-service. During a call, the VSDK looks up all the elements necessary for generating the requested VoiceXML file, and references the corresponding audio fragments (in the language of the user that is calling) in the generated VoiceXML document. The VoiceXML browser (VXI) then loads these audio fragments and relays them to the user.

[52]For an example, see: http://aviavox.com/#listen

6.4 Call sessions

In order to manage variables and keep track of calls processed by a voice application, the VSDK keeps track of calls in the form of call sessions. These sessions are stored in the VSDK's database and contain essential information about the call, such as the caller ID, time and language, as well as the 'path' the user has taken through the application. This functionality enables the service's administrator to analyze the usage of the application and can also be used for statistical analysis and debugging purposes.

6.5 Testing and debugging

The process of developing a voice-service is different from that of developing a conventional application or web-based service. While in the development of websites and other conventional applications the usage of automated testing applications is considered normal, for voice-applications such testing suites can not be used. This is because testing the application in a realistic environment involves calling the service with an actual phone and trying out all functionalities in the application. In traditional applications a bug usually causes an error message to be presented or the application to crash. In voice-applications a small bug or problem can mean that the user hears nothing, or that the connection is suddenly terminated without explanation to the user. Due to the lack of a TTS system that covers all domains (see Section 6.3.1), it is not possible to present a (specific) error message to the user. This causes a very negative experience for the user, reinforcing the need for extensive testing by the developer in order to reduce the amount of errors in the application. While an experienced developer of voice-application can usually quickly determine the cause of an error; From previous experience of working with students during the ICT4D course, an inexperienced developer can spend a very long time without progressing on gaining insight on the issue at hand (usually resulting in an trial-and-error approach to the problem).

6.5.1 Voice service validation

In order reduce the complexity of voice-service development, the VSDK includes some built-in checks and validations that check the voice-service for completeness and tries to warn the developer about any detected issues. An example of a problem that often occurs during the development of a service, is a missing voice label of an element in a certain language. The voice-service will function without any problems while missing this voice label, but when an user accesses the application in his/her language (usually a language not spoken by the developer and thus not used during testing) this element will be skipped by the VoiceXML interpreter (as the audio fragment cannot be loaded). This results in an incomplete sentence towards the user, leaving the user confused and potentially making the service useless to users speaking this language.

To prevent this kind of error from happening, the VSDK validates all elements belonging to the voice-service for voice labels in all languages that are listed as being supported by the service. If a voice label turns out to be missing or not accessible (or some other problem arises), the VSDK will alert the developer in the administrator interface. This significantly reduces the amount of time spent testing the application by the developer, and prevents frustration during development.

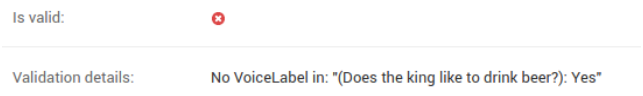

Figure 13: An example of a validation error message displayed by the VSDK indicating a missing voice label, which is an error that is easily overlooked by a developer, but can cause the service to not function correctly.

7. EVALUATION

The VSDK was evaluated during the ICT4D course at the Vrije Universiteit Amsterdam. For the 2017 edition of the course, the VSDK was used to create voice-services for various ICT4D use-cases. This served as a thorough evaluation of the first version of the VSDK and provided insight in the different approaches to voice service development taken by the students, as well as feedback for further development of the VSDK. This first version of the VSDK can be considered the minimal viable product (MVP) and is limited to basic features only. It allows for the creation of basic voice-service services that can be used for rapid-prototyping purposes. Using a graphical interface, voice-services consisting of simple choices with associated options and messages can be designed without having to write any code. These prototypes can be made in a quickly and without extensive knowledge of the underlying technologies, which is useful for rapid prototype development and evaluation; After set-up, a simple service can be developed and tested through calling in under 30 minutes, however the development of complex use-cases take a considerable amount of time. In order to fully implement more complex use-cases, the developer will have to extend the VSDK with data-models that are relevant to the use-case and write corresponding views to allow the user to interact with the data. Extending the VSDK requires knowledge of several programming languages and frameworks, including (but not limited to) Python, Django and VXML.

7.1 Set-up of the ICT4D course

The ICT4D course consists of a theoretical part, focusing (among other things) on the many aspects of ICT4D and it's impact on the world, as well as a practical part, in which the students (in groups of 3 or 4) created their own voice application that is relevant for a use-case of their choosing. Over the past years the practical part of the course has evolved, allowing the students to experience the current state of development of voice service research at the VU.

The students iterated three times during their development, producing three versions of their voice applications and the accompanying paper. The final iteration results in a working voice-application for the chosen use-case, as well as an accompanying paper explaining the context of the use-case (requirements, business model, etc) and the design of the application. During the following sections we will mostly focus on the technical part of the development, as these results are the most relevant to the further progress in developing the VSDK.

During the course there were several working sessions, in which the students were able to ask for help on any questions or problems they encountered in the development pro-

cess.[53] In addition to the working sessions, one lecture was dedicated to the inner workings of the VSDK (and the underlying KasaDaka platform) and an instruction into setting up and deploying the VSDK[54].

The experiences and conversations during the working sessions as well as the resulting application and paper, provide valuable information about the development process of the students and their experience with using the VSDK.

It is worth noting that while the students were encouraged to use the VSDK to build and extend their application, this was not obligatory and one student group decided to build their own VXML generating application (not using the VSDK).

A summarization of the three iteration steps in the ICT4D course:

1. Creating an interaction prototype on the Voxeo platform, consisting of one or more VXML files that demonstrate a voice interaction with the user. The goal of this iteration is for the student to get to know VXML and the general structure of a voice interaction.

2. Creating a functional prototype using the VSDK and extending functionality of the prototype. In this iteration the students learn the workflow of the VSDK and recreate their interaction prototype from the first iteration (written in VXML) using the VSDK. The iteration also includes an extension of the functionality (this can be in any form e.g. extra language support, more options, etc), however extending the models and views included in the VSDK do not yet have to be extended upon.

3. Finalizing towards an implementation prototype, extending the VSDK functionality. The final iteration should produce a working implementation of the voice-service, in which the functionality of the VSDK has been extended to include models, views and templates that implement the required functionalities for the use-case. The final paper describes the use-case and how the built voice-application fits in this context, as well as documentation on the voice service and it's development (including a demo scenario walk-through).

For more information about the course, please refer to the Vrije Universiteit Amsterdam study guide.[55]

7.2 System Architecture during the ICT4D course

The system architecture that was used during the ICT4D course is a variation on the architecture that is used for deploying voice-services in low-resource conditions (i.e. sub-Saharan Africa). The goal of this architecture is to allow development without a Raspberry Pi and to allow easy modification of the VSDK code. Because the KasaDaka platform is set up in a modular fashion, it is possible to easily exchange modules for others with similar functionality. In this case the hosting of the VSDK is moved 'into the cloud'.

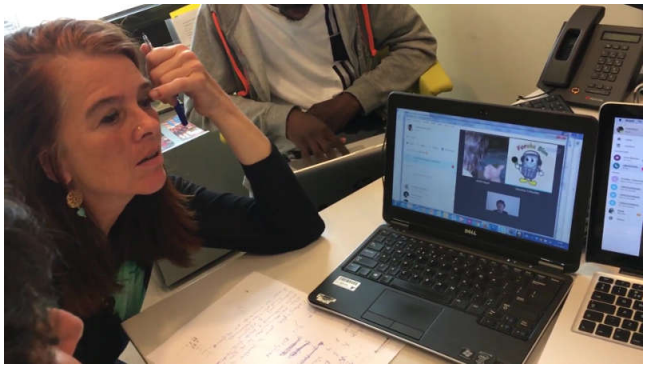

Figure 14: Evaluation of voice-services during the ICT4D course: Skype session with Julien Ouedraogo (Burkina Faso) and Amadou Tangara (Mali)

7.2.1 Cloud-based hosting of the VSDK

As it was not possible to give each group a Raspberry Pi to run the VSDK on, the system architecture during the ICT4D course was changed to a partially cloud-based architecture. The students used Heroku[56], a cloud platform that facilitates easy deployment and hosting of web-applications. In order to keep the functionality and usage scenarios as close to the real platform as possible, calling and testing the service was possible through a shared Raspberry Pi. This Raspberry Pi hosted each group's static audio files and a script (called the VoiceXML Switcher) that allowed the student groups to change the VXI configuration (see Section 5.2.3) to their application. The applications could then be called with a regular (mobile) phone by calling the number of the Raspberry Pi's GSM modem. A diagram describing this adapted architecture, as well as a description of the interactions is provided in Appendix C.

7.2.2 Feedback and testing of voice-applications

After each iteration the students received feedback on their applications and papers, which they could use the improve their applications for the next iteration. The applications were tested and evaluated by a member of the W4RA, who has a lot of knowledge about the contexts of the use-cases and has extensive experience with the end-users of voice-services. The applications were also evaluated with key stakeholders from Mali and Burkina Faso, through a Skype call. (see Figure 14) This feedback of experts on the use-cases allowed the students to better fine-tune their applications to their contexts.

7.3 Results

In total 31 students participated in the practical sessions, spread over 10 groups. Each of the groups selected a use-case in the context of ICT4D in sub-Saharan Africa and built a

[53]These working sessions were organized by the researcher and Francis Dittoh, a PhD student in ICT4D with experience in working with the VSDK and the KasaDaka platform.
[54]A recording of this session is available on Youtube: https://www.youtube.com/playlist?list=PLIZ5uFyl_SghojAEpzoUtnEczZLRKaAG0
[55]https://www.vu.nl/nl/Images/vu_Study_m_information_sciences_20-7-2017_tcm289-851223.pdf
[56]https://www.heroku.com/
[58]Built their own application using PHP and MongoDB
[58]Used an extension shared by group 1

Table 3: ICT4D course results

Group	Students	Use-case	Results			
			Group used VSDK to build application	The application was functioning correctly	Group extended VSDK with custom data models	Group extended VSDK with custom types of interactions
1	3	Citizen Journalism	yes	yes	yes	yes
2	3	Weather Information	no[57]	yes	n/a	n/a
3	3	Animal Health	yes	yes	yes	yes
4	3	Animal Vaccination	yes	yes	yes[58]	no
6	3	Weather Information	yes	yes	no	no
7	4	Diary Value Chain	yes	yes	yes	yes
8	3	Citizen Journalism	yes	yes	no	no
9	3	Diary Value Chain	yes	no	yes	yes
10	3	Weather Information	yes	yes	yes	yes
11	3	Animal Health	yes	no	yes	yes
total	31		90%	80%	78%	67%

voice-service during 6 weeks of the course. The use-cases are described in Appendix A.

The pool of students consisted of a mix of many different backgrounds, and different levels of experience in developing services and software. While there were some computer science students, most students had very little programming experience and none had previous experience in the development of voice services. In order to be able to compare the (technical) results and achievements of the groups (even though the use-cases are different), the level of technical development of the applications can be assessed by some general measures. These measures aim to determine the extent to which the students used and extended the VSDK in developing the application for their use-case. While students were not required to extend the VSDK (as this requires significant technical knowledge, which can not be expected from all students), they were encouraged to do so in a way that is useful for the specific chosen use-case. Besides extending the application, the students could also choose to focus more on the business or societal aspects of their use-case.

7.3.1 Analysis of results

In Table 3 the students' results are listed, the applications are available on GitHub[59].

From this data can be concluded that almost all of the students were able to build a working voice-service in the time provided. A majority of the groups extended the VSDK with custom data models relevant for their use-case, around two thirds of the groups also implemented specific interactions (views and VXML templates) with their data models (a more involving task).

These results show that the first iteration of the VSDK succeeds in being a tool that enables quick development of voice-service prototypes. In order to extend these prototypes into fully functioning applications, the VSDK needs to be expanded upon with relevant data models and (if required) specific interactions with this data through voice. These more advanced and diverse functionalities were not yet included in the version of the VSDK used in this evaluation. While the process of expanding the VSDK requires some effort, it shows that the foundation offered by the VSDK is able to support many different voice-services by expanding the included functionality.

[59]https://github.com/abaart/ICT4D-2017

7.4 Evaluation and feedback

While these students have a relative high level of experience with information technologies (there was a mix of mostly computer science and information science Master's students following the course) and thus are not very suitable to compare with the target user group of sub-Sahara African NGO-workers; The students are able to quickly and easily provide feedback about the VSDK, which can be used to further expand and refine the VSDK, which provides feedback without having to travel. This approach, while not entirely comparable to having a feedback session with the actual African user group, provides feedback that will be used to iterate upon the design and functionality, while saving the time and money that would be required to travel and do a similar session in sub-Saharan Africa.

At the end of the course the students were asked to fill in a short survey on their experience with creating a voice service and using the VSDK. The goal of this survey is to learn about the process that the students went through as they developed their first voice service. The survey consisted of statements about the usefulness of the VSDK, which had to be answered in a Likert scale. There were also open (qualitative) questions about VSDK features, improvements and suggestions, as well as questions about the development process. (encountered difficulties, easy aspects, etc.) The students were asked to fill in the survey per group, after internal discussion. This was done to prevent inaccurate results that could be caused by the internal division of work on the project; This could result in some members of a group having done all the development work in the VSDK and others having written the use-case description and business-model, which could cause inaccurate survey results.

The quantitative results of the statements are printed in Table 4, a summary of the qualitative results of the open questions are included in Appendix D. The raw data gathered in the survey is also available online. (Baart, 2017a)

7.4.1 Survey results analysis

From the responses to the statements about the usefulness of the VSDK can be concluded that the (limited) function-

Table 4: ICT4D course student survey results on a 5-point Likert scale (1 Agree, 5 Disagree)

Statement	Mean	σ
S1: "The VSDK seems suitable for creating simple voice-application prototypes"	2.1	1.73
S2: "The VSDK's approach is suitable for the development of voice-services for 'small languages'".	2.1	2.00
S3: "The VSDK was more user friendly than manually writing VoiceXML files"	2.7	1.00
S4: "By 'hiding' the VoiceXML layer, people without programming knowledge should be able to develop voice-service prototypes using the VSDK (excluding the set-up process)"	2.2	1.87
S5: "The validation system (detecting any errors in your application) was useful to me"	3.1	1.22
S6: "The VSDK was useful for developing simple voice-application prototypes"	2	1.87
S7: "After following the Django tutorial, I understood the mechanisms of models, views and templates"	2.6	2.92
S8: "By extending the basic functionalities in the VSDK with my own, I was able to create a voice-service relevant to my use-case "	2	1.87
S9: "The functionalities included in the VSDK would have saved me time, compared to building a VoiceXML generating system myself"	2.3	1.22
S10: "The VSDK's approach seems suitable for the development of complex voice-services (by extending it)"	2.1	1.58
S11: "I would consider using the VSDK again when developing a voice-application in the ICT4D context"	2	1.58
S12 "Being able to record audio fragments in the web-interface would improve the development process	2.4	1.22
S13: "Having uploaded audio files be automatically converted to the right format would improve the development process"	1.7	2.83

ality that the VSDK provided the students, offers a feasible platform for the development of voice-applications. The responses to statements S1, S2, S4, S6 and S11 show a general positive response to the suitability of the VSDK's *building block* approach to voice-service development. For the development of more complex voice-service applications (statements S8 and S10) the responses are also positive, while it should be noted that the functionality included in the version of the VSDK that was used was lacking in included functionality for advanced voice-services.

More neutral responses were given to statement S3, which is slightly out of line with the responses to the previously discussed statements. A possible explanation for this result is that the students' experiences with writing VoiceXML files was limited to simple applications, which did not include the usage dynamic data, *slot-and-filler* (see Section 6.3.1) and multi-language support; This caused the comparison between writing VoiceXML documents and using the VSDK to not be fair, as the experiences were different in nature. In hindsight this question was not useful in the context of the evaluation, as the students did not have experience to answer the question appropriately.

A feature of the VSDK that did not seem to fulfill its goal is the validation system (S5), which checks the voice-service for any missing voice-labels and references. What the problems were with this functionality needs further research. Another possibility is that the usefulness of the feature was not very apparent to the students, as they did not have experience developing without this functionality.

The experiences of working with the Django framework were mixed, which shows by the high standard deviation in statement S7. This matches the researcher's experiences during the working sessions. While some students (with some previous programming experience) were quick to pick up the workings of the Django framework, for others the learning curve seemed steep. This further emphasizes the need for a complete programming-less development experience for the targeted sub-Saharan user group.

Features that were not included in the VSDK, but that emerged (and appeared to be useful) during the course are generally agreed upon to be welcome additions to the feature-set of the VSDK. (S12 and S13)

7.4.2 *Qualitative results from survey*

A summarization of the qualitative results from the open questions in the survey can be found in Appendix D. The responses show that in general the functionality of the VSDK enables the development of simple voice-services. The most challenging factor in the students' development process was the learning curve of learning Python and Django in order to extend the functionality of the VSDK. As the VSDK's functionalities will be expanded further, extending the VSDK should no longer be necessary, eliminating this problem. (see Section 10)

Another aspect that comes up repeatedly is the debugging process in voice-service development; Debugging voice-services is difficult and clumsy compared to regular application debugging. While most programming languages have debuggers and clear error messages that point at the problem, this is not the case with voice-services. In most cases when an error occurs during a voice-service the telephone connection is terminated, or a long silence occurs. There is no further information about what went wrong, making debugging the error difficult, often resulting in an *trial-and-error* problem solving approach by the developer. Furthermore, there are currently no automated testing methodologies for voice-services, which implies that a lot of time in the development process is spent in testing the application. Considering that many voice-services in the ICT4D context support several languages and have many branching structures, the amount of possibilities that need to be tested is very high.

8. DISCUSSION

While achieving the end-goal of enabling complete development and maintenance of voice-applications without programming would enable many sub-Saharan NGOs to develop and run voice-services, and the VSDK attributes a step towards this goal; This goal is not yet in reach and requires significant additional work in refining the VSDK and KasaDaka platform, training local personnel and creating developer communities in several Sahel countries. So while the VSDK is a first and significant step towards the self-sufficiency of ICT4D voice-services in the Sahel, the complete process of self-sufficient voice-service development communities will take (many) more years.

The evaluation of the VSDK shows that the *building block* approach to voice-service development is a feasible approach to improving user-friendliness. While the idea of using templates (as *building blocks*) is not new and is implemented in other voice-development environments, (see Section 2) the VSDK does add several significant functionalities to the field of voice-service development. However, where other voice-service development environments also provide templates for voice-service interactions, the platforms on which their voice-services have to be deployed mostly use TTS and ASR for the facilitation of interactions. Furthermore these services are all provided as commercial products which require (expensive) licenses and limit deployment to their own specific platforms. These platforms are often hard to connect to local phone numbers in sub-Saharan countries, and require (expensive) contracts with telephone companies. Due to the closed-source nature of these commercial environments, adapting or extending the functionalities of these environments is only possible by the company that developed the product. As these companies are all based in highly developed countries, the costs of these adaptations and extensions will be very high, defying the reason behind a voice-service development environment: reducing the costs of development.

In conclusion, the VSDK is a significant contribution in the field of ICT4D voice-services. The VSDK allows for the development of voice-services in the ICT4D context by supporting all languages and running on low-resource hardware. By enabling the inception of development communities in developing countries, it is now possible to develop voice-services tailored to the local context with a significantly reduced cost. The open-source license of the VSDK, combined with the (future) availability of local developers allows for *complete ownership* of sustainable voice-services by local communities in developing areas of the world.

8.1 Limitations

The VSDK currently allows the development of simple voice-services by using a GUI and fulfills the requirements posed in Section 4.4. However many use-cases require more advanced styles of interactions. Most of these interactions involve handling data which is specific to the use-case, such as rainfall measurements or market information. The current set of interaction templates provided by the VSDK is still limited, and does not support handling data inputs and outputs. In order to achieve these functionalities the VSDK can be extended by custom data models and templates required by the use-case, however in order to make these extensions the developer needs to be able to write Python code and VoiceXML documents. This limitation prevents the VSDK of being suitable for more complex voice-services, as 'traditional' voice-service development skills are still required. Although this limits the reduction in development complexity, it is still an improvement over the previous way of voice-service development on the KasaDaka platform, which consisted of writing new (unique) views and templates for every interaction in a voice-service. In the case of a custom extension to the VSDK, the functionality of this extension can be reused throughout the application and shared with the rest of the development community (through GitHub). Because the extension will be built on the foundations of the VSDK it is easily reusable by others. An example of this occurred during the ICT4D course, where a student group extended the VSDK with user voice recording functionality, which was shared and reused by other groups. Furthermore the administrator interface can easily utilized by these custom extensions, which allow voice-service maintainers (without programming knowledge) to change settings and other elements of the extension's functionality. Thus after the development of the extension is completed, maintenance can still be performed by others without knowledge of it's inner workings, maintaining the advantage of ease of use offered by the VSDK.

9. CONCLUSION

The results of the evaluation of the VSDK show that the *building blocks* approach to voice-service development is suitable for reducing the complexity of voice-service development and simultaneously speeding up this process. The VSDK provides an abstraction layer to the development process, allowing for voice-service development in a web-browser. By providing templates for the most used interactions in voice-services, a developer can quickly build applications by applying and customizing these templates to create a voice-application. It is not required for the developer to have experience in programming or any of the other technologies that are used to allow a voice-service to function. Furthermore, the basic usage of the VSDK on the KasaDaka platform does not require any software installation on the developer's computer and can function without an internet connection. These characteristics of the VSDK provide a significant improvement over the past development workflow, which required extensive knowledge and experience in all the technologies used for the hosting of voice servies. This reduction in complexity of the development process, combined with the support for under-resourced languages and the ability to run on low-resource hardware, allow developers using the VSDK to quickly develop and deploy ICT4D voice-service prototypes on the KasaDaka platform, as well as maintaining and changing elements of existing voice-services; Without requiring these developers to have knowledge of the underlying technologies.

From these findings we can confirm the main hypothesis of this research: by generalizing the interactions in voice-services and providing these interactions as *building-blocks* in a development environment, inexperienced users are able to build simple voice-applications by deploying and customizing these *building-blocks*. (see Section 3)

With further refinements and extension of the VSDK's functionality and documentation, the VSDK can be used autonomously by NGO workers, radio presenters in the Sahel region and other (ICT4D) voice-service use-cases. Future voice-service maintainers and developers can be trained

in the usage of the VSDK, which is a process that is realistic and much less invoved than learning to work with languages such as Python and VoiceXML. By being able to source the maintenance and development personnel locally (instead of hiring expensive foreign developers) has the potential to cause a large decrease in costs, which in turn reduces the total *cost of ownership* of voice-services. The low *cost of ownership* of voice-services using the VSDK and the KasaDaka voice-service hosting platform, combined with the offered functionalities in supporting under-resourced languages, make the VSDK and the KasaDaka platform a comprehensive solution for sustainable voice-services in the ICT4D context.

10. FUTURE WORK

In order to further progress towards the goal of sustainable voice-services in the Sahel, there are steps that have to be taken in all aspects of the life-cycle of voice-services. Following is a summarization of the issues and improvements that are relevant at the time of writing.

10.1 Further development of the VSDK

As discussed in section 8.1, the VSDK is not yet suitable for the development of complex use-cases because it currently supports only a limited set of interaction templates. The simplest way of improving this situation is by implementing more templates into the VSDK, which allows it to support more use-cases. These interaction templates will be largely based on the use-cases brought forward by the research of the W4RA (see appendix A) and will thus cover a variety of interactions. However it is impossible to include all possible interactions, and thus the VSDK will need to be continuously expanded and refined to cater to new use-cases.

10.1.1 Linked data & dynamic data model generation

Another problem in implementing more advanced use-cases is the storage of data that is specific to the use-case. Django provides advanced data modeling techniques and allows management of this data through the administrator interface. However the definitions of the data models are defined in the Python code. This implies that any custom data models have to be written in Python and cannot be created or changed easily (through a web-interface).

Together with the issue of a limited set of interaction templates, this issue prevents the VSDK of reaching the goal of voice-service development of any use-case through a graphical interface. In order to solve this limitation, the VSDK should implement *dynamic model generation* functionality. This functionality allows the developer to define and implement data models relevant to the voice-service and design specific voice interactions with data in these models. While there are some implementations of this functionality for the Django framework, this is still a difficult problem to solve as not only the data models should be generated from the web-interface, but also the voice interaction templates that access/handle this data. Furthermore these custom models can become obstacles in the process of sharing data between multiple applications, as each voice-service is likely to store their data in a different way.

Another possible solution to this problem is to use *linked data* to store the voice-service structure and data. *Linked data* does not require data to adhere to a strict model, rather it can be used with vocabularies that define the properties of relations between objects. One of the advantages is that when data is exchanged between two nodes using different data models, this data can easily be combined into one data-set. Users can use well known vocabularies that exist for many domains, which allows voice-services to use and exchange information with other stores of linked-data. Linked-data allows for new ways of sharing and accessing data and allows data to be used in new contexts. (Bizer et al., 2009) Previous research by Nieland (2013) has shown the possibility of creating voice-interfaces to linked-data.

The Data2Documents vocabulary allows content management in RDF, which can be used to store and generate HTML web-pages from triples, using templates similar in functionality as those found in Django. (Ockeloen et al., 2016) If Data2Documents would be adapted to create VoiceXML documents from triples, the structure of voice-services can be stored in triples as well and reusable interaction templates as well as custom VoiceXML templates can be used interchangeably.

Although these possibilities are very interesting and would solve some of the current issues with the VSDK, the move from SQL to a triple store would likely mean that Django is no longer suitable as the framework for the VSDK. The main downside of discarding Django is that all the web-interfaces which are currently generated by Django will have to be recreated by hand. Additionally the loss of rich documentation and availability of extensions will increase the learning curve of future VSDK development, so this is a complex consideration.

10.1.2 Advanced implementations of slot-and-filler

During the Lwazi II project an application was developed that provides a TTS implementation of *slot-and-filler*, built with under-resourced' languages in mind. (Calteaux et al., 2013) The initial goal of this research was to develop full TTS systems for South African languages.

During the VOICES project, Daniel van Niekerk adapted this system to be suitable for the development of *slot-and-filler* systems for voice-services. During the project, support was developed for the African languages Bambara and Bomu. Instead of the usage of voice fragments that is currently used in the VSDK, this system takes as inputs recordings of complete sentences which are then split into words to form a word repository. Spoken outputs are then formed by filling slots with spoken words from the repository. The advantage of this system is a higher reusability (as words can be reused), which reduces the amount recording that is necessary to cover a domain. Making adjustments to the messages used in the voice-service can be done without having to re-record audio fragments, as the text that is to be spoken is stored in written form, without the use of voice-labels. Another advantage is that the size of the domain in which the TTS system can be used increases, as the fragments are now the size of words, which less tied to the context in which they were recorded. (Marsman, 2017)

Recently Kleczar (2017) has researched this system and provided a working implementation of the system in a voice-application on weather information. This proves that the system is still usable for usage in ICT4D voice-services today. If this improved *slot-and-filler* implementation is included in the VSDK, the amount of work required in the recording of voice-fragments could be reduced and the VSDK

would become more flexible in the handling of changes to spoken messages. However, in order to be useful in the context of the VSDK, ease of use in setting up this *slot-and-filler* system should be ensured by the VSDK.

10.1.3 Alternatives for VXI

The VoiceXML browser used in the KasaDaka platform works well for current use cases, but may become a limiting factor in the future. VXI is the only software component used in the KasaDaka platform that is not open-source and requires a license, which is undesirable in the ICT4D context. An alternative to VXI will probably have to be found in the future. A possible solution would be to 'adopt' the voiceglue project, however this is a very large amount of work and currently above the skill level of the researcher. Another possibility would be to build a custom VoiceXML browser that is linked to Asterisk. As the applications for the KasaDaka platform do not require TTS and ASR and have relatively simple means of interaction, it is not necessary to support all of the VoiceXML functionality. A simple VoiceXML browser implementation would be sufficient to ensure the independence, affordability and stability of the KasaDaka platform into the future.

10.1.4 Other enhancements

Besides these main points of refinement, there are many other features that could be added to the VSDK's functionalities, including:

- Automatic testing of voice-services
- Allowing transactions using mobile money
- An interface to the top-up process of prepaid SIM cards
- Data exchange between KasaDakas
- Backup of voice-services and data
- Implementation of a *BIP*[60] voting and callback system.
- Automatic outgoing calls (animal vaccination use-case)
- Implementing sending and receiving SMS
- Developing tools for voice-service debugging

10.2 Overcoming hardware limitations

While the conbination of a Raspberry Pi with a GSM dongle is a good fit to the conditions of the ICT4D context, there are also several limitations which can become problematic. Especially when KasaDakas are to be rolled out on a large scale it is important that the hardware is stable, which means the following problems need to be addressed:

- The Raspberry Pi does not have a *Real Time Clock*, which means that when the power is removed from the Raspberry Pi and there is no internet connection, the date and time will be incorrect from that moment forward.
- The Raspberry Pi does not function well on 5 volts, as it requires 5.2 volts to function correctly. This means that regular micro-USB phone chargers and power banks cannot be used.
- When the power supply fails when the Raspberry Pi is running, there is a chance at corruption of the SD card, as it was writing during the power failure. When this happens (and this is a question of time with an unreliable power supply as in the ICT4D context) the Raspberry Pi will no longer boot and hence become useless.
- The GSM/3G modems used in the KasaDaka platform are being phased out for LTE modems, which do not support voice over the GSM network. Currently these modems can still be found on the second-hand market, but this will become problematic when the KasaDaka platform has to scale.

10.2.1 Designing specialized hardware

These limitations can be addressed in several ways, but the most rigorous way is to design custom hardware for the KasaDaka platform. By designing a custom Printed Circuit Board for the KasaDaka, these problems can all be addressed by including hardware on the board. This board could be based on the Raspberry Pi, including all the same components of a regular Raspberry with the addition of components that solve the shortcomings of the Pi.

10.3 Pilot Burkina Faso

In a recent trip to Burkina Faso (Baart, 2017b) the researcher has taken part in a workshop with local innovative farmers, that was centered around weather information, specifically rainfall. During the workshop the researcher worked together with local radio presenters to develop a demonstration of a rainfall voice-service overnight, which was shown on the next day of the workshop. As the researcher now has experience with the stakeholders of this use-case, it is a likely candidate for a pilot deployment of a KasaDaka running the VSDK. This pilot can be used to do further research on voice-services and serves as a long-term test of the VSDK.

10.4 Starting a local development community

When the VSDK and the KasaDaka platform has matured sufficiently and has been tested in the field, the benefits of the improvements in development usability can be reaped by training local voice-service developers. In order to reach the goal of self-sufficient voice-services, these communities should be able to teach and maintain themselves, should be spread throughout the Sahel region and the development should provide a sufficient wage for the developers. Farmer communities could then hire these local voice-developers to develop and host a voice-service tailored to their information needs.

Acknowledgements

I would like to thank Victor de Boer for his help and guidance in this research project and the many caffeine buzzed brainstorming sessions. Also I would like to thank Hans Akkermans, Anna Bon, Wendelien Tuijp and Gossa Lô for their support and inspiration throughout my projects. Special thanks go to Francis Dittoh for assisting me with the ICT4D working sessions. I also want to thank Amadou Tangara and Julien Ouedraogo for their feedback and support during the trips to Mali and Burkina Faso and with the evaluation of voice-applications.

[60] Also known as 'flashing' in English, calling a number and disconnecting before a connection has been established. This technique is used in voting about opinion or the content of radio programs, as it is a free means of communication and allows for many concurrent votes to be cast.

References

Aker, Jenny C and Isaac M Mbiti (2010). "Mobile Phones and Economic Development in Africa". In: *Center for Global Development Working Paper* 211.June 2010, pp. 1–43. ISSN: 08953309. DOI: 10.1257/jep.24.3.207.

Akkermans, Hans, Chris van Aart, Victor de Boer, Nana Baah Gyan, Anna Bon, Wendelien Tuyp, and Amadou Tangara (2013). *VOICES Deliverable D3.1: m-Agro Knowledge Sharing Field Pilot Final Evaluation*. Tech. rep.

Ali, Maryam and Savita Bailur (2007). "The challenge of sustainability in ICT4D-Is bricolage the answer". In: *Proceedings of the 9th international conference on social implications of computers in developing countries*. Citeseer.

Baart, André (2016a). *Andre Report Bamako Trip Oct 2016*. DOI: 10.6084/m9.figshare.5688799.v1.

– (2016b). "Creating a flexible voice service framework for low-resource hardware : extending the KasaDaka". In: *Bachelor Thesis Vrije Universiteit Amsterdam*.

– (2017a). *ICT4D 2017 survey RAW results*. DOI: 10.6084/m9.figshare.5702452.v1.

– (2017b). *Trip report workshop Gourci (Burkina Faso) Feb 2017*. DOI: 10.6084/m9.figshare.5688769.v1.

Bagshaw, Paul, Etienne Barnard, and Olivier Rosec (2011). *VOICES Deliverable D3.1: Report on state of the art and development methodology*. Tech. rep.

Berment, Vincent (2004). "Méthodes pour informatiser les langues et les groupes de langues ´peu dotées˘". PhD thesis. Université Joseph-Fourier-Grenoble I.

Besacier, Laurent, Etienne Barnard, Alexey Karpov, and Tanja Schultz (2014). "Automatic speech recognition for under-resourced languages: A survey". In: *Speech Communication* 56.Supplement C, pp. 85–100. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2013.07.008.

Bizer, Christian, Tom Heath, and Tim Berners-Lee (2009). "Linked data-the story so far". In: *Semantic services, interoperability and web applications: emerging concepts*, pp. 205–227.

Black, Alan W and Kevin A Lenzo (2000). *Limited domain synthesis*. Tech. rep. Carnegie-Mellon University.

Boer, Victor de, Nana Baah Gyan, Anna Bon, Wendelien Tuyp, Chris Van Aart, and Hans Akkermans (2015). "A dialogue with linked data: Voice-based access to market data in the sahel". In: *Semantic Web* 6.1, pp. 23–33.

Bon, Anna (2016). "ICT4D 3.0, - An adaptive, user-centered approach to innovation for development." In: CAiSE 2016.

Bon, Anna, Victor de Boer, Nana Baah Gyan, Chris van Aart, Pieter De Leenheer, Wendelien Tuyp, Stephane Boyera, Max Froumentin, Aman Grewal, Mary Allen, Amadou Tangara, and Hans Akkermans (2013). "Use Case and Requirements Analysis in a Remote Rural Context in Mali". In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*.

Calteaux, K, Febe De Wet, C Moors, D Van Niekerk, B McAlister, A Sharma Grover, T Reid, M Davel, E Barnard, and C Van Heerden (2013). *Lwazi II Final Report: Increasing the impact of speech technologies in South Africa*. Tech. rep. CSIR.

Chapman, Robert and Tom Slaymaker (2002). "ICTs and Rural Development: Review of the Literature, Current Interventions and Opportunities for Action". In:

Chhetri, Deepak (2013). "Voice User Interface Design for m-Event Organizer". In: *Vrije Universiteit Amsterdam Master Thesis*, pp. 1–28.

Davies, Tim and Duncan Edwards (2012). "Emerging Implications of Open and Linked Data for Knowledge Sharing in Development". In: *IDS Bulletin* 43.5, pp. 117–127. ISSN: 1759-5436. DOI: 10.1111/j.1759-5436.2012.00372.x.

de Boer, Victor, Pieter De Leenheer, Anna Bon, Nana Gyan, Chris van Aart, Christophe Guéret, Wendelien Tuyp, Stephane Boyera, Mary Allen, and Hans Akkermans (2012). "RadioMarche: Distributed Voice- and Web-interfaced Market Information Systems under Rural Conditions". In: *Advanced Information Systems Engineering* 7328, pp. 518–532.

Farrugia, Paulseph-John (2005). "Text to speech technologies for mobile telephony services". In: *Pace and Cordina [PC03]*.

Fuchs, Christian and Eva Horak (2008). "Africa and the digital divide". In: *Telematics and Informatics* 25.2, pp. 99–116. ISSN: 07365853. DOI: 10.1016/j.tele.2006.06.004.

GSMA (2016). *GSMA report: The Mobile Economy: Africa*.

Gyan, Nana Baah (2016). "The Web, Speech Technologies and Rural Development in West Africa". PhD thesis. Vrije Universiteit Amsterdam.

Gyan, Nana Baah, Victor de Boer, Anna Bon, Chris van Aart, Hans Akkermans, Stephane Boyera, Max Froumentin, Aman Grewal, and Mary Allen (2013). "Voice-based web access in rural Africa". In: *Proceedings of the 5th Annual ACM Web Science Conference on - WebSci '13*, pp. 122–131. DOI: 10.1145/2464464.2464496.

Heeks, Richard (2008). "ICT4D 2.0: The next phase of applying ICT for international development". In: *Computer* 41.6, pp. 26–33.

Heine, Bernd and Derek Nurse (2000). *African languages: An introduction*. Cambridge University Press.

Ittersum, Martin K. van, Lenny G. J. van Bussel, Joost Wolf, Patricio Grassini, Justin van Wart, Nicolas Guilpart, Lieven Claessens, Hugo de Groot, Keith Wiebe, Daniel Mason-DCroz, Haishun Yang, Hendrik Boogaard, Pepijn A. J. van Oort, Marloes P. van Loon, and Kenneth G. Cassman (2016). "Can sub-Saharan Africa feed itself?" In: *PNAS Early Edition* 113.52, pp. 1–6. ISSN: 0027-8424. DOI: 10.1073/pnas.1610359113.

ITU (2016). "ICT Facts and figures 2016". In: p. 8. ISSN: 14713063. DOI: 10.1787/9789264202085-5-en.

Jůzová, Markéta and Daniel Tihelka (2014). "Minimum Text Corpus Selection for Limited Domain Speech Synthesis". In: *Text, Speech and Dialogue: 17th International Conference, TSD 2014, Brno, Czech Republic, September 8-12, 2014. Proceedings*. Ed. by Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala. Cham: Springer International Publishing, pp. 398–407. ISBN: 978-3-319-10816-2. DOI: 10.1007/978-3-319-10816-2_48.

Kleczar, Justyna (2017). "General purpose methodology and tooling for Text-to-Speech support in voice services for under-resourced languages". MA thesis. Vrije Universiteit Amsterdam.

Krasner, Glenn E, Stephen T Pope, et al. (1988). "A description of the model-view-controller user interface paradigm in the smalltalk-80 system". In: *Journal of object oriented programming* 1.3, pp. 26–49.

Lewis, James R (2010). *Practical speech user interface design*. CRC Press.

Lô, Awa Gossa (2014). "The power of knowledge sharing : innovative ICTs for the rural poor in the Sahel". In: *Bachelor Thesis Vrije Universiteit Amsterdam*.

Mantel, Stephan (2014). “Small hardware solutions for voice services”. In: *Vrije Universiteit Amsterdam Bachelor Thesis* June.

Marsman, Rudy (2017). “The implementation of Artificial Intelligence in the re-use of old media corpora”. MA thesis. Vrije Universiteit Amsterdam.

McTear, Michael, Zoraida Callejas, and David Griol (2016a). “Creating a Conversational Interface Using Chatbot Technology”. In: *The Conversational Interface*. Springer, pp. 125–159.

– (2016b). “The conversational interface”. In: *New York: Springer* 10, pp. 978–3.

Nieland, Rianne (2013). “Talking to Linked Data : Comparing voice interfaces for general-purpose data”. In: *Vrije Universiteit Amsterdam Master Thesis*.

Nierstrasz, Oscar, Simon Gibbs, and Dennis Tsichritzis (1992). “Component-oriented Software Development”. In: *Commun. ACM* 35.9, pp. 160–165. ISSN: 0001-0782. DOI: 10.1145/130994.131005.

Nissilä, Jussi (2016). “Promoting Scalability and Sustainability of Ict4D Projects Using Open Source Software”. PhD thesis, p. 117. ISBN: 9789512966189.

Ockeloen, Niels, Victor de Boer, Tobias Kuhn, and Guus Schreiber (2016). “Data 2 Documents: Modular and Distributive Content Management in RDF”. In: *Knowledge Engineering and Knowledge Management: 20th International Conference, EKAW 2016, Bologna, Italy, November 19-23, 2016, Proceedings 20*. Springer, pp. 447–462.

Papeschi, Franco, Nicolas Chevrollier, Filipe Pinto, Carole Salis, and Hans Akkermans (2011). *VOICES Deliverable D6.1: Mobile Training Lab Requirements*. Tech. rep.

Poushter, Jacob (2016). “Smartphone ownership and internet usage continues to climb in emerging economies”. In: *Pew Research Center* 22.

Reij, Chris, Gray Tappan, and Melinda Smale (2009). *Agroenvironmental transformation in the Sahel: Another kind of" Green Revolution"*. Vol. 914. Intl Food Policy Res Inst.

Sendzimir, Jan, Chris P. Reij, and Piotr Magnuszewski (2011). “Rebuilding Resilience in the Sahel: Regreening in the Maradi and Zinder Regions of Niger”. In: *Ecology and Society* 16.3, p. 08. ISSN: 17083087. DOI: 10.5751/ES-04198-160301.

Toyama, Kentaro (2010). “Can technology end poverty”. In: *Boston Review* 36.5, pp. 12–29.

UNESCO (2011). *UNESCO report: Regional overview: sub-Saharan Africa*.

Vries, Nic J. de, Marelie H. Davel, Jaco Badenhorst, Willem D. Basson, Febe de Wet, Etienne Barnard, and Alta de Waal (2014). “A smartphone-based ASR data collection tool for under-resourced languages”. In: *Speech Communication* 56.Supplement C, pp. 119–131. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2013.07.001.

Warschauer, Mark and Morgan Ames (2010). “Can One Laptop per Child save the world’s poor?” In: *Journal of international affairs*, pp. 33–51.

Webster, Frank (2014). *Theories of the information society*. Routledge.

APPENDIX

A. SUB-SAHARAN ICT4D VOICE-SERVICE USE CASES

A.1 Citizen Journalism: Foroba Blon

This use case was brought forward by a use case gathering effort of the W4RA in 2011. Foroba Blon is meant to be a citizen journalism platform, based on the ability to leave voice messages.

Foroba-Blon is mainly used by radio stations, enabling interaction with listeners. Listeners are able to leave voice messages, stating their opinion or reporting on local news events. The presenter of the radio can include these messages in their program, enabling local and independent journalism.

When a user calls the system, he/she is greeted and invited to leave a message. The message is then recorded and stored on the system, where the radio operator is able to access and manage the left messages through a web interface. (Bon et al., 2013)

A.2 Market Information: Radiomarché

The Radiomarché use case is a marketplace platform, enabling users to exchange information about agro-related products in the regional market. This platform simplifies the trading of products by interconnecting the markets of several villages and allowing access to this market data without the need for travel. Users can request current market information, e.g. the current offers and prices of a produce; As well as advertise their produce that is for sale to potential customers. (Gyan et al., 2013; Gyan, 2016) In co-operation with local radio stations, the current offerings on the system are combined in to a spoken message which is regularly updated and broadcast on local radio stations.

A.3 Weather Information: Meteo

The goal of the meteo use-case is to provide weather information. The details of the implementations of this weather information service differ depending on the local needs and context. In many of the Sahel countries reliable weather information is not available, while this data (rainfall in particular) is of great importance for local farmers.

The weather data can be sourced from an online API (e.g. OpenWeatherMaps), weather sensors connected to the Raspberry Pi, or by crowdsourcing where users contribute their own measurements. In the latter case users call the service and enter their measurements, which is then presented to other users.

A.4 Animal Health: DigiVet

The DigiVet use-case provides a service that diagnoses diseases in cattle, such as cows and chickens. By asking the users simple yes/no questions about the symptoms of the animal, a rough diagnosis can be made without having to travel and pay for a veterinarian (which is often an issue in sub-Saharan Africa). Based on the diagnosis the system can recommend further care or connect the user with a veterinarian.

DigiVet has been implemented by Lô (2014) using a graphical interface on a small touchscreen connected to a Raspberry Pi, but can also be implemented through a voice-service.

A.5 Animal Vaccination

The Animal Vaccination use-case is a reminder service for animal vaccinations. Cattle can get many diseases which often prove to be lethal. However many of these deaths can be prevented by vaccinating the animal.

For example, chickens can be vaccinated for relatively little cost, this can be done by the farmer himself. However in order for the vaccinations to be effective, they have to be administered repeatedly over a long period. This is difficult as the farmer is unable to read or write and thus is not able to read and follow a vaccination schedule.

The voice-service can allow the farmer to keep track of the animal vaccinations by providing the farmer with automated spoken reminders about when to vaccinate their cattle.

A.6 Diary Value Chain: Milk

The milk use-case is a voice-service that improves the logistics and value-chain of fresh milk. Because of the temperature in the Sahel and the lack of refrigeration, milk has to be processed in a factory within four hours of production. Due to a lack of information about the amount and location of milk produced, it is difficult to meet this requirement, which leads to loss and a lower quality of milk.

The milk voice-service aims to improve this situation by connecting the milk producers with the milk processing factory. The milk producers announce their produced milk to the system, and the pick-up service then knows where there is milk that needs to be transported, preventing unnecessary driving and delays. By improving the value-chain more milk is available to the local market and less milk is wasted.

B. INSTALLATION OF VSDK & SOURCE CODE

More information about the installation process of the VSDK and the documentation about usage of the VSDK can be found in the documentation: http://kasadaka-vsdk.readthedocs.io/en/latest/.

The source code of the VSDK is available on GitHub: https://github.com/abaart/KasaDaka-VSDK

C. SYSTEM ARCHITECTURE DURING ICT4D COURSE

In Figure 15 the architecture during the ICT4D course is shown. Marked are the interactions that changed significantly compared to the standard architecture, which are described below.

1. The developer deploys the VSDK (with own modifications or extensions) on the Heroku platform.
2. The developer uploads the static audio files to the Raspberry Pi through FTP.
3. The developer changes the configured VoiceXML URL of VXI through the VoiceXML Switcher script hosted on the Raspberry Pi.
4. When the application is called, VXI retrieves the configured VoiceXML document URL, which is hosted on Heroku.

D. QUALITATIVE RESULTS OF ICT4D COURSE SURVEY

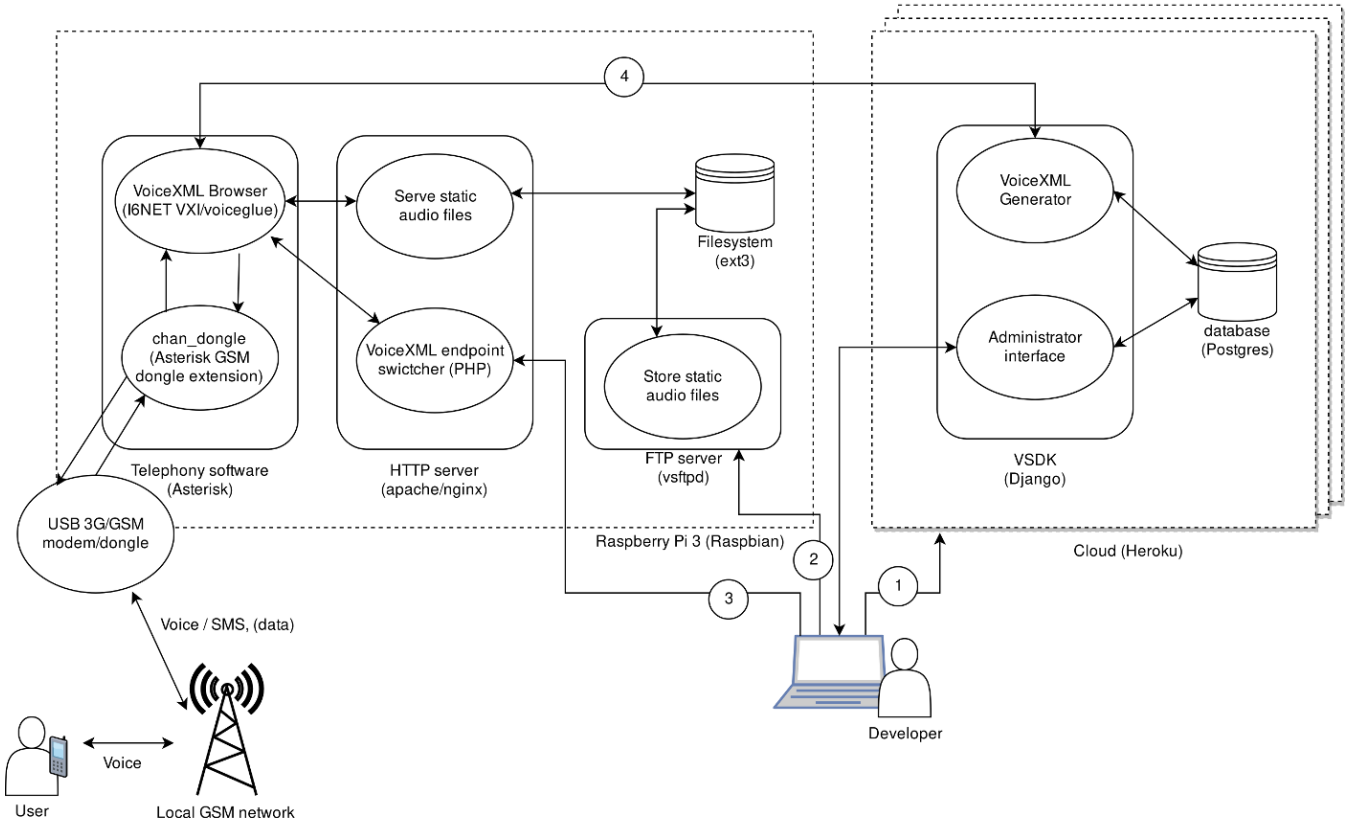

Figure 15: System architecture during the ICT4D course

Following is a summarization of the responses of the qualitatative questions of the ICT4D course student survey. The raw data is also available. (Baart, 2017a)

What are [...] features you would like to see added [to the VSDK]?

- A built-in testing system
- Element duplication
- Defining element ordering
- Adding voice-fragment transcription
- Allow loops in call-flow
- Implement support for VoiceXML snippets
- Visualization of the call-flow
- Call forwarding and SMS support
- Easy integration of APIs and external data stores
- Performing data management through a voice-service

What are aspects [of the VSDK] that you would like to see improved?

- The debugging workflow, there is no feedback from the VoiceXML interpreter and debugging through Heroku did not work well
- More comprehensive instructions for testing applications
- Error messages and error handling
- Being able to record voice-fragments directly from the VSDK

What are the aspects you found most difficult in developing your prototype on the VSDK?

- Error handling
- Not having a backup feature and not being able to copy and move elements
- Adapting and expanding the VSDK code
- Learning Django, Git and Heroku
- Debugging on Heroku
- Connecting external APIs
- The difference in ease of use between the VSDK and VoiceXMl
- Converting the audio files to the right format

What are the aspects you found most easy in developing your prototype on the VSDK?

- Uploading audio files
- Managing languages
- Creating a linear call flow
- Creating a simple voice-service
- Creating new voice-service elements
- Adding new models and templates

What are some of the most difficult problems you (tried to) tackle(d) during the development process?

- Fixing errors in the application
- Learning to use Python and Django to extend the VSDK
- Debugging dus to unclear errors
- Setting up a testing environment to test the extensions to the VSDK
- Connecting other services to the VSDK platform
- Recording voice in the VSDK and retrieving stored data