Institutt for informatikk

Information Infrastructure Growth: Installers as Frontier Objects

Masteroppgave

Øystein Skadsem

30. juni 2008

Abstract

This thesis aims at improving our understanding and management of Information Infrastructure Growth. An open source Health Information System and its constituting participants, users and projects is proposed as a specific example for researching methods and tools for stimulating this growth. The main focus will be on the aspects of installation and upgrade routines/procedures, bearing in mind the open source nature of the project and how this both limits and improves upon the available choices for said tools.

Research has been done as an action research project in two parts. The first was a local effort in a state capital in India including development and implementation of installation and upgrade routines for the HIS in question. The second part was further research and refinement of this work for the global HIS project, i.e. the local solution scaled up to a generic solution for the project.

The thesis will also discuss how technical solutions interplay with social issues and local capacity, most importantly how simplicity and usability can play an important role in empowering less technically skilled users, incidentally having the potential to reduce the level of capacity building or training needed for each user. Examples here include how simple tools can move the boundaries of the implementation of an Information System, as less knowledgeable users gain the ability to push the system to new locations, effectively growing the installed base of the II. The expansion to new user groups is similarly affected, as such tools can lower the bar of acceptance for new users previously unfamiliar with the IS. An easy and error-free installation is essential for the important first impression, and in this type of system, the ability to easily roll out in a large scale is a crucial selling point in demonstrations for health officials and large interest organizations alike. Additionally, geographical distance between the potential users and the people with the knowledge needed to set up the IS, becomes much less relevant when the difference between the groups is marginalized.

Acknowledgements

Thanks to Knut Staring for guidance and bringing me into HISP, it has truly been a great experience.

To Gianluca Miscione for guidance and sticking up with me.

To friends and familiy for input, proof-reading and moral support.

Table of Contents

Illustration Index

E-mail Listing

Code Examples

Chapter 1 Introduction

1.1 The Research Project

Several attempts at creating installation routines for DHIS2 has been made throughout the first two years of development, most of them aimed at various specific implementations. A rough time line:

- May 2005, Kerala, India
- April - August 2006, Ho Chi Min City and Hue, Vietnam
- Fall 2006, Oslo
- Early 2007, Ethiopia
- August - October 2007, Kerala, India

In addition to being geographically spread out, these projects were also using different technology and approaches.

The first implementation in Kerala came as a result of the local health authorities wanting an open source web based solution, ruling out DHIS 1.x as a possibility. At the time DHIS2 was far from complete as well, but it was seen as an opportunity for a great pilot project, with the parts of DHIS2 that was not complete at installation coming later on as they were finished, thanks to the modular structure of DHIS2. So in May 2005 one of the developers from the university of Oslo went to Kerala to train a local team and aid with installation. At the time, this entailed unzipping a directory structure including the necessary packages and manually creating the required shortcuts for starting and stopping the system. (Nordal 2006) There was not a lot of systems involved in this first round, and:

> A sensible approach to rolling out the system was not really our focus.
>
> *Mail 1: To me from a DHIS2 developer involved in the pilot project, subject: DHIS2 installation, Keral, date 07.04.2008*

The second round of installations happened in Vietnam in April of 2006 (Øverland 2006). Unlike the Kerala roll out, significant work had been done with regards to installation routines, and a graphical installation package had been created using Installer2Go (Installer2Go 2007; HCMC Installer 2008). The choice of technology had been largely based on the fact that this tool allows use of batch scripts, which the developers were already familiar with, and could do most of the required setup in minimal extra time (Brucker, 2007). This work was used successfully in rolling out DHIS2 to all of Ho Chi Min City, and later to the province of Hue.

In the fall of 2006, two students from the university of Oslo, as part of the course on “Open Source Software Development” featuring DHIS2 as the main example, wrote a set of scripts for creating installation packages based on the Nullsoft Install System (NSIS 2007). However, having merely part of the time of a one-semester-course, this project was never finished to the specification originally set down for the project:
In addition to this, some work was done by another developer in Oslo at about the time this project finished, and in the early part of 2007. Various ways of running DHIS2 from CD's and USB sticks were explored (DHIS2 CD/USB 2008; DHIS2 Linux 2008), as well as solutions for booting a computer from CD/USB and running DHIS2 from there, without any installation needed. Virtualization was also explored cursorily, as well as a windows installation solution based on zip-files and bat-scripts. These possibilities were all documented with code snippets, examples and/or proof of concept modules.

In the early part of 2007, an additional installation project was started in Ethiopia, which was really a continuation of the work done in the course at the University of Oslo:

> In Ethiopia I continued on that with another developer (...) There we were able to create the installer. It worked very well on all the machines that we were testing on it. (...) But it was not put in use
>
> *Mail 2: To me from a student from the University of Oslo involved in the Etheopia project, subject: DHIS2 NSIS installer, 12.04.2008*

So unfortunately, both the work on the installer and the project in Ethiopia more or less ran out in the sand, though once again a working package was created.

The currently last round of implementations, and the main focus of this thesis, was done in Kerala in September/October of 2007, and will be discussed in greater detail in chapter 3. Additional work on installation routines for future projects will also be touched upon.

1.2 Motivation

Setting aside the why of implementing health information systems in themselves, there are several points to be made for working on the how. Having an easy way to get the system up and running should be part of the project plan for any IT system. The less time a new developer spends setting up the system the more time he can spend contributing to further development. Quicker and more fail-safe setup means less hassle for local implementations, and for any OSS project, easy installation is necessary for allowing potential new users to quickly try out the system.

Chapter 1.1 outlines the various efforts at creating installation routines for DHIS2, and while such a quick run-down says little about what the right way to do this would be, it is clear that there are many different ways it can be done, and that a lot of manpower has been spent producing similar solutions to the same problem. Despite this, the project had still not settled on common standard for how the system should be set up or any form of installation procedure. The installation page simply listed a low-level step-by-step guide to setting up the system, with a few notes at the end about the aforementioned projects. Logic dictates that unifying these efforts and standardizing how installation will be performed, and how the system is set up would mean more time for other tasks, both for developers and local implementers. Studies on system development indicate the same, in that implementation and training account for a surprising amount of the total time spent before a system is fully functional, and in almost all situations where DHIS2 has been implemented so far, less time spent on setup can be almost directly converted to more time spent on training, which is an admirable goal in itself.

Results of standard installation routines also include more standardized setup, installations across the globe will have similar properties, paving the way for easier inter-communication within the project. A common platform presents global developers with the fore-knowledge of how a specific implementation would be set up, and answers and solutions for local problems can be produced faster. It also creates clear-cut upgrade-paths, as they will no longer need to be tailored to each and every implementation, and can thus cut significantly down on time spent on upgrades and bug fixes through a standardized upgrade routine. This is imperative for the continued success of an information system, as the process of maintaining a system is shown to account for a large amount of its total total cost of operation, when viewed over a longer period of time.

All of these points fall under the overall motivation of promoting the growth of DHIS2. Several earlier HISP-related articles and theses (e.g. Braa et al (2004), Øverland (2006), Nguyen (2007)) have discussed the importance of health information systems in developing countries, and the opportunity to help increase this coverage was too interesting to pass up.

1.3 Research Questions

This thesis will be focused on information infrastructure growth, through viewing installation

> Create a system for packaging the system for new installations and for doing upgrades of existing installations on Windows and Linux systems. The upgrades should be able to handle changes to the data model.
>
> *Text 1: Installer project specification*

and upgrade routines as part of an overall strategy for an open source Health Information System, using DHIS2 as a specific example. I will not discuss the advantages/disadvantages of DHIS2, nor the merits of health information systems in general, as it is not relevant to the work done in this thesis, and it has been gone over in great detail in earlier HISP writings.

The brunt of my research on the subject comes from an implementation effort in Kerala. Significant additions include work on improving the results for more general use, and modifications to fit within the framework of a WHO-initiative for a national health tool kit, as well as easing the access to interacting projects through common installation procedures.

Analysing the effects this research has had on the growth of the system, and its potential for further influence, will be the overall goal, and to this end I have formulated two research objectives:

First research objective: *How can simplifying installation and upgrade routines benefit the adoption and maintenance of an open source health information system?*

Regarding adoption, I will draw mainly upon the work done after the initial Kerala effort, and how this has impacted the exposure and outreach of the system. For maintenance, my experiences from Kerala and similar from earlier HISP projects will be the primary sources.

Second research objective: *How can standardization and rationalization help Information Infrastructure Growth?*

Here the interesting research is the results of the Kerala work in hindsight, and in particular how these have influenced later work on refining the original work for future use.

1.4 Thesis Structure

This thesis is split into 4 parts:

Part 1:
Literature review – review of literature relevant to the research project which I will draw upon for discussions

Part 2:
Methods – overview of research methods used in this thesis

Part 3:
Empirical study – Overview and recount of the research done

Part 4:
Discussion and Conclusion

Chapter 2 Literature review

This chapter will outline the background literature and theoretical basis for this thesis, which will form the foundation for the empirical study, and discussion of the link between the two.

2.1 Health Information Systems, and Open Source Software

As the example used throughout the thesis is an open source health information system, but there is not much written on this subject apart from literature from the same project, this section will cover both OSS and HIS in more general terms.

2.1.1 Open Source Principles

"Release early, release often" is a term first coined by Eric S. Raymond in his famous article, "The Cathedral and the Bazaar" (Raymond 1999). Concerning the rise and success of the GNU/Linux operating system, Raymond argues in the article that one of many important factors for why Linux gained so rapidly in popularity in the hacker community, was the concept of releasing new versions as often as possible, whenever you had completed something that someone might find useful. This concept has since become important in open source software in general, not just confined to the Linux kernel (Lakhani et al 2002; Madey et al 2002).

When discussing releases, the issue of formats, or how to distribute the software comes up. Continuing the example of the Linux world, most Linux distributions contain a plethora of software packaged with each release, for instance the latest Debian stable release ("etch" at the time of writing), came with over 18200 packages (Etch 2008). This hints at the fact that being part of a distribution is one way of distributing your open source software, but perhaps not the best way for a health information system, which has arguably little value to the average computer user. Erenkrantz (2003) has the following comment:

Ideally, enough formats should be covered so that all users can deal with the release in their preferred packaging format with a minimal degree of hassle.

What «enough» is, is of course a matter of definition, but the choice of how to package the software is a interesting aspect of releasing it, which again is an important part of running an open source project (ibid.).

2.1.2 Health Information Systems

Health Information Systems and their implementation and effect on developing countries has been the focus of quite a few articles, with a considerable amount coming from the HISP project (Mengiste and Nielsen 2006), and while this thesis will not be concerned with their creation and implementation in the way a distinct number of other master thesis's within the project have been, there are a few points worth mentioning.

For instance Braa et al. (2004) discusses the importance of local capacity building with regards to sustainability, how failure to ensure that the health workers at the local level are given sufficient training may lead to implementation failure. This is especially true for an HIS based on open source software:

Technical capacity to deal with OSS development and support is still limited, and lies primarily in the domain of computer scientists working on closer to the machine applications (like operating systems and compilers). When we talk of applications within domains like health or education, there is very limited capacity to deal with OSS technologies. Nguyen (2007)

Similarly, L. H. Øverland (2007) shows how the creation of local communities can be of benefit to a global HIS project. While it is of course not straight forward, the idea of taking solutions to local problems and local extensions, and generalizing them to make them palatable for inclusion in a global solution, means each additional local community represents possible new development.

Like most software systems, a HIS will sooner or later require updates, be it to fix bugs, encompass new features or ensure compatibility with new software solutions. Regarding updates to an installed HIS, a representative from a Dutch company that produces a HIS for use in the Dutch health sector had the following comment:

"...the health-care application domain is very conservative when it comes to managing ICT. As software changes might interfere with day-to-day operations and government regulation change regularly requiring software changes anyway, other changes are usually undesirable. Regular small change to the software are all that is needed and desired." (Ballintijn 2006)

A glaring difference between this context and the situation usually present where DHIS2 is concerned is of course that the there is a great deal more resources available in a developed country in comparison to the general case of developing countries. Additionally it does not quite fit with the general open source principles outlined in the previous chapter. Still, there are arguments to be made for this viewpoint, so it merits further discussion.

2.1.3 Adoption of Open Source Software

Perhaps the most important reason why having regular releases of your software package, delivered in formats easy to use for the end-user, is important, would be the human tendency towards group mentality. Hanseth and Aanestad (2003) discusses the concept of «critical mass» with regards to network growth:

"Critical mass" is a concept often employed in relation to these challenges. A common strategy in line with this concept is to identify and subsidize a number of users willing to adopt the technology. If this number of users is high enough, the network will start to grow by itself. But this strategy only works where there is an agent who may do this subsidizing. This is not necessarily the case with regard to telemedicine. (Hanseth, Aanestad 2003)

I.e. there's a certain number of users needed before a network has its own inertia, when it has the ability to survive on its own, and attract new users on basis of itself. In a later article, Hanseth (2004) comments how this is relevant to more general principles than just networks:

Nobody will join the new paradigm until it already has a significant number of other members - the 'critical mass'. (Hanseth 2004)

For any form of system or network, reaching this critical mass is essential for its survival (ibid.)

There are also arguments to be made for how this may translate to open source software in general. The more people using an open source package in practice, the higher the installed base it will have (see chapter 2.4), and the more interesting it will be to new users. An appropriate analogy could be how buying the same type of home stereo equipment as a friend of yours will let you share experiences and ideas. In the same way, an existing large user base can make adoption of a new piece of software less daunting, as it will be perceived as easier to get help or support if necessary. Or, in other words, the value of an open source project is proportional to its users (Bonaccorsi and Rossi 2003).

Additionally, there's a distinct advantage to open source developers from enlarging the user base of their software. There's the obvious ones of making it more interesting for potential new developers, more testing leading to more bugs found, cf. Raymond (1998) and what he defines as Linus's Law: “given enough eyeballs, all bugs are shallow”. Finally, it goes back to the mentioned of how people will tend towards the software with the largest user-group, since that is what will in general have the most value (Hanseth 2004)

2.2 Boundary Objects

Boundary objects is an interesting way of viewing objects or concepts that has differing meanings for different people (C. P. Lee 2007). The term was originally coined by Star and Griesemer (1989) in a well-known article on divergent viewpoints between the many involved actors at the Museum of Vertebrate Zoology. It has later been used to describe behaviour in a great many different contexts (Gal et al, 2004; C. P. Lee 2007).

2.2.1 Boundary Objects as Translators

Star and Griesemer (1989) defined boundary objects as:

In particular, we are interested in the kinds of translations scientists perform in order to craft objects containing elements which are different in different worlds - objects marginal to those worlds, or what we call boundary objects.

This is not to say that that any object which two people will view differently can be defined as a boundary object, but rather how an object can, possibly temporary, act as an anchor or a bridge between understandings (Star, Griesemer 1989, p414). These objects have several properties that enable them to serve as translators between different groups. They exist in different social worlds and can have different meanings in each, while still having enough in common to serve their role as translation devices (Gal et al, 2004).

Harvey and Chrisman (1998) wrote an interesting article on geographical information system (GIS) technology, the connection between it and people, and how they connect different social groups. For describing this connection they use the concept of boundary objects:

Boundary objects mediate between different groups; they do not provide a common understanding, or consensus between participants. Instead, they serve a dual function similar to that of geographic boundaries; at the same time as they serve to distinguish differences, they also supply common points of reference. A key difference is the dynamics. Whereas geographic boundaries seem over time to become relatively solid anchors for social relationships between groups, technological boundary objects remain subject to change. (Harvey and Chrisman 1998, p1686)

Geographical information systems are just one example of an information system that works like a boundary object though. Pawlowski et al (2000) argue in an article about shared information systems, how information systems in general can be viewed as boundary objects between communities of practice. In their example it is different parts of a company, but the concept is easily expandable to any organization where the common object is an information system.

2.2.2 Methods Standardization

In addition to the concept of boundary objects, Star and Griesemer (1989, p387) argue that there exists a second important activity for translating between viewpoints, namely standardization of methods. This because methods standardization not only makes information compatible between groups of people, but also because it allows for a longer reach across divergent worlds (Star, Griesemer 1989, p407). And, perhaps most relevant to this thesis:

Because the natural history work took place at highly distributed sites by a number of *different people, standardized methods were essential* (Star, Griesemer 1989, p411)

Standardized formats let amateur zoologists report findings in a way that made them palatable to professionals so there could be actual cooperation, and let them both save time on how to enter and read data, benefits that are common among many industries (Farrell and Saloner, 1986). This translates over to Information Infrastructure as well, in how it cannot exist unless its respective members can communicate in a standardized fashion (Hanseth et al 1996; Hanseth 2004). The importance of standards in communication is something we've also seen regarding DHIS2, and much the same thing mentioned by Harvey and Chrisman (1998, p1683),

Boundary objects provide coherence by linking multiple social groups through the stabilizations of facts and artefacts

and it is further commented on by C. P. Lee (2007):

While Star notes that this list is by no means exhaustive, it is interesting to note that two of the four types of boundary objects listed have standardization as a key component. Repositories are indexed in a standardized fashion and standardized forms are standardized indexes. Furthermore, it could be argued that political boundaries or atlases also relay on standardized forms of both measurement and representation. This is particularly interesting given that methods control and boundary objects were said to be two different strategies for cooperation across social worlds. Standardization is integral to the definition of boundary objects.

This article is also critical to how the concept of boundary objects has been used for so much since the original article by Star and Griesemer, that perhaps the term has been applied uncritically in many cases. She further cites how this has led to cases of boundary objects "blamed" for failure to satisfy informational requirements (p311), where perhaps the failure is in the definition of the object itself.

2.2.3 Boundary Negotiating Artefacts

Through the aforementioned chain of reasoning, C. P. Lee (2007), whose work will be the background of this entire sub-chapter, claims that it is important to differentiate between boundary objects, and what she defines as *Boundary Negotiating Artefacts* That is, artefacts that help coordinate communities and foster communication and cooperation, but which do not fit the definition of a boundary object. These she further separates into 5 sub-categories

1) Self-explanation
2) Inclusion
3) Compilation
4) Structuring
5) Borrowing

Of these, self-explanation artefacts are the ones with the least in common with boundary objects, in that they are artefacts for use by 1-3 individuals within a community of practice, and as such will not cross boundaries, but may enlarge the boundaries of the community to which it belongs (ibid.)

Inclusion artefacts may be seen as the evolution of self-explanation artefacts, in the sense that they are the ones deemed interesting enough to be presented to other communities. In a sense they can be viewed as the first bridge between communities of practice.

Compilation artefacts are arrangements, objects or constructs specifically designed to ensure a common understanding of a problem or situation between two communities of practice. They may or may not include self-explanation artefacts, and they may perfectly well be temporary, but the key point is the creation of a shared agreement on a given issue.

Like compilation artefacts, structuring artefacts are objects essential for communication and sharing knowledge, but with the important distinction that they are not temporary, and they also serve as direction and coordination of activity in the distinct communities. In many ways, they can serve as guidelines for binding all communities involved in a project together, and as such may be the focal point of discussion and conflicts therein. The latter, and this is of course the most interesting point with regards to this thesis, can lead to the negotiation and moving of boundaries.

Finally, borrowed artefacts are artefacts taken from one community of practice, and used in another one in ways not originally intended by its creator(s).

To avoid too much paraphrasing when summarizing the concepts defining a boundary negotiating artefact, here is a summation as written by C. P. Lee (2007):

In summary, boundary negotiating artefacts:

- Are surrounded by sets of practices that may or may not be agreed upon by participants;
- Facilitate the crossing of boundaries (transmitting information);
- Facilitate the pushing and establishing of boundaries (dividing labour);
- May seem ''effortful'' in use as opposed to effortless;
- Are fluid: (1) a boundary negotiating artefact can change from one type to another when the context of use changes; and (2) a boundary negotiating artefact can sometimes also simultaneously be physically incorporated or transformed into another artefact;
- Can be largely sufficient for collaboration;
- Are possible predecessors of boundary objects.

In other words, there are objects that move between groups or communities of practice that may not quite fit a stricter definition of boundary objects. These objects will however still exist in the interface between social worlds, and the idea of objects crossing and moving boundaries is central to the work done in this thesis.

2.2.4 Intermediary and Frontier Objects

As a bit more recent idea than Boundary Objects, Intermediary Objects has become a popular concept for aiding in the understanding of design process. The original definition by Boujut and Laureillard (2002) reads as follows:

Intermediary object is a general category embracing all types of artefacts, whether physical (plans, mock-ups, sketches, etc.) or virtual (CAD models, calculation results, etc.), produced by the participants during their work. In other words, it covers all kinds of externalisation.

Boujut and Blanco (2003) further elaborate on the features usually applied to IOs:

Intermediary objects have three main features:

• *mediation,*

• *transformation or translation,*

• *representation.*

The main differentiation of IOs from BOs is how IOs are centred around being a representation of the intermediate states of the product (ibid.) Finally, IOs are usually found in connection with industrial design, but can also be used to explain the design and development process of computer software (Strandli 2008).

However, as we further examine the theories concerning boundary objects, boundary negotiating artefacts, intermediary objects and the issues facing communities of practice when coordinating work on a project, it is a bit difficult to find exactly where the study and practice done in this master thesis fits in. Because of this, I would like to coin the term “Frontier Object” to discuss the work done in this thesis, and propose it as a theoretical framework to discuss my research questions in light of the empirical data produced (chapter 4). The concept of FO will be discussed in detail in chapter 5.3.

2.3 Communities of Practice

Communities of practice is a concept first introduced by Lave and Wenger in 1991. Concerning how social groups have common knowledge and practices that are learned and transmitted within the community, simply through taking part in the activities, possibly information that cannot be gained elsewhere, the theory has become immensely popular. And while the theory is fairly new, the concept it describes is as old as man, in that as long as we as a race have been forming communities (i.e. technically as long as we can define the human race), there has existed a common learning within a group (Wenger 2000)

Kimble et al (2001) wrote an article about how these concepts translate to a virtual world, where the communities can be spread over great geographical distances, and where contact is rarely direct but instead mostly virtual. Combined with the idea of separating knowledge into hard and soft components (ibid.) – where hard knowledge is made up of knowledge that is written down and clearly defined, and soft knowledge is the unwritten knowledge, the things you learn while participating in the community – there are interesting implications for how CoP-theory works in these situations Especially how sharing of soft knowledge, one of the cornerstones of any CoP, works without physical contact, as the latter is seen to be essential to high-complexity collaboration (Hemetsberger, Reinhardt 2004).

In the context of this thesis, the theory of communities of practice is mainly of interest in how it relates to the aforementioned theory of boundary objects:

Boundary objects are described as objects that coordinate the perspectives of various communities of practice (Lee 2005)

In other words the social groups boundary objects are often defined through, in that the interaction of an object between two is what defines the object as being boundary object, can just as easily be viewed as communities of practice. This has the interesting consequence of showing how these boundary objects, boundary negotiating artefacts or frontier objects may not only have different meanings, but their impact can provide a different learning experience to each of the communities in question. Additionally, studies indicate that these shared artefacts are crucial to the sharing of knowledge, especially in the distributed case mentioned above (Kimble et al 2001)

Another important thing to note when discussing communities of practice, is how they cannot exist without having boundaries between them. These need not be clearly defined, and indeed in almost all cases they are implicit, rather they can easily be fluid and susceptible to change. Nor do the communities in question need to be clearly defined, or physically close (Kimble et al 2001). However, these boundaries, or interfaces between communities, will quite often be the location for exchanges of information and experience, leading to interesting new insight (Wenger 2000).

2.4 Installed Base

Installed base refers to the existing base of installed hardware and software, personnel skills and know-how. In business it can also mean the number of a system in use, as opposed to market share, which give different indications of popularity. There's also an idea of self-reinforcement, in that given two products, the one with the largest installed base is what end-users will generally prefer (Rolland 2000) In network theory the concept can be expanded to include users, in the sense that a bigger installed base in the form of people using a network, will mean the network has greater value (Rolland 2000; Hanseth and Aanes 2003; Farrell and Saloner, 1986).

In an article on how to scale health information systems in developing countries, Mengiste and Nielsen (2006) comment on the importance of being aware of the challenges of interaction with the installed base when designing information infrastructure:

Information systems are not built from scratch, but build upon and necessarily relates to existing information systems, communication networks, structures such as institutions, user practices, human resources etc. Therefore, rather than being occupied with developing independent information systems, the challenges of information infrastructure builders are related to the complexity of openness, heterogeneity and what already exists: the installed base. (Mengiste, Nielsen 2006)

Continuing this thought, the article focuses on how the success of implementing a health information system is dependant on the interaction with the installed base, which is not set in stone, but can be shaped with the help of the human resources who are involved (ibid.). Furthermore, they stress the importance of ensuring the information system in question is installed base «friendly», as opposed to installed base "hostile", meaning understanding and making the best out of what already exists (ibid.)

In the above definition of installed base, personnel skills and «know-how» was included as part of the concept. The thinking here is that when you consider the complete package that exists in the location where you are interested in e.g. implementing your software, you need to consider what the people who will be in contact with the system know. For example, if they have in-depth knowledge of an existing system, playing to this strength in ensuring there are similarities can ensure quicker adoption and/or acceptance. Hanseth (2004) touches upon this subject, referring to it as "Knowledge as Infrastructure":

Knowledge could be said to be related to a materially heterogeneous 'installed base' that has a certain transparency, a taken-for-grantedness, that works as a fundamental, shared resource for the community relating to it. Such a view of knowledge provides striking similarities with common definitions of information infrastructures (Hanseth 2004)

The article also has some interesting views on standards and standardization (as previously mentioned) and how these are key issues for changing or expanding on the installed base. Three different scenarios for modifying the installed base is outlined, through extensions, i.e. adding on top of what already exists, through moderate changes that can be incorporated with a gateway or boundary object acting as a translator between versions, or through completely scrapping the old standard and setting up a new one. Here the last is of course the most radical one, and thus the hardest sell, since it does not leverage the existing installed base in the way the first two do (ibid.). Another way to look at this connection:

the value of a standard is the number of adopters in its installed base. (Hanseth 2004)

Meaning that a standard that is backed up by a great amount of users and a big installed base will be difficult to change, even if for the better, or dislodge/replace for a competing standard. The inertia of existing knowledge and implementations is a great hurdle to overcome.

Chapter 3 Methods

This chapter will outline and explain the research methods relevant to this thesis, as well as their usefulness and application in my research.

3.1 Research Methods

This section will outline the research methods relevant to the research performed as basis for this thesis.

3.1.1 Action Research

Action Research is a well established research method, originally conceived in the 1950s. Unlike more traditional research methods, there is no concept of the detached observer, the impartial and impassive researcher unwilling to interfere in the object(s) of the research. Instead the researcher is part of the project of study, observing as well as participating, i.e. the method combines practical problem solving with research.

The process of Action Research is cyclical, through five phases, as described by Baskervill (1999):

The most prevalent action research description (Susman and Evered, 1978) details a five phase, cyclical process. The approach first requires the establishment of a client-system infrastructure or research environment. Then, five identifiable phases are iterated:

1. *diagnosing,*
2. *action planning,*
3. *action taking,*
4. *evaluating and*
5. *specifying learning.*

This process is also illustrated in Illustration 1 below:

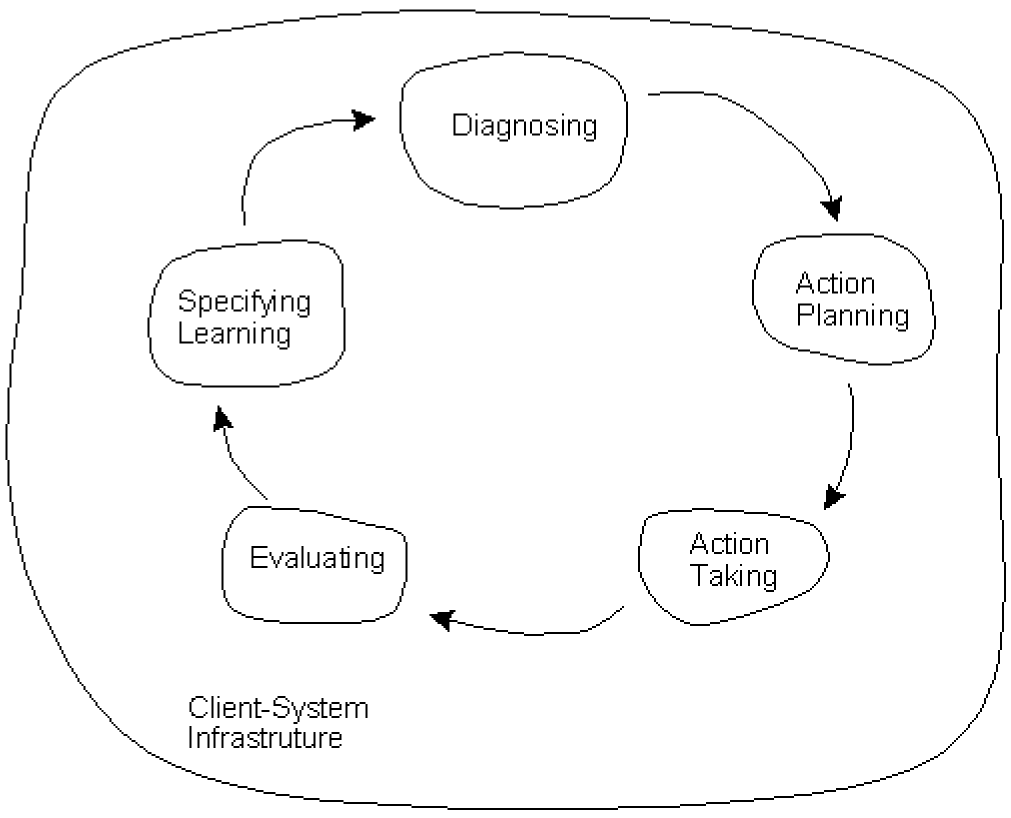


Illustration 1: Action Research Cycle (Baskervill 1999)

As action research is based on practical work, solving a specific problem, implementing a given solution, it produces highly relevant results (Baskervill 1999). Other beneficial characteristics includes (Baskervill and Wood-Harper 2002):

1. *The researcher is actively involved, with expected benefits for both researcher and organization*
2. *The knowledge obtained can be immediately applied. There is not the sense of the detached observer, but that of an active participant wishing to utilize any new knowledge based on an explicit, clear conceptual framework*
3. *The research is a cyclical process linking theory and practice*

Of course, action research is not problem free, and critique of the method includes:

- lack of impartiality

- lack of rigour
- context-bound

The first issue is easy to visualize, as the researcher will due to the nature of action research become heavily invested in the research project, which can potentially make it difficult to keep a completely objective view of the knowledge and learning produced. (Benbasat et al 1987; Baskervill and Wood-Harper 2002)

The second is partly rooted in the same issue as the first, in that heavy involvement in the practical side of the research project may leave the theoretical part suffering. If the researcher is too preoccupied with solving a specific problem, too little time will be spent on the actual research that should be at work in parallel. (Baskervill and Wood-Harper 2002)

Taken too far, either of the first two issues may lead to what has been disparagingly referred to as "consulting masquerading as research", i.e. that what is happening is not really research, but merely problem solving (Avison et al 1999; Baskervill and Wood-Harper 2002).

Finally, as action research is meant to "solve" and do research upon one specific situation, there are issues with the research being very tied to the context in question, and it can be hard to generalize the research (Baskervill and Wood-Harper 2002 ; Benbasat et al 1987).

3.1.2 Action Research and Information Systems

In the context of Information Systems, action research has become an important research method in the 1990s (Baskervill 1999). Despite having taken some time to gain acceptance, action research is now viewed as a good methodology for investigating information systems:

By emphasizing collaboration between researchers and practitioners, action research would seem to represent an ideal research method for information systems. Such systems represent an applied discipline, and the related research is often justified in terms of its implications for practice. (Avison et al 1999)

In the case of HISP-related action research, focus has often been on the sustainability of IS implementations. For example, Braa et al (2004) discuss how donor-funded information systems will often fail to persist without proper training in usage and maintenance of the system for the local team responsible after investors finish implementation and pull out. Or in the words of Fossum (2007):

Transfer of appropriate knowledge must be embedded in the alignment processes, because local expert groups will be responsible for continuing maintenance and development when the researchers have left.

Since the matter of training local groups is so crucial to sustainability and persistence of IS, action research is a fine choice of research method, as the collaboration between researcher and participants grants great opportunities for this dissemination of knowledge.

3.1.3 Case Study

Case study as a research method has been around for quite some time. One definition of the method is the following from Benbasat et al (1987):

A case study examines a phenomenon in its natural setting, employing multiple methods of data collection to gather information from one or a few entities (people, groups, or organizations).

The article also discusses reasons why case study is a viable strategy for doing information systems research, citing both the ability to study a problem in a natural setting and that it gives the researcher the ability to answer “the how and why”, as well as perhaps the most relevant for this thesis, that it provides the researcher with the tools required to study an area where little research has been done before (ibid.).

Case study is classified as a qualitative research method, alongside other methods often used in information systems research, such as ethnographic research, grounded theory and the aforementioned action research (Dubé and Paré 2001). One of the common key aspects of these methods, and thus a key aspect of case study research, are the data collection methods used, including interviews, documentation, observations, and questionnaires, resulting in qualitative and/or quantitative data (ibid.).

Avison et al (1999) has some interesting comment on action research versus case study, commenting that in many cases they can be quite close but for important details:

Alternatively, interviewing and observing people in these situations without the insight associated with intervention is also not action research. It might be described instead as case study research. Such research frequently reports what practitioners say they do. In action research, the emphasis is more on what practitioners actually do.

3.2 Research Approach

The following section will describe how and why I employed the outlined research methods for this case.

3.2.1 The Research Project

A lot has been written about the organisation around the DHIS2 project, for instance most earlier HISP-related master theses have covered the subject, e.g. Fossum (2007), Øverland (2006), Nordal (2006), so I will merely cover my role and involvement.

I first came in touch with HISP when I took the aforementioned course on open source projects, and after my project group was one of the few to produce a usable result, I was approached on the subject of writing my master thesis on research related to DHIS2. Thrilled at the idea of doing research that would hopefully be immediately useful, I accepted and started my research early August of 2007.

The following is a rough time line of the research done:

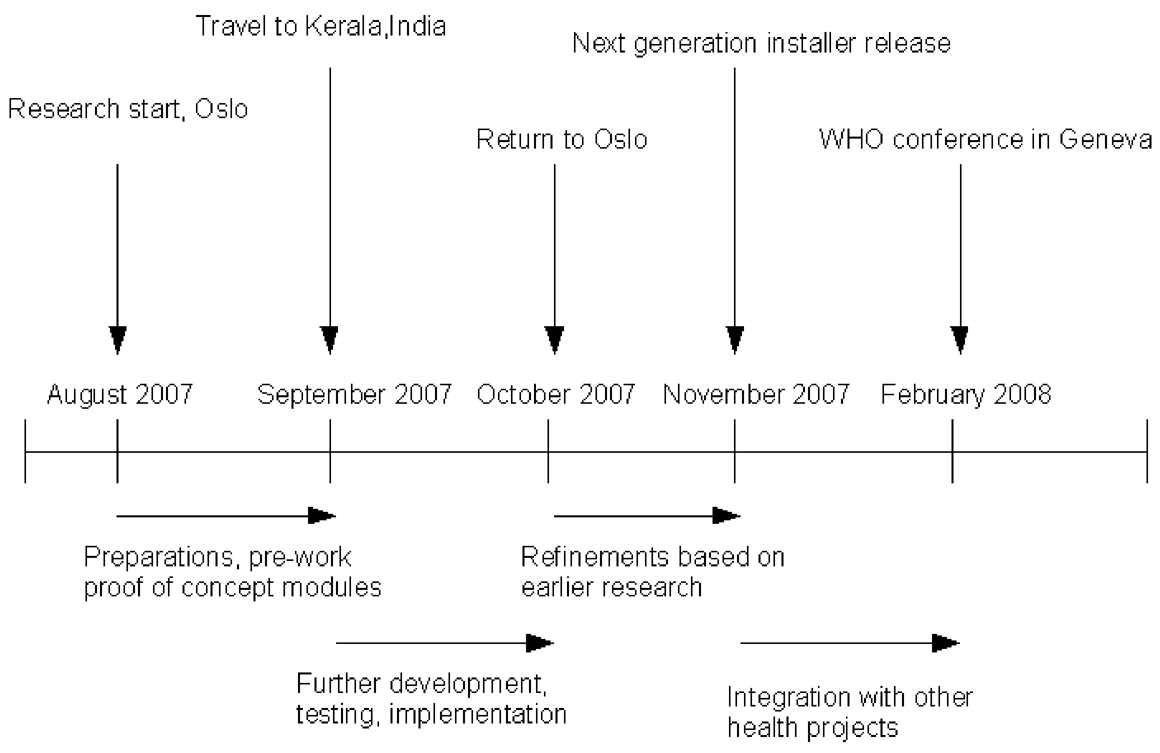

Travel to Kerala,India
Next generation installer release
Research start, Oslo
Return to Oslo
WHO conference in Geneva
August 2007
September 2007
October 2007
November 2007
February 2008
Preparations, pre-work proof of concept modules
Refinements based on earlier research
Further development, testing, implementation
Integration with other health projects

Illustration 2: Research timeline

The first part of this research was conducted at the university of Oslo, mostly alone, but with the occasional meeting with other HISP developers for input and feedback.

The second and most important part of the research took place in Trivandrum, the state capital of Kerala, which is a state in south-western India. This trip was done together with another master student at the university of Oslo, who was doing different HISP-related work, and proved to be a valuable help for discussions and ideas, and vice versa.

The third part was conducted back at the university again, and culminated in an invitation to attend a small conference at the WHO in Geneva, as they were interested in DHIS2. I was asked to come along in light of the research I had done, since the focus on the early part of the conference would be on installation, packages etc.

3.2.2 The Action Research Cycle in Practice

The image below covers the action research cycle in the first two parts of the research project. It is basically the same as the one depicted in Illustration 1, but with the steps marked with what was done in the different phases of the research, illustrating how the action research progressed:

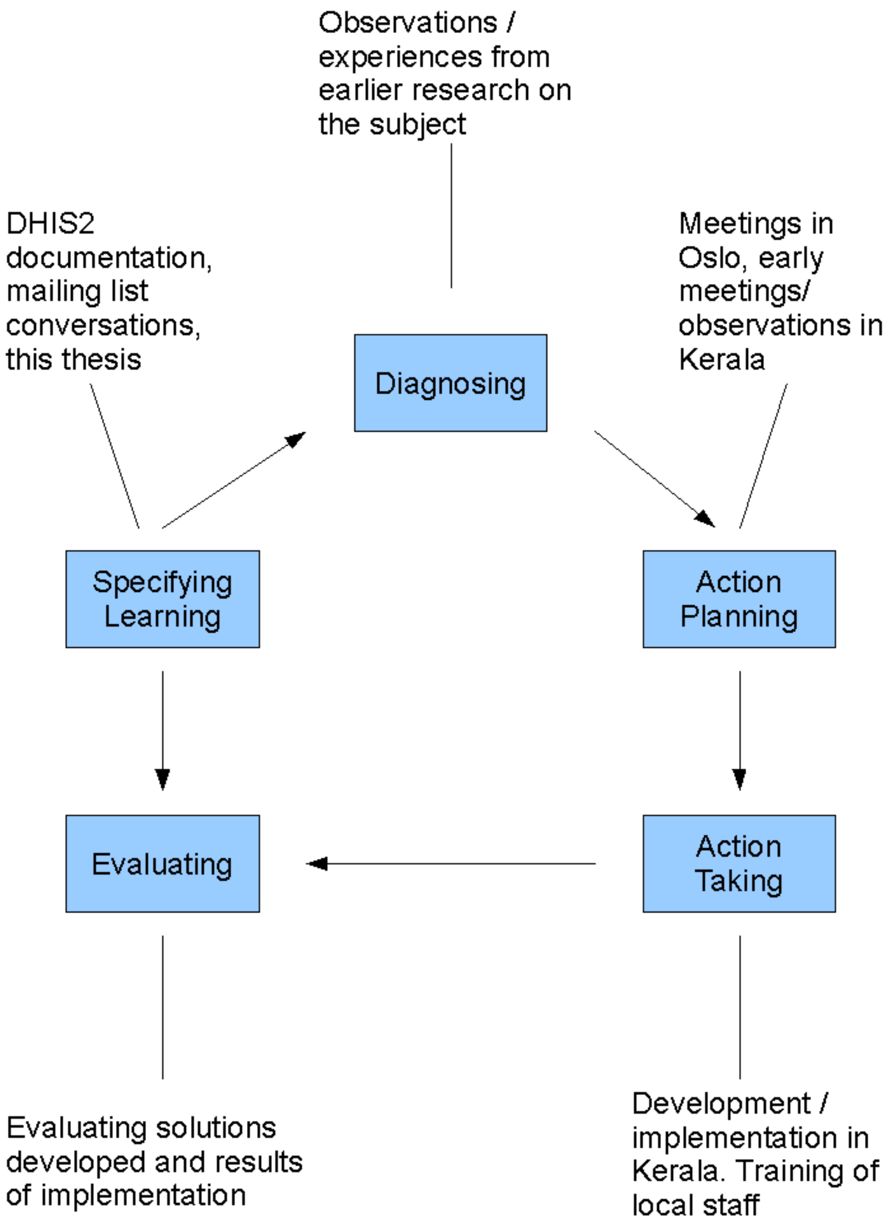

Observations / experiences from earlier research on the subject
DHIS2 documentation, mailing list conversations, this thesis
Meetings in Oslo, early meetings/ observations in Kerala
Diagnosing
Specifying Learning
Action Planning
Evaluating
Action Taking
Evaluating solutions developed and results of implementation
Development / implementation in Kerala. Training of local staff

Illustration 3: First Action Research Cycle

This is of course slightly simplified, there were several sub-process going on in a similar cyclic fashion. For instance, the process of testing involved quite a few trips to clinics, testing the project in practice, and then going back to the office to refine and fix issues before more rounds of testing. However, it fits the overall shape of the research well.

To further underline how the cyclic nature of the action research project, the following graph details the work done in the third part:

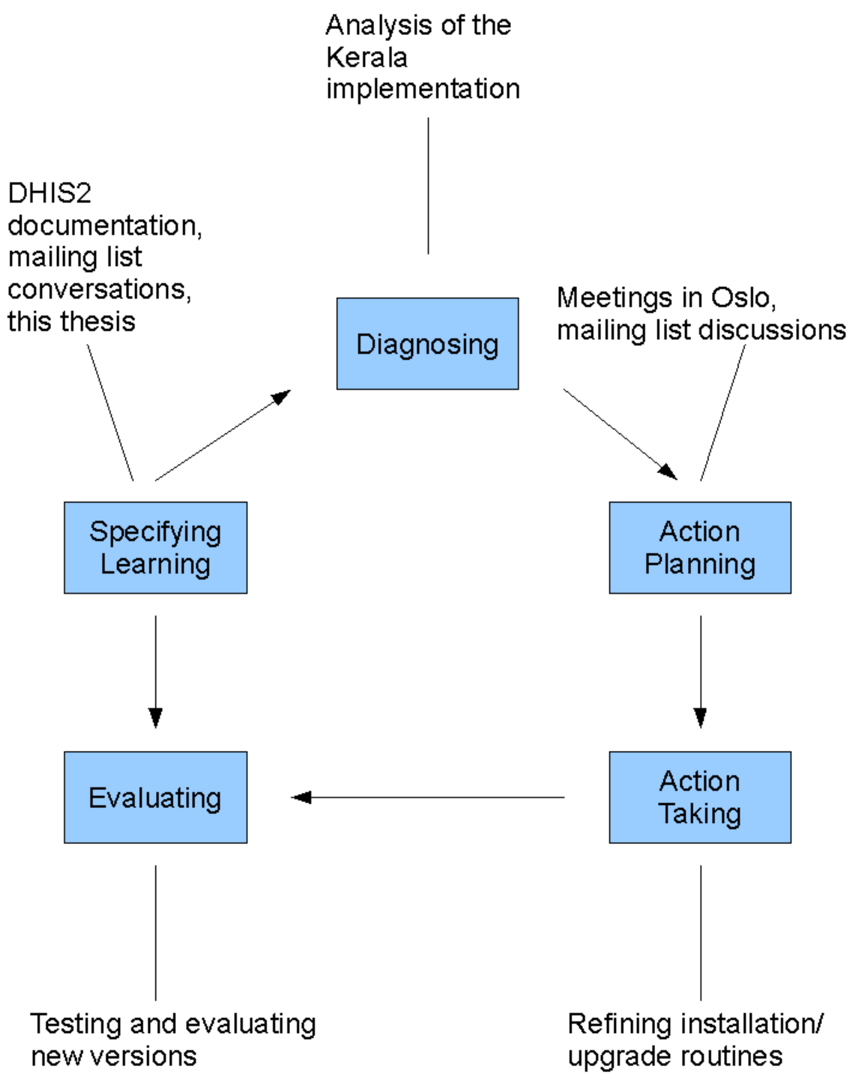

Analysis of the Kerala implementation
DHIS2 documentation, mailing list conversations, this thesis
Diagnosing
Meetings in Oslo, mailing list discussions
Specifying Learning
Action Planning
Evaluating
Action Taking
Testing and evaluating new versions
Refining installation/ upgrade routines

Illustration 4: Second Action Research Cycle

Again, this does not cover everything, but outlines how the research progressed.

3.2.3 Data Collection Methods

Following are the methods employed for collecting data for the research project.

Interviews:

Preece et al (2007) define interviews as belonging to one of four main types:

- open-ended or unstructured
- structured
- semi-structured
- group interviews.

Here the fourth deals more with how to interview several people at once, while the first three concern how the interview is structured (ibid).

In the case of my research there were plenty of situations where I needed to get information about something out of people, and thus interviews became one of my most important methods for data collection. However, these situations, be it during my pre-work in August or in the field in Kerala, were usually involving other project-members, and so the interviews conducted were informal and unstructured, as it would have been strange to perform rigid, formal interviews of people one otherwise work together and socialize with. In fact, most of these interviews took the form of quick on-the-spot questions with nothing written down apart from a few notes in my field diary, fitting perfectly within the category of unstructured interviews, which can also be viewed as conversations around a particular topic (ibid). Perhaps the only times I did more structured questioning was when I contacted other developers through email, as the medium then required more rigidity by default.

Observations:

Though action research does not employ observations as a data collection method in the same way more traditional research methods do – as mentioned the action researcher is the participant, not the passive observer – it was still an important part of the work. Observation can be either direct, in that users are observed directly as they are working on / testing the system, or indirect in that one observes records or results (ibid.).

Testing:

As my research was focused on solving a particular problem, and thus produced results that were applicable to the context, continuous testing was essential. From simple tests and proof of concept runs on my own laptop, to more rigid testing on the computers in the clinics in Kerala, testing was essential both to shake out bugs and for research material.

This process was also combined with observation, as watching facilitators and developers from HISP India performing testing produced valuable insight, not only on how well it worked, but how it was perceived by the people who would be using it.

Training:

Unlike most other research projects related to DHIS2, mine did not cover much training. The people I were in contact with have all extensive knowledge of the system already, and so training needs were limited to how to use the installation and shorter lectures on how it was set up. Nevertheless, these small sessions were an extra source for data, as both the need to explain and outline what I had done, and especially the feedback I got, was very relevant to the research.

Documentation analysis:

While there was not much documentation written or available on subjects relevant for this research, valuable contributions was still made by the documentation written about similar earlier projects, as mentioned in chapter 1.1. Analysing the how, what and why of these attempts produced insight into several elements of my action research.

Field notes, laptop:

Last but not least, quite a bit of the data collected came in the form of notes during development, testing, implementation etc., either in my field diary, or jotted down on my laptop. The latter of which was also and important help, as some parts of the research required more tinkering at low levels than what is practical or feasible with computers available at the university or the HISP Kerala offices.

Chapter 4 Empirical Study

Illustration 5: Presentation in progress

When viewing the research objectives stated in chapter 1.3, they may seem slightly broad, which is part of their appeal. Not much has been written on the subject, and DHIS2 was an excellent project for research, especially, as the following quote will show, as it had been of interest to the project for some time:

> We find an example of the fluidity of the DHIS 2.0 unconventional and quite representative. Many problems have been encountered to install and use the reporting software in peripheral clinics, where computers' maintenance and skills are rare. The lack of Internet connections (and of continuous electricity supply, often) does not allow to use the DHIS 2.0 online (although designed also for that). Installing and maintaining locally all the programs needed revealed to be a hard, and often unmanageable task. Viruses, present on CDs and USB memories, spread after each formatting, and anti virus updates are not easily accessible. Machines' configurations proved difficult for most users. FOSS allows to redesign the whole set of required programs to be run -on a thin version of GNU/Linux- from bootable CDs, which can skip the above problems of viruses and configurations. This would have been impossible if the software was not based on FOSS.
>
> *Mail 3: To me from HISP developers, excerpt from a draft to an article on FOSS, DHIS2 and viruses, date: 13.05.2008*

I was tasked with an Action Research project for delving into these issues, with an initial focus on combining DHIS2 with Free Open Source Software, and specifically on solutions for running a combination from bootable media (CDs mainly, but also USB memory sticks etc.). However, the DHIS2 project is dynamic and in constant change, and in the year of research, the work done is not the same as initially prospected. Soon after my research started for full, I was asked to assist in a local implementation in Kerala, India, by researching install routines[1] for a round of initial installation. After completion of this field study, the results of our research was refined and the local solution made more generic for further use in the global project. The following chapter will describe the empirical study done throughout all of this, in a mostly chronological fashion for clarity.

The first sub chapter can be viewed as an introduction to DHIS2, as an overview of the system is necessary to understand both the how and why of the empirical study done. The second chapter outlines the work done to allow DHIS2 to run from bootable CDs, including an outline of the possibilities in this field. Chapter 4.1.3 explains a local implementation effort

1 Throughout this chapter, installation routine, installer and installation package will be used interchangeably

in Kerala, India, including further customisation of the previous work, while chapters 4.1.4 and 4.1.5 covers how the local solution is scaled up to a generic level for the global project, and the impact this has had on the projects connection to international organisations.

4.1 The DHIS2 Software Package

To explain most of the technical work done in this thesis, it is necessary with an overview of how DHIS2 works. DHIS2 is not a stand-alone application, it is dependent on underlying technologies, what we have termed the DHIS2 Software Package, the details of which will be outlined in the following chapter.

DHIS2 is a web application written in Java, using a database back end for data storage, distributed as a Web archive or war file (Sun War 2008). As such, the following is required for running it:

- a web application server
- a database server
- a Java runtime environment
- a web browser (technically not needed for running the system, but for using it)

These are the functional requirements, but for building packages available for free use and distribution within the project – CDs, Installers and the like – the licensing of each piece of software making up the package is also of great importance. The following two sub chapters will outline the license requirements and the alternatives.

4.1.1 Licensing

While the technical difficulties of putting together a package that could run DHIS2 was interesting, equally important was ensuring that said package was completely legal for our purpose, in other words, all components needed to allow for free redistribution.

DHIS2 itself is open source software (OSS 2008), likewise with all frameworks used. So there was a strong wish for basing the complete package on OSS as well, similar to what had been done in earlier implementations. The licenses used by the components covered in chapter

4.1.2 – Mozilla Public License (MPL 2008), GNU General Public License (GPL 2008), Apache Software License (ASL 2008), the BSD license (BSD 2008) – all allow redistribution with light requirements, most of which were easily covered by including the license text in our package. To exemplify, the most relevant part of the Apache Software License, Version 2.0 (ASL 2008):

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1) *You must give any other recipients of the Work or Derivative Works a copy of this License; and*

The issue of a Java Runtime Environment was the only one requiring a non OSS solution. Since this version of Java we were basing our work on (DHIS2 at the time was not functional with Java version 1.6, so we used Java 1.5) was proprietary software, there was a much higher restriction on redistribution than the other components mentioned, but it was allowed under certain conditions (Java Redist 2007). Careful examination however, indicated that DHIS2 fulfilled all of these, filling the last requirements for a re distributable complete DHIS2 package.

4.1.2 Software Alternatives

For all the following application layers, the choices had been narrowed down significantly for the research project, by simple fact of what was in common use by the developers and users already. Implementing a solution using software used by no other project members would deprive the research project of any prior knowledge and experience, so there would have to be strong reasons to do so, which was not found to be the case for any part of the package.

Web Application Server:

For the web application server, there are basically two technologies that have been used by the DHIS2 project, Tomcat (Tomcat 2007) and Jetty (Jetty 2007). Other alternatives include Resin (Resin 2008) and JBoss (JBoss 2008), but the pervasiveness of the first two in the the DHIS2 project in general, lead to an extremely brief consideration of other alternatives before their outright dismissal.

Among DHIS2 developers, Jetty is the most commonly used web application server, due mainly to its great integration with the build system used, Maven (Maven 2008), as the Maven Jetty Plugin (Maven Jetty Plugin 2008), provides a build target for running the DHIS2 web application directly. Due mainly to this fact, and also because the differences between the two are otherwise small, Jetty was the choice we originally worked with, and part of the first version of the DHIS2 Live CD covered in chapter 4.2. However, due to circumstances covered in chapter 4.3, Tomcat was the choice in the end for subsequent installation packages.

Finally, Tomcat and Jetty are both open source web servers written in Java, both licensed under the Apache Software License (ASL 2008).

Database Server:

DHIS2 can be run in persistent and non-persistent “modes”. For the latter, an in-memory database HSQLDB (HSQLDB 2008) is used, and no data will be stored between sessions. The former and more useful mode, and the main focus for the rest of this chapter, requires a database back end.

As DHIS2 uses Hibernate (Hibernate 2007), the choice of database server is completely transparent. Through simply changing three lines in a configuration file, you can switch out your database implementation, and Hibernate supports a wide array of database servers (Hibernate2 2007). However, most of these are commercial solutions, and not suitable for use with DHIS2, and again, common usage among developers and users of DHIS2 narrowed the choice down to two alternatives, MySQL (MySQL 2007) and PostgreSQL (PostgreSQL 2007). Both of these are open source, GPL and BSD licensed respectively.

From our point of view, the differences between the two are small, and ironically more or less directly coincides with how the projects describe their products. The MySQL project advertises their database as “The world's most popular open source database”, while the PostgreSQL caption is “The world's most advanced open source database”. In the DHIS2 project MySQL has indeed been the popular choice among both core developers and for example local developers in HISP India, the latter which of course played a great part in our choices in that specific case, see chapter 4.3. On the other hand, PostgreSQL offers some interesting advanced possibilities, which was of especial interest when coordinating with other projects (see chapter 4.4.3).

In the end, this choice was not as clear cut as the web application server, as the project wanted to support both solutions, and due to how upgrading DHIS2 works, the database choice has had more of an impact on the research done, and both choices have been used to some extent. MySQL for both the DHIS2 Live CD, and the implementation in Kerala, covered in chapter 4.2 and 4.3 respectively, and PostgreSQL for some of the later work, integration with other open source projects in particular, see chapter 4.4 and 4.5.

Java Runtime Environment:

Without a doubt the easiest choice to make, there was simply no alternatives to consider. Version 1.5.0 was needed for running DHIS2, and since space was not so much an issue in any of the cases of the research, the complete Java Development Kit was chosen, just in case it would be needed. Prior experience indicated that some parts of DHIS2 had at one point needed the complete JDK, so this was the simple choice to minimize possible complications.

Web Browser:

Currently almost any modern web browser work with DHIS2, including, but not limited to, the following:

- Internet Explorer 7.x
- Opera 9.x
- Firefox 2.x
- Safari 3.x
- SeaMonkey 1.1.x
- Konqueror 3.5.x

However, it is developed and tested with Firefox and this is thus the only officially supported browser. Additionally, only Firefox, SeaMonkey and Konqueror allow free redistribution, as they are all open source software, the first two released under the the Mozilla Public License

(MPL), the latter under the Gnu General Public License (GPL). Thus the choice of web browser was really a non-issue, Firefox was by far the best alternative.

4.2 Bootable DHIS2 CDs

The initial suggestion for research for a master thesis proposed to me in December 2006, after finishing the DHIS2-related open source course at the university of Oslo, was the idea of creating a bootable CD containing DHIS2. As mentioned in the chapter introduction, this had been an idea in the project for some time, spurred mainly by ideas of simplifying configuration/setup for the end user, and possibly avoiding issues related to viruses, in addition to gaining the ability to easily demonstrate the system.

This chapter will briefly outline what a bootable CD entails, our possible choices of existing technologies, which elements were relevant to our choices, why we chose as we did and finally cover our solution.

4.2.1 Live CD Technology

Live CDs provide an easy and convenient way for users to try out a new operating system (OS) by letting them run the software directly from a bootable CD. While variations of such Live CDs exist for many free operating systems, they usually are developed for a specific purpose by third parties(...) (Schaumann 2006)

The latter is exactly what we were interested in, as being able to demonstrate our third party software – DHIS2 – by simply rebooting the computer with the CD inserted, would provide the project with more ways to “show case” and spread the system.

Additionally, Schaumann (2006) notes how modern open source install CDs are basically Live CDs with a different focus, namely getting the operating system installed to the hard disk, but still Live CDs in the sense that they boot minimalistic versions of the operating system from the CD. This interchangeability in the terms and concepts are underlined by how many major Linux distributions offer Live CDs of their operating system, with the ability to install to the hard drive (Fedora Live CD 2008; Ubuntu Live CD 2008; Debian Live CD 2008). This was also part of our motivation for researching a Live CD and DHIS2 combination, as there was increasing interest in the project for running DHIS2 on Linux (in itself not a problem, and something many of the core developers do). However, both from discussions with other developers about their experiences regarding Linux in third world pilot projects, and my own experiences in the early stages of the local implementation effort in Kerala, it became obvious that the Linux installation itself could pose problems in these regions. A combined DHIS2 and Linux Live CD / installation CD was thought to be an interesting possibility for easing the process of setting up both Linux and DHIS2 in such cases.

Note that "Live CD" is not always an exactly accurate term, as at is quite possible to extend the media used to bootable USB memory sticks or similar (ibid.). Running DHIS2 from USB sticks has as mentioned been part of earlier installer projects.

4.2.2 A Plethora of Options

While the scientific literature on Live CDs are sorely lacking due to its quite technical nature and limited general applicability, the concept is quite popular and documentation and projects are plentiful in the world of open source software. So when we were looking at which of these distributions to base our version on, there were quite a few choices.

For instance the Live CD List (Live CD, 2007) has a listing of 315 different live CDs at the time of writing, and the vast majority (296/315 to be exact) are variations of Linux for the x86 platform[1], and as it was also the only OSS alternative in use or simply known to DHIS2 developers, core and local alike, Linux was the OS of choice.

Having decided on an OS, we then defined what we wanted from the Live CD, ending up with the fairly wide description of:

A CD that boots straight to a desktop where a user can simply click a DHIS2-icon to have it started, and that can be installed to the hard disk with a simple point and click interface from the same desktop.

Text 2: "Spec" for the bootable CD agreed upon by me and other developers

1 As Linux and most of the accompanying software making up what people call «Linux» (which is technically just the kernel) are free open source software, there tends to be a lot of different versions. Everyone who dislikes how something is done, or think there's a better way to do something, is free to change and/or repackage all of the software. This goes for the individual software pieces as well as the collection of them into a complete operating system, referred to as a distribution. Thus the Linux world has several different distributions, mostly based on the same software, but the way they are made up differs based on ideology, technology or plain old disagreements between developers.

From a technical point of view, this narrowed down our criteria to two main points:

1. As any choice of Live CD distribution will have to be modified to include DHIS2, ease of modification – commonly referred to as “remastering”, see chapter 4.2.3 – is perhaps the most important. Not only for the initial creation, but to ensure it continues to be easy to modify our solution for future releases of DHIS2 or similar scenarios where change is needed.
2. For installing the distribution to the hard disk, we want a solution where this is feasible and preferably extremely simple, allowing even people who knew little about Linux, or even computers in general, to perform an installation.

There was of course no way we could go through every item on the list of existing Live CDs. Nor would that serve any great purpose, as the majority are based on the same technology. So in our process of reviewing different distributions for our purpose, we limited our scope to a mix of Live CDs we found good reviews of (e.g. Wolvix Review (2007)) – working on the premise that good reviews could save us the time needed to review it ourselves, even if “regular” reviews would not focus on the same issues we would – and the most renown. I had had some experience with Linux Live CDs before, albeit from a few years back, and my recollection of what were the most widely used distributions coincided with what we found were currently the case.

A lot of different Live CDs were tested to find something suitable, including, but not limited to: Dreamlinux (DreamLinux 2007), Slax (Slax 2007), Wolvix (Wolvix 2007) and Knoppix (Knoppix 2007). All of the experiences are listed in the HISP documentation, see Appendix F.

Two distributions / technologies deserves special mention here, Knoppix and the Linux Live Scripts (LLS 2007), as they were the two most important distributions from a technical standpoint. Quite a lot, if not most, Linux Live CD distributions are originally based on one of these two. The latter is the basis for a host of popular Linux live distributions, perhaps most famous of which is Slax, while the former was for a while interchangeable with the term Live CD (Schaumann 2006).

4.2.3 Remastering

Both Knoppix-based and LLS-based systems are fairly easily customizable by a process referred to as «remastering», which means:

- downloading an existing CD-image of the Live CD
- expanding it
- adding, removing or modifying the included software
- creating a new CD-image with the changes included

Producing a custom version of the given Live CD. While this is as mentioned not terribly difficult for anyone with some Linux/Unix experience, quite a bit of research on various Live CD options was still needed to find something that was as close to what we wanted as possible, to minimize the changes needed and making sure the process was as simple as possible for later developers to repeat.

Viewing our choices in light of options regarding remastering, the Linux Live Scripts derivatives seemed the most interesting, due to how easy it was to add software to most of these distributions. Comparing The Knoppix Remastering Howto (2007), to the Zenwalk Remastering Documentation (2007) or Wolvix Remastering Documentation (2007) the need for documentation is far less involved for the latter. Of course, less documentation does not necessarily equate to less work, but the procedure for Knoppix remastering is a lot more involved. There is a lack of a lot of the tools available with LLS, meaning that there is more manual work. For example, using Knoppix, one would have to completely expand most of the directory structure on the CD, and then make whatever changes necessary, before repackaging. In comparison, with LLS based systems, it is simple to add software with the use of modules that can be added without any modifications of the base system at all. These modules can be created from existing Linux package-formats (rpm, tgz, deb) or custom-made from scratch, and can then be copied into a raw copy of the original CD, and then you simply run a script to create a new CD-image.

4.2.4 Wolvix

As alluded to by the previous chapter, our choices finally came down to the various Live CDs based on LLS, due to their, for our case, superior remastering process. At this point, however, not much separated our candidates in terms of technical solutions, or “look and feel”. The layout of the operating system and process for changing it was practically identical, and most had clean graphical interfaces upon loading, with only superficial differences, and slight variations in default programs, neither of which were heavy weighing matters for our purpose.

However, when it came to our second point of interest, installing to the hard disk, there were more significant differences. Most of the Live CDs had limited or cumbersome options for doing a hard disk installation, notable exceptions included Wolvix and DreamLinux, both of which had excellent install routines with a simple graphical interface. However, DreamLinux, being one of the few reviewed distributions not based on LLS or Knoppix, had its own remastering routine, which failed us miserably on all our attempts and thus disqualifying the project as a possibility.

An added bonus and the final touch was that the entire installation routine of Wolvix was written in Python, a scripting language I was quite familiar with, which also made modifying the installation routine itself trivial. While this had not up to this point been a major point for us, it did open up some very interesting possibilities for further customisation, as in cases where some of the variables that normally needs to be provided for an installation are known, the process can be simplified by explicitly providing these variables, and lessen the knowledge needed to complete the installation.

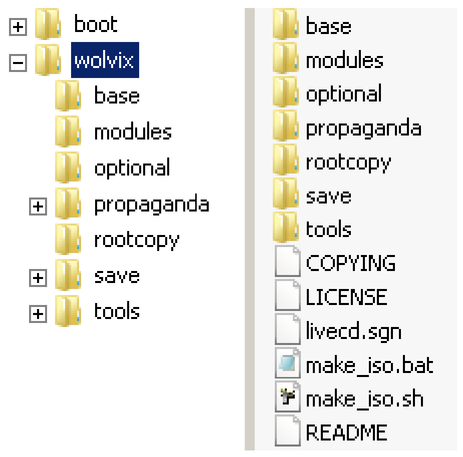
4.2.5 Wolvix DHIS2 Edition

After deciding on the Linux Live CD we were going with, we had to chose a version, as Wolvix came in two variants, “Cub” and “Hunter”. As the names allude to, Cub is smaller, with fewer features and programs, but as the only things missing from it that we wanted were also missing from Hunter, we decided to go with Cub, on the premise that size and memory use while running would be an important factor when using the CD on low end computers.

When modifying Wolvix Cub for our purposes, there was two main criteria we had to cover. In addition to adding all the software necessary for running DHIS2, as outlined in chapter 4.1, we had also decided that we wanted to provide a desktop that was sufficient for office use to ensure that a hard disk installation of the modified distribution would cover the same needs as a standard Windows XP setup, as otherwise it would be of limited use to end users.

Fortunately, most of the items we deemed relevant – audio, video, email, web browser, etc. – were already present, the only thing missing was a proper office solution, of which we concluded that the one that would be most familiar and thus intuitive to users would be OpenOffice.org (OpenOffice.org 2008).

For explaining how programs were added to the modified Wolvix Live CD, a more in depth look at the remaster process is required. The entire process can be done in either Windows or Linux, though anything beyond basic adding / removing of modules requires Linux, and it is started by copying the contents of the CD to be modified to the hard disk, which will produce a directory tree as shown here:



```
⊞ boot                base
⊟ wolvix              modules
    base              optional
    modules           propaganda
    optional          rootcopy
  ⊞ propaganda        save
    rootcopy          tools
  ⊞ save              COPYING
  ⊞ tools             LICENSE
                      livecd.sgn
                      make_iso.bat
                      make_iso.sh
                      README
```

Illustration 6: Wolvix directory tree

Ignoring most of this information, the important parts are the folders marked *base*, *modules* and *optional*, and the *make_iso.bat/make_iso.sh* scripts (Windows batch script and Linux shell script respectively). To create a new CD from this point, one simply has to add a module with the wanted program in either *modules*, to have it loaded at startup, or *optional*, to have it available for load on demand, and then run whichever *make_iso* script is applicable for the platform the remaster is performed on. These modules to add have to be in a specific format called "lzm", however these are common for all Live CDs based on LLS, and can additionally be created from other Linux package format with simple programs, so they exist in plentiful and are easy to create if no one has done it previously. Taking OpenOffice as an example, adding it in our case consisted of downloading an OpenOffice package from the Slackware project (a number of other sources would have sufficed as well), and converting it to the correct format, thus[2]:

2 Through the rest of this chapter, text frames containing only one line indicate one line linux commands, details of which will be outlined below the frame.

```
tgz2lzm openoffice-2.2.1.tgz openoffice-2.2.1.lzm
```

Here tgz2lzm is a Linux tool for converting files in the format of tgz to lzm. The resulting lzm-module was then copied to the *modules* folder on the remaster, and running the *make_iso* script would now produce an Wolvix iso-image including OpenOffice, ready for recording to a CD.

Regarding the software required for running DHIS2, the difficulties in including the components varied in difficulty. The Firefox web browser was already included as part of the original Wolvix CD so this was a non-issue, and Java was similar to OpenOffice in that pre-packaged versions were easy to find and convert. As I had a version of the Java 1.5 JDK lying around in the form of a Debian package (.deb), the conversion was done as follows instead:

```
deb2lzm jdk-1-5.deb java-1.5.lzm
```

As mentioned, the database server of choice ended up being MySQL for the first implementations done as part of my research, and so the Live CD needed to include a version of MySQL as a database back end for DHIS2. There were no available lzm-modules of MySQL, this again required us to look for packages modify. However, most distributions of Linux split the MySQL-package into several smaller packages, for example Debian has at least three distinct packages required to install for covering the aspects we needed (Debian MySQL 2008). To avoid trying to piece together a MySQL module from disjoint entities, we deemed it easier to modify a module taken from Slax, which included a wide array of Linux server options (dhcp-server, database-server, web-server, mail-server, dns-server etc), and trim it down:

```
lzm2dir servers.lzm tmp/
```

This extracts the servers lzm-module to a directory *tmp/*. Looking through *tmp/var/log/packages/*:

```
ls tmp/var/log/packages/
```

will then show which packages this module contains, and everything but the MySQL module can be removed like this:

```
ROOT=tmp/ removepkg dhcpcd sendmail bind ...
```

Once mysql is the only package left, the module can be repacked thus:

```
dir2lzm tmp/ mysql.lzm
```

This example also illustrates how the modules used in Wolvix can all be expanded to a directory tree with *lzm2dir,* modified, and the repackaged with *dir2lzm*.

One final modification required before repackaging MySQL was to prepare a database for DHIS2. MySQL stores its data and databases in a folder referred to as *datadir,* which in the package we modified is located in */var/lib/mysql/*, or in the example above of an expanded module, *tmp/var/lib/mysql/*. So by copying into this folder the contents of an existing MySQL database data folder, one can produce CDs with existing databases included, which can for instance be useful for showcasing DHIS2. In our case, we originally simply wanted an empty database for DHIS2 as proof of concept, which was solved by starting a new instance of MySQL, connecting to it, and creating an empty database suitable for DHIS2 by issuing the following MySQL-statements:

```
mysql> create database dhis2 default character set utf8 collate utf8_general_ci;
mysql> grant all on dhis2.* to dhis@"localhost" identified by '';
```

CodeExample 1: MySQL statements required for creating an empty database for DHIS2

After having done this and having shut down the MySQL server again, the new contents of the data folder could now be copied into the unpacked MySQL module for inclusion. In other applications of a DHIS2 Live CD different data may be wanted, in that case the data folder for MySQL can be changed to reflect that. For example, the commonly used sample data from South Africa could be included to show how DHIS2 functions with a fully fledged database.

A web application server had to be packaged for the Live CD as well. While the first iterations of the CD were using Jetty for this purpose, the solution was inelegant (not because of any limitations with Jetty, but due to being a first attempt) and proof of concept more than anything else, so the focus will be on how Tomcat, which was used in all later revisions, was included.

At the time this CD was created, there did not really exist any pre-packaged versions of the latest version, but as the vanilla Tomcat package is distributed as a simple self-contained folder, repackaging in the format of an lzm-module posed few problems. Upon downloading Tomcat, unpack it into a folder intended used for repackaging, and under a sub folder that makes sense. For instance, we wanted to have Tomcat under */opt/tomcat/* upon startup of the system, so it was unpacked to *tmp/opt/tomcat/*. Now a simple repackage similar to those shown above:

```
dir2lzm tmp/ tomcat.lzm
```

will produce a module that adds Tomcat to the CD, and would expand it to */opt/tomcat/* upon boot. However, some additional steps were needed, among them to include the DHIS2 war file in *tomcat/webapps/* as well as including a config file necessary for running DHIS2 in *tomcat/lib/*, but also creating a script for starting and stopping Tomcat when booting and shutting down the system. This file was added as */etc/rc.tomcat* and in the first iteration was as simple as follows:

```
#!/bin/sh

usage()
{
    echo "Usage: $0 {start|stop|run|restart|check|supervise}
[ CONFIGS ... ] "
    exit 1
}

export JAVA_HOME=/usr/lib/java

case "$1" in
  start)
        /opt/tomcat/bin/startup.sh
        ;;
  stop)
        /opt/tomcat/bin/shutdown.sh
        ;;
  restart)
        $0 stop $*
        sleep 5
        $0 start $*
        ;;
*)
        usage
        ;;
esac

exit 0
```

CodeExample 2: Startup / shutdown script for Tomcat for the DHIS2 bootable CD

This shell script will start or stop the Tomcat server depending on if it is called with “start” or “stop” as its parameters, very much like standard SysV rc-scripts found in most Linux distributions as the control points for starting and stopping services and daemons.

Finally, some changes had to be done to the base system. Going back to the overview of the

Wolvix CD, there is a folder called *base*, which includes basic modules that includes everything needed to run the operating system. Among them is a module named *core*, inside of which is, among many other items, all the files that control which services and servers start and stop when the system is booted or shut down. To include Tomcat in this process, the following code snippets had to be added to the files controlling startup and shutdown of the Wolvix system, */etc/rc.d/rc.M* and */etc/rc.d/rc.0* respectively:

```
# Start the Tomcat server
if [ -x /etc/rc.d/rc.tomcat ]; then
  /etc/rc.d/rc.tomcat start
fi
```

CodeExample 4: Code-snippet which includes the Tomcat RC-script in the startup routines

```
# Stop the Tomcat Server
if [ -r /var/run/tomcat.pid ]; then
  /etc/rc.d/rc.tomcat stop
fi
```

CodeExample 3: Code-snippet which includes the Tomcat RC-script in the shutdown routines

For the observant reader, similar measures were not necessary for the MySQL module as it – though it had been ripped out of another module – had still been a LLS module and the startup and shutdown scripts as well as the necessary additions to the control files were thus already there. This core module was repackaged in the same manner all other modules mentioned so far have been, the only difference being that it was of course added back to the *base* folder instead of the *modules* folder where the additional software was added.

At this point, the modified CD could be remastered by running the *make_iso* script, and recording the resulting iso-file to a CD. Booting a computer from this CD would then result in a Wolvix Linux desktop with DHIS2 running on a MySQL back end, which was the working solution we had before the local implementation effort in Kerala. As will be covered in chapter 4.3.3, quite a few further modifications were done to the system there however.

Complete technical details on how the Wolvix DHIS2 Live CD was created can be found in Appendix A.

4.3 The Kerala Case

From my initial contact and agreement to join HISP and work on DHIS2 for my master thesis, there was a further development in the spread of DHIS2. Trivandrum, the state capital in the

Indian state of Kerala, was to have DHIS2 installed on 80 new machines in clinics in the city. As Kerala state has a strong FOSS policy (Kerala State 2007), this was a great opportunity to do further research on both Linux and Windows DHIS2 installation and setup, and as such I was tasked to contribute to this implementation as part of my ongoing action research project.

4.3.1 Kerala Background

The state capital of Trivandrum (Indian name: Thiruvananthapuram) houses a HISP office, and they were contracted by the health ministry to install a HIS on around 80 clinics in and around the city. Their budget was of course limited, but each of the clinics were getting a relatively new computer for this purpose. However, there was for the time being no internet or intranet connection, though it was planned to be installed at a later date, and as such, all DHIS2 installations would have to be stand alone. The installation was meant to be done and the system in full operation by November 1st, 2007. I arrived in Trivandrum September 1st and left 11th October, and in that time we had to work out what was required, design and code up a solution, and test and implement it.

Regarding man-power and human resources, HISP Kerala is structured in layers. At the Trivandrum office you have the software developers and administrative staff. These were in the process of training up system facilitators, which were 12-15 people with minor IT background. They were given extensive training in the use of DHIS2, as well as basic training in how to get it installed on a computer. Once their training was done, they were given responsibility for 4-6 PHCs (Primary Health Clinics), where they would visit 1-2 times a week to train up the staff in how to use their new health information system. Additionally, they would take last weeks data with them when they left and bring it to the District Health Office, cf. the lack of internet / intranet connection at the clinics. There were also three senior facilitators who had been working with HISP for a while that served as contacts for the other facilitators in addition to handling their own PHCs.

Arriving in Kerala I was not sure exactly what to expect. I had a general idea of what I was there to do – help out with the installation of DHIS2 in the PHCs in the area – but nothing concrete on how. For instance, the earlier mentioned issue from Keralas Department of Information Technology indicated that a Linux solution would be of interest, and the aforementioned quote from HISP developers on virus-related problems supported this as a possibly interesting idea, but conversations with HISP-people suggested that the staff who would be developing and using DHIS in Kerala would have almost exclusively experience with MS Windows. This I confirmed shortly after arrival in discussions with the local HISP personnel. There was technical interest and professional curiosity about a Linux/DHIS2 combination, but clearly overshadowed by a want to go with something familiar, and a need for a solution that could be implemented quickly. So the goal we set for our work there in the first couple of days, was an easy Windows installation, with the possibility of a Linux installation working alongside it.

4.3.2 Basic MS Windows Installation Routine

The first few days of our work in Kerala was spent doing research on how to best approach the problem of creating an installation routine that would suit the local needs. The local HISP developers were interviewed – albeit not in the strict, structured sense of the word – to complement the ideas of global HISP developers and my own ideas on how best to proceed. The general agreement seemed to be that, as time was of the essence, something that could be both developed and employed quickly was preferred.

However, the perhaps most interesting data came from observing the training of the system facilitators. Some of these had literally touched a mouse for the first time around three months prior to our stay, and as such were not very comfortable around computers yet. Because of this, we soon decided that we needed to adhere strictly to a certain software design principle, namely «Keep It Simple, Stupid», or KISS. As the name so eloquently indicates, the concept urges simplicity in software design: «design simplicity should be a key goal and unnecessary complexity avoided» (KISS 2007). This we viewed as important to keep from confusing users, especially those with limited computer knowledge, and the anecdotal evidence of my four years of working in IT support confirms this.

With this basis in mind we started looking at technologies. As mentioned in chapter 4.1.2, there is quite a few pieces of software needed to run DHIS2 and since all the computers in question were new installations, we had a pretty clear idea of what software could be expected to be present, which was basically none of the components we needed. So our installation package needed to include all of these. As thoughts on each of the possibilities are outlined above, it's sufficient to state that for our Kerala implementation, we started out with MySQL, Jetty, Firefox and Java as our platform.

However, during later testing, we had some inexplicable problems with Jetty. For some reason, Jetty would sometimes «hang» on slower computers. This was especially apparent on one computer at the HISP office which was used a lot for testing, as it had only 256MB RAM, half of what most of the computers composing the target for our installation packaged had, and thus served nicely as a baseline, on the premise that "if it runs here, it will run anywhere". There would be no error-messages apart from a single line indicating a time-out waiting for the MySQL service, which was running fine, and a restart of the Jetty service would always fix the problem. As it so happens, this was during the most frantic first phase of the project, and we were quite pressed for time, so to avoid wasting time on debugging that might not produce results, we simply switched web application server to Tomcat, as Tomcat did not produce any similar issues.

Concerning how these components were strung together, this is also documented more thoroughly in Appendix B, but in short our installation package was organized as follows:

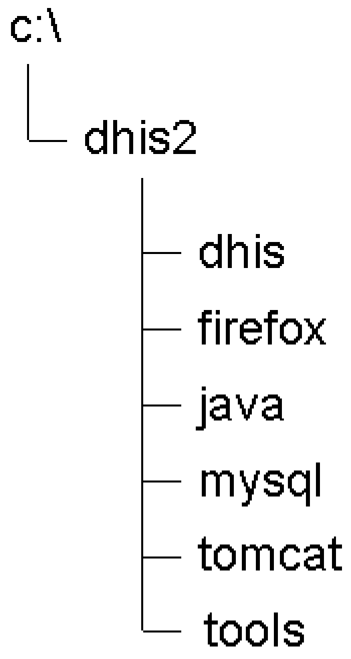


Illustration 7: Kerala DHIS2 software structure

While this directory structure did not necessarily rely on a certain location, the *dhis2* folder could technically reside anywhere in the file system, we decided on a common location for all

PHC computers, *c:\dhis2,* both for simplicity and inter-OS compatibility, described in the next chapter. This structure could be compressed with any regular compression algorithm (we used plain old zip), and uncompressed at a target machine, which even on the slowest machines we tested on did not take more than a few minutes.

Regarding the sub folders, there was one for each of the software components necessary, as well as two different folders for utilities and extras. The folder marked *dhis* contained some configuration files, shortcuts and XML files dictating the layout of local data reports. These reports were custom tailored to resemble the paper reports in use in the health sector up until the DHIS2 transition. The idea was to ensure that the transition was as smooth as possible for the nurses in the PHCs, by leveraging a sense of familiarity. The tools folder contained the scripts and programs required to install, remove, manipulate and use the software package, these will be covered in detail later in the chapter.

The *java* and *firefox* folders were fairly straightforward copies of what was at the time the latest distribution of the software. Similarly with the tomcat folder, but this also contained the DHIS2 war file residing in *c:\dhis2\tomcat\webapps\*, as well as a configuration file for DHIS2 in *c:\dhis2\tomcat\lib\* (see more details below).

For MySQL, data in the database is as mentioned stored in a folder designated “datadir” by MySQL, in this case *c:\dhis2\mysql\data*. As the solution we were developing was to be used in production, pre recorded data and data structures was of course a necessity. In short, DHIS2 relies on a structure of Organization Units, one for each node collecting data (PHC, District Health Office, Ministry of Health, etc.), and this structure has to be defined up front for users to be able to enter data for the relevant Organization Unit. Additionally, dataelements and indicators, as well as the datasets they are connected to, need to be defined for the region in question. All of this data had already been prepared by the HISP India developers precipitating our arrival, and now had to be included in the software package so that the workstations in the PHCs would have the required content in DHIS2 for their data collection.

During this setup, we discovered a long-lasting “bug” in DHIS2, regarding how its resource directory was being searched for. This directory should contain implementation specific resources as well as the configuration file required to connect to the database . In our case, the first affected the XML files for the local reports. To explain the second properly, some quick notes on how DHIS2 connects to a database upon start up is needed:

As mentioned, DHIS2 uses the Hibernate framework to communicate with the database, to make the choice of database server transparent to the application. This requires the application to get some information to pass on to Hibernate however, on the specific information needed to connect to the database. This is read from a configuration file, called "hibernate.properties", which lists what Hibernate needs: database URL, port number, user name and password. An example "hibernate.properties" for use with MySQL running locally is shown below:

```
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class = com.mysql.jdbc.Driver
hibernate.connection.url = jdbc:mysql://localhost/dhis2?
useUnicode=true&characterEncoding=UTF-8
hibernate.connection.username = dhis
hibernate.connection.password =
hibernate.hbm2ddl.auto = update
```

CodeExample 5: Sample hibernate.properties file

The problem with the method employed by DHIS2 to search for the resource directory upon starting up, was flawed in the sense that it used Java properties to search for a variable named "user.home". However, this location varied greatly on the computers in the PHCs in Trivandrum, unlike on my own and other developers computers. It turned out that the location was not necessarily consistent across Windows XP versions, which was not acceptable in our case, we needed to be able to provide uniform installations with all files in the same place regardless of slight variations in the Windows XP install. For the "hibernate.properties" file, this could be circumvented by adding the file to Tomcat's *lib* directory, as this would always be included in the the web application server's search path. However, this was somewhat of a "hackish" solution, and it did not work for the XML files for the reports, so a change was needed. As the Java property could not really be set in any way on the machine, and since mailing list discussions agreed that this was not the intended behaviour, the DHIS2 source code was modified to use environment variables instead of Java properties for building a search path, allowing us to set specific locations for the hibernate configuration file as needed:

```
propertiesFilePath = System.getProperty( "user.home" ) + File.separator + "dhis";
```

Text 3: Before

```
propertiesFilePath = System.getenv( "USER_HOME" ) + File.separator + "dhis";
```

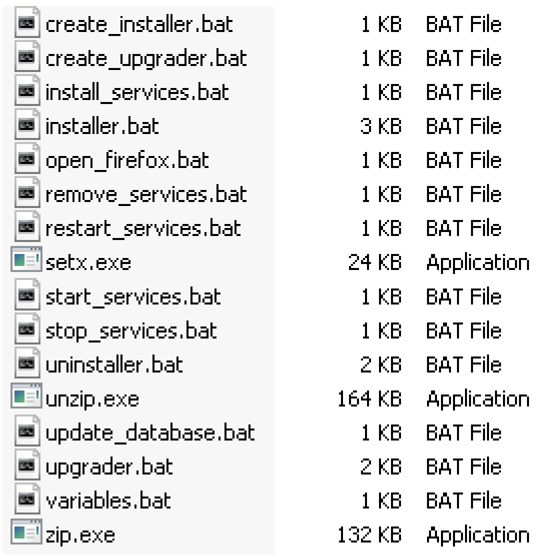
Text 4: After

After creating a software structure and modifying DHIS2 as required, the next question was how to perform the install. At this point, the remaining process of installation boiled down to automatically unpacking the software package and setting up a way for DHIS2 to be run. Since the machines in question were to be used primarily for data collection through DHIS2, it was decided that DHIS2 should be run at start up, which suggested that running MySQL and Tomcat as Windows services as the simplest solution, as both projects had documentation available for how to set this up.

In general, installation routines for DHIS2 on windows were a bit lacking when we started, though as mentioned, some work on more automated setups had been done. The official pages had rough guides, but they were fairly complicated, requiring a manual install of a database server and a web server, including setup of these. The HISP team in Ho-Chi-Min City, Vietnam, had created some scripts for use with Installer2Go (Installer2Go 2007), and a team in Ethiopia had done some work with NSIS (NSIS 2007). These were valuable resources for our work, and the scripts from Vietnam in particular saved us a lot of time, though modifications had to be done as their solution was based on PostgreSQL, while we had decided on MySQL.

In the end, our first version of an installation routine was written almost entirely as batch scripts, and again, complete documentation, examples and code can be viewed in Appendix B. Going with batch scripts was partly because this let us work with the aforementioned material from Vietnam, but also because it was the quickest way to our goal, and time was as mentioned of the essence. There was of course another aspect of that decision, which is that the method of installing programs on a windows-system that is most common, is a GUI wizard with several pages you have to click through with a «Next»-button. (And indeed, time showed that there was a want for such an installer for DHIS2 as well, as I will discuss later, in chapter 4.4.1) So common design paradigms would have this be the logical choice for an installation interface. However, we came back to the fact that several of the people who would be using the installer would not have any experience with program-installation in general, and that the best choice would be an installer that required as little input as possible.

This is an overview of the contents of the tools folder:



```
create_installer.bat      1 KB   BAT File
create_upgrader.bat       1 KB   BAT File
install_services.bat      1 KB   BAT File
installer.bat             3 KB   BAT File
open_firefox.bat          1 KB   BAT File
remove_services.bat       1 KB   BAT File
restart_services.bat      1 KB   BAT File
setx.exe                 24 KB   Application
start_services.bat        1 KB   BAT File
stop_services.bat         1 KB   BAT File
uninstaller.bat           2 KB   BAT File
unzip.exe               164 KB   Application
update_database.bat       1 KB   BAT File
upgrader.bat              2 KB   BAT File
variables.bat             1 KB   BAT File
zip.exe                 132 KB   Application
```

Illustration 8: Tools content

The applications – setx, unzip and zip – are freeware tools for setting environment variables, and unpacking/packing zip files respectively.

Of the bat scripts, the most interesting is the installation script *installer.bat* – details of which are outlined below, with comments stripped out for brevity – as it covers most of the functions used in the remaining scripts and serves as a good example of the code written for our implementation.

The following sets default location for the installation at the top of the installation script, more to allow for easy use of a single variable whenever the installation directory is needed, than for setting a different one, as differing locations will break the OS interdependency covered in 4.3.3.

```
set drive_letter=c
set dhis2_dir=%drive_letter%:\dhis2
```

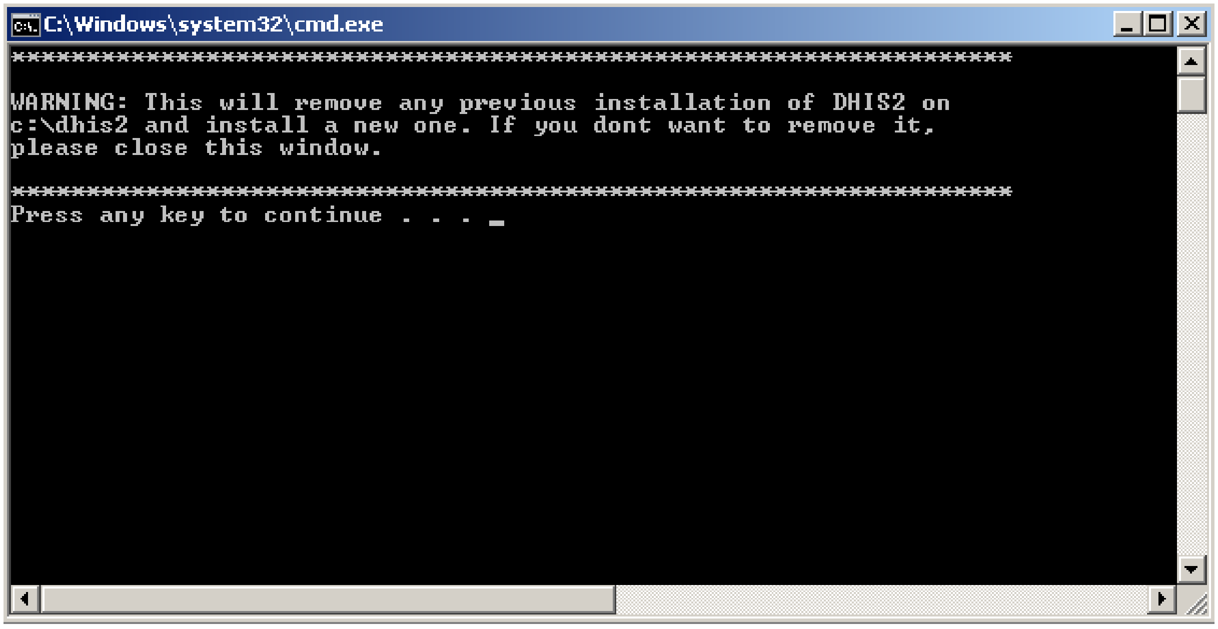
CodeExample 6: Default installation location

This is the only interactive part of the installation

```
cls
echo ******************************************************
echo.
echo WARNING: This will remove any previous installation of DHIS2 on
echo %dhis2_dir% and install a new one. If you dont want to remove it,
echo please close this window.
echo.
echo *****************************************************
pause
```

CodeExample 7: Kerala Windows DHIS2 first installation screen

as this produces the following screen:



```
*******************************************************************

WARNING: This will remove any previous installation of DHIS2 on
c:\dhis2 and install a new one. If you dont want to remove it,
please close this window.

*******************************************************************
Press any key to continue . . . _
```

Illustration 9: Kerala Windows DHIS2 first installation screen

The installation here continues and completes if you press any key, allowing for as automated an install as possible while still allowing for cancellation if started in error.

This next part will remove any old services by the same name as the services used by the installer, this to prevent errors if the services already exist. Mainly useful in the testing and debugging phase, as the computers subject to installation would never have any of these services in place prior to installation.

```
echo Removing old services (if they exist)

echo Stopping and removing MySQL service.
net stop mysql
sc delete mysql
echo Removing MySQL service done.

echo.
echo Stopping and removing Tomcat service.
net stop tomcat6
sc delete tomcat6
echo Removing Tomcat service done.
echo.
```

CodeExample 8: Removing old windows services

The same goes for the following commands, which simply remove any existing software before unzipping the software structure.

```
echo.
echo Unzipping dhis2.zip, this might take a few minutes.
rd /s /q %dhis2_dir%
mkdir %dhis2_dir%
unzip -q -o dhis2.zip -d %dhis2_dir%
echo Unzipping done.
```

CodeExample 9: Removing old and unzipping new software

After having unzipped the software, to installer needs to set some variables both for use in the install script (the "set" lines) and for later use by the software whenever the system is started (the "setx" lines):

```
echo.
echo Setting JAVA_HOME to %dhis2_dir%\java
set JAVA_HOME=%dhis2_dir%\java
%dhis2_dir%\tools\setx.exe JAVA_HOME %dhis2_dir%\java -m

echo.
echo Setting USER_HOME to %dhis2_dir%
set USER_HOME=%dhis2_dir%
%dhis2_dir%\tools\setx.exe USER_HOME %dhis2_dir% -m
```

CodeExample 10: Setting windows environment variables

After these have been set, it saves the installation location to a local script, which all other scripts will call upon to determine the location of the installation, to enable easier relocation. Again, this was not recommended, but to make it easier in the case it was wanted, we strived to write our scripts to be easy to accommodate.

```
echo.
echo Saving install-dir location
echo set drive_letter=%drive_letter%> %dhis2_dir%\tools\variables.bat
echo set dhis2_dir=%dhis2_dir%>> %dhis2_dir%\tools\variables.bat
```

CodeExample 11: Saving install location

During discussions with the local HISP developers, the issue of users “messing around” with the system was raised, as several of them had experience with users trying to fix and add things which they did not fully understand, potentially creating problems for themselves. As a stopgap measure, we added an extra non-admin user to the system named “dhis2”, and set a password on the account doing the installation, which would invariably be the administrator account. We had no illusions to this being anything but “security through obscurity”, but it was the hope of the local developers that it would cut down on what they viewed as unnecessary support:

```
echo.
echo Setting up user "dhis2"..
net user dhis2 /delete > nul
net user dhis2 /add /passwordreq:no /usercomment:"DHIS2 user"
net user %username% nrhmkerala
```

CodeExample 12: Adding windows DHIS2 user and password "protecting" current user

Shortcuts to starting DHIS2 were added to both the desktop and the Start Menu:

```
copy /Y %dhis2_dir%\dhis\DHIS2.lnk "%allusersprofile%\Desktop\DHIS2.lnk"
> nul
copy /Y %dhis2_dir%\dhis\DHIS2.lnk "%allusersprofile%\Start
Menu\Programs\DHIS2.lnk" > nul
```

CodeExample 13: Windows shortcuts

Then the Windows services for MySQL and Tomcat were created and set to start automatically when the computer was started:

```
echo.
echo Creating service MySQL.
cd /d %dhis2_dir%\mysql\bin\
mysqld-nt.exe -install

echo.
echo Creating service Tomcat.
cd /d %dhis2_dir%\tomcat\bin\
cmd.exe /c service.bat install
rem NB! the space between "start=" and "auto" has to be there!
cmd /c sc config tomcat6 start= auto
```

CodeExample 14: Creating windows services

And finally, the system was started after installation:

```
echo.
echo Starting MySQL service.
net start mysql

echo.
echo Starting DHIS2, this might take a few minutes.
net start Tomcat6
echo ********************************************************
pause
```

CodeExample 15: Starting services

The remaining scripts written as part of our implementation do not really cover anything new after a thorough coverage of the installation script, as they are mostly designed as helper scripts for manipulating services or aiding starting / stopping of DHIS2. Notable exceptions include the *uninstaller* script which is basically a reversed installer script to remove services and files, the *create_installer* script which was used during development to produce new installers quickly after changes but has no practical use in a production installation, and finally the *upgrader* / *create_upgrader* scripts, both covered in detail in chapter 4.3.5.

While the installer we created was in no way a very fancy solution, it fit our criteria of simplicity and usability, and it got the job done. Most of the feedback we received on it was in the form of discussions and comments at the HISP India office, but I have included one quote from an email I received from a HISP India developer a couple of months after my stay in Kerala ended:

> This installer helped us a lot for installing.Previously we will take nearly 30 minutes for the DHIS2 installation.But now by ur installed by just a single enter key everything will be installed.it's a great job.
>
> *Mail 4: To me from a HISP India developer, subject: Re: hello, date 13.12.2007*

4.3.3 Modifications of Wolvix DHIS2 Edition for Kerala

As mentioned in the chapter introduction, there was interest both at state level and from core HISP developers regarding open source software, and so we did work on solutions for running DHIS2 on Linux on the PHC computers in parallel with our work on the Windows installers. The basic idea was to be able to easily install Linux, hopefully easily enough to let all the system facilitators perform the installation, and preferably let DHIS2 on the two operating systems share the same database, so that they would not be tied down to using either OS after starting to enter data into the system.

Work on the subject was of course based on my preliminary work on running DHIS2 from a bootable Linux CD, as per chapter 4.2.5, and indeed one of the criteria that Wolvix fit in becoming our choice, was the ability to install the Live CD to the hard disk easily, however we tackled the issue of sharing data first, as without this, easy hard disk installation would lose most of its appeal.

Using a common database across the two operating systems, is not as straightforward as it could sound, considering that the two use very different file systems and technologies for accessing files. Windows XP has no native way of accessing file systems from other operating systems, though third party software exists that makes it possible (e.g. FS Driver 2008). Linux has historically been a lot better at accessing the file systems of other OSs, but for a long time the FS used by Windows XP (and it's predecessors Windows 2000 and Windows NT), NTFS, was not one of them. Fortunately, through the work of the Linux NTFS Project (2008), full read and write support of NTFS was available in Wolvix, meaning that after correctly setting up the Linux installation, it was possible to access all the DHIS2 software installed on Windows XP on the same machine in Linux.

Having found out how to share files between the two OS's, the key to sharing data between the two database installations was taking advantage of how MySQL stores it's databases. All MySQL databases are contained in a single data folder, referred to as *datadir* in MySQL configuration files and command line options, and this can be set to an arbitrary location. MySQL in the Windows package was set up to take advantage of this, as a service with a specified directory set up as the *datadir* (*c:\dhis\mysql\data* for reference). As Wolvix was set up so that Windows XP file system could be accessed (was mounted) under */mnt/<hdtype>/* and we knew from our Windows installation package that the DHIS2 software was installed under *c:\dhis2* this led to the Windows MySQL installations *datadir* being located in */mnt/<hdtype>/dhis2/mysql/data/* and the Linux MySQL installation could then set its *datadir* to that location to read the same databases.

Two problems remained though, the first of which requires some more details on the computers in the clinics in Trivandrum:

The computers were ordered to a specification written by one of the HISP India developers after discussions with core developers, and two Indian companies, Wipro and HCL, were contracted to deliver 80 identical computers. The exact split was hard to get a number on, but it seemed to be around 50/30 Wipro/HCL from our testing phase, and a Wipro computer was also the only one we had access to in early development.

When you order computers to a specification, you would expect each computer to be identical, but this was rarely the case. Even when they were the same brand, there were often subtle differences between them, which could throw off especially the Linux part of our work. Small technical details on how the hardware is put together can make a big difference, and it was really strange to see how there was little consistency in how the computers were set up, including, but not limited to, partitioning and IDE setup. Almost as if each computer was put together by a different technician, each with a slightly different view on the specifications.

The relevance of the differences in the computer setups to the MySQL issues came from the fact that the type of the hard disk and how it was connected in the machine decided what name the Windows XP partition was mounted as under Linux. For instance, Serial ATA IDE hard drives showed up as sda, sdb, sdc or sdd, while “regular” IDE hard drives showed up as hda, hdb, hdc or hdd depending on how they were connected and which jumpers were set, none of which was possible to know up front. Partition numbers could likewise be anything from 1 to 4 for the partition we were interested in. I personally witnessed at least three different configurations, and so the MySQL startup scripts from the MySQL module had to be modified to accommodate for this, the added logic is shown below:

```
WIN_DIR=""
# see if there's a windows install on the machine
if [ -r /etc/rc.d/rc.dhis2 ]; then
  . /etc/rc.d/rc.dhis2
fi

if [ ! -z $WIN_DIR ]; then
  if [ -d $WIN_DIR/mysql/data ]; then
    DATADIR=$WIN_DIR/mysql/data
  fi
fi
```

CodeExample 16: Shell code to find and read MySQL data from a Windows XP partition

The script calls another script common for our modified MySQL and Tomcat modules, which searched for a readable Windows file system with a dhis2 folder. If found, this would be used instead of the local Linux data directory. This common script, which basically loops through all the possible different hard drive configurations and sets a variable if it finds a dhis2 installation on a Windows XP partition, was written as follows:

```
#!/bin/sh

WIN_DIR=""

for type in "hd" "sd"; do
  for drive in "a" "b" "c" "d"; do
    for part in "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"; do
      #echo "Trying /mnt/$type$drive$part/dhis2/"
      if [ -d /mnt/$type$drive$part/dhis2 ]; then
        WIN_DIR=/mnt/$type$drive$part/dhis2
        #echo "Found windows install at $WIN_DIR"
      fi
    done
  done
done
```

CodeExample 17: Shell code to search through all possible Windows partition namings for a mounted partition

This script was far from a perfect solution, it loops inefficiently through so many different possibilities and will simply use the last one it finds, however, it was sufficient to cover all the different configurations on the clinic computers, and we knew that no combination of hardware setup and DHIS2 installation should result in more than one hit.

One final problem remained, that Windows and Linux file systems do not have the same views on file permissions. In Linux, MySQL expects to be run as a named user, “mysql”, and the data directory to be owned by this process, without which it will refuse to run. However, because of limitations in the Linux NTFS driver, all files on an NTFS volume mounted in Linux will be owned by the user who mounts the volume, which is usually root. Because of this, how Wolvix mounted the Windows partition had to be modified, which entailed changing a line in the control file for file systems, */etc/fstab*, to set the mysql user as the owner of the file system:

```
/dev/sda1 /mnt/sda1 ntfs-3g uid=1000,gid=1000 0 0
```

CodeExample 18: Example /etc/fstab line for NTFS partitions mounted as specific users

Here the numbers uid and gid are the file and group numbers of the mysql user, which can be found in the configuration file for users, */etc/passwd*.

Together these modifications ensured that as long as DHIS2 had been correctly installed on a Windows partition, our DHIS2 package in Linux would run using the same database data.

Similar to the task of ensuring MySQL in Linux used the same data as the Windows version, setting up DHIS2 and Tomcat correctly in Linux required some tricks. As our Linux package had full access to the Windows version, we decided to use this to our advantage and let the Linux Tomcat scripts grab the DHIS2 war file and the XML files for the local reports from the Windows installation, so that upgrades and changes would only need to be done in one operating system (cf. the work described in chapter 4.3.5). The Tomcat module was modified again to include the following in the startup scripts:

```
        WIN_DIR=""
        # see if there's a windows install on the machine
        if [ -r /etc/rc.d/rc.dhis2 ]; then
          . /etc/rc.d/rc.dhis2
        fi
        # See if we can find the war-file on a windows partition
        echo "Looking for dhis.war file..."
        rm -rf /opt/tomcat/webapps/*
        WAR_FILE=/opt/tomcat/dhis.war
        if [ ! -z $WIN_DIR ]; then
          if [ -r $WIN_DIR/tomcat/webapps/dhis.war ]; then
            WAR_FILE=$WIN_DIR/tomcat/webapps/dhis.war
          fi
        fi
        ln -s $WAR_FILE /opt/tomcat/webapps/dhis.war
        echo "Using $WAR_FILE"

        # see if we can find the ra-folder on a windows-
partition
        rm -rf /root/dhis/*
        RA_FOLDER=/opt/tomcat/dhis/ra/
        if [ ! -z $WIN_DIR ]; then
          if [ -d $WIN_DIR/dhis/ra/ ]; then
            RA_FOLDER=$WIN_DIR/dhis/ra/
          fi
        fi
        ln -s $RA_FOLDER /root/dhis/ra
```

CodeExample 19: Tomcat RC-script modified to search for DHIS2 war files and reports on Windows XP partitions

The Tomcat installation was as mentioned located in */opt/tomcat*, and the DHIS2 war file bundled with the Linux package was placed in */opt/tomcat/dhis.war*. Upon Tomcat start, all contents of */opt/tomcat/webapps/*, where all web applications are located, were deleted, and if a Windows partition with a war file in the right location was found (using the same script detailed above) this file was used, otherwise it used the one bundled with the Linux package. Similar logic was applied to the XML files for the local reports. Thus the DHIS2 software and reports would always be the same in both operating systems.

After completing the technical setup of DHIS2 modified to interconnect with the Windows DHIS2 software package for Kerala, the Wolvix desktop had to be modified to be as usable as possible for potential users, all of whom would likely be fairly unfamiliar with computers in general, and Linux in particular. As the rest of the chapter is not as relevant to the main focus of my research, I will go into less detail here, the complete documentation can be found in Appendix A.

The first order of business was to clean up the desktop, removing all icons and shortcuts we deemed uninteresting for our purpose, and then adding shortcuts to OpenOffice and DHIS2 to the bottom panel for easy access. All of these files and changes are part of the Wolvix module *core* discussed earlier, and the process for modifying it is the same as well. Then the Wolvix boot process was changed so it would automatically log in a new user, called dhis2, mirroring the Windows setup, and the same password was set for the root user in Wolvix as for the Administrator user in Windows.

The final course of action for the new Linux CD was to ensure that the process for installing it to the hard disk was as simple as possible. As the installation program was "hidden" in a menu choice, we added a shortcut to the desktop instead. The installation programs themselves were located in another Wolvix module, *wolvix-control-panel*, inside of which could be found two python scripts, *wolvixinstall.py* and *mkbootloader.py*, covering the installation and the creation of a boot loader respectively. The installation routine was simplified a lot, reducing the number of mouse-clicks necessary, as I could define quite a few default options on knowledge of the computers we were installing on. Among these was the automatic installation of a boot loader without prompting (this was necessary to allow for selecting which operating system to start when booting the computer). The boot loader script was modified so it configured sensible names for the OS menu, and written to default to booting Windows after a short wait, to ensure that users who knew nothing about Linux would get a more familiar interface unless specifically choosing otherwise.

Additionally, during our preliminary testing of this new version of Wolvix, it turned out that the low memory computer at the HISP office mentioned earlier could not easily run the CD in Live mode with all modules added. So our final iteration moved all the extra DHIS2 modules to the *optional* folder in the Wolvix directory structure, to avoid them being loaded at startup, and a small piece of code was added to the installation script to include all of these modules in a hard disk install:

```
# DHIS modification, automatically install all modules in /optional
# (so we can put our stuff for installation here for quicker live CD load)
self.TextBufferChange(self, None)
self.textbuffer.insert(self.textbuffer.get_end_iter(), "\nInstalling modules from
optional (i.e. DHIS2)...\n")
self.TextBufferChange(self, None)
basedir = "/mnt/live/mnt/"
moddir = ""
for dir in os.listdir(basedir):
        moddir = basedir + dir + "/wolvix/optional"
        if os.path.isdir(moddir):
                break
if not moddir:
        self.textbuffer.insert(self.textbuffer.get_end_iter(), "Could not find the
optional directory, won't install any of the modules (i.e. DHIS2...)\n")
else:
        for f in os.listdir(moddir):
                file = moddir + "/" + f
                cmd = "lzm2dir %s %s" % (file, mountpoint)
                self.textbuffer.insert(self.textbuffer.get_end_iter(), "Installing:
%s\n" % f)
                self.TextBufferChange(self, None)
                os.system(cmd)
                self.TextBufferChange(self, None)
self.textbuffer.insert(self.textbuffer.get_end_iter(), "\nFinished copying
directory structure\n")
```

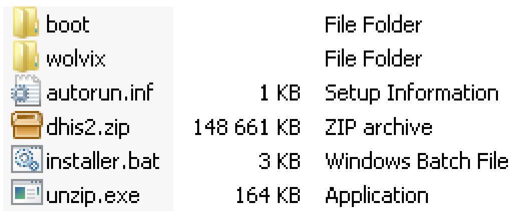
CodeExample 20: Python code added to the Wolvix hard disk installer to include optional modules during installation

However, the most complex, at least from the point of view of a less technically skilled user, part of the Linux installation, the repartitioning of the hard disk we were powerless to simplify. The aforementioned widely different configuration of hard disks and partitioning schemes from the computer vendor contracted by HISP India made any attempt at automating the process futile, as in a worst case scenario, such an automation could easily completely erase the Windows XP installation present. As a plan B, we instead wrote a detailed guide for the entire process, with screenshots and explanations of every step. Excerpts from this guide can be viewed in Appendix G. Observing the facilitators working with this guide and trying out the Linux installation, it was very obvious however that the repartitioning was causing them by far the most problems – in the form of uncertainty and fear of making errors – and our irritation at the computer vendor grew, as their inability to deliver identical computers severely hampered our ability to automate the issue causing our users the most headache. In fact, the entire implementation process ended up being done mainly be me, two other HISP developers and the senior facilitators in great part because the process of installing Linux was deemed to hard for the junior facilitators. As an illustration of the last point, we had exactly one case of installation where a junior facilitator was not able to complete the task, I came and finished it a couple of days later. He had been thrown off by a computer configuration he had not yet seen, and so had only been able to complete the Windows installation.

4.3.4 Common Installation CDs

Upon completion of both the Windows XP installation routine, and the modifications of the Wolvix DHIS2 CD, we were faced with the issue of how to distribute the implementation. While there were not that many facilitators, we found it preferable to try to combine the two to reduce both the amount of CDs and the issue of explaining which CD was for which part, and I had also gotten a feeling from observing the facilitators that if we handed them a "Windows CD" and a "Linux CD", the latter would more easily be "forgotten".

As fortune had it, our decision to go with a light Linux version produced a Live CD that even with all the DHIS2 modules and OpenOffice still had quite a bit of space left on the CD, enough to add all the files needed for the Windows installation routine. Looking back to chapter 4.2.5 and the directory layout of the Wolvix CD, the root of the CD contains only two folders, and anything outside of these are completely ignored by Wolvix. Additionally, the file system of CDs is common for all operating systems, so the root of the CD could easily be used for the Windows files, allowing us to create a common CD image, as shown here:



```
boot                      File Folder
wolvix                    File Folder
autorun.inf        1 KB   Setup Information
dhis2.zip    148 661 KB   ZIP archive
installer.bat      3 KB   Windows Batch File
unzip.exe        164 KB   Application
```

Illustration 10: Contents of install-CD

The file named *dhis2.zip* contained the Windows directory structure, *installer.bat* the install script outlined in chapter 4.3.2, *unzip.exe* was required to unzip *dhis2.zip* automatically, and the *autorun.inf* file instructs Windows with what to do when the CD is inserted when Windows is running. The last is extremely simple, containing in our case only two lines:

```
[autorun]
open=installer.bat
```

CodeExample 21: Contents of autorun.inf file

So when the CD was inserted in Windows, the installation routine would start automatically, meaning all that was required to complete it was to press a key at the prompt. After completion, the machine could simply be rebooted with the CD still inserted, and the Linux Live CD would boot and be ready for installation.

4.3.5 Testing and Implementation

After the initial prototypes of the installation CDs were produced, a great deal of time was spent doing testing. This testing consisted of me and one more person from the HISP India team travelling from PHC to PHC installing the system and noting what went wrong if anything.

Most of the issues we encountered during this phase, was related to the aforementioned setup of the computers in the clinics, there continued to be small surprises for us almost every trip we took. This phase was time-consuming for many reasons. The PHCs were physically quite spread out, and distances were magnified by bad roads and transportation, and in many cases the exact location of the PHC was not known either and our driver often spent time asking directions. Additionally, there was no phone registry with contact persons for the PHCs, so our trips were hit and miss, more than once we would finally locate a PHC only to find out that it was closed for the day due to a local holiday. Or testing would take longer than necessary because the computers were not actually set up or the power was gone when we arrived.

In addition to my own testing, another HISP India developer started a bit later with separate testing and implementation trips to different PHCs, and would share his notes and experiences with us in the evening. His experiences rarely varied from my own, as the major issues had already been ironed out, but one of his visits produced a very interesting reaction. The PHC in question had as the others received a new computer, but here one of the employees had prior experience with computers, and had already been using the computer a bit. However, the problem of viruses mentioned in the introduction had really struck on this computer because of this, and because of this it was so slow DHIS2 could barely be installed, and was not really usable. The HISP India developer merely noted to me upon explaining this in the evening that:

> but it is not a problem, they can just use Linux instead
>
> *Text 5: HISP India developer commenting on virus-infected Windows XP installation*

As the days passed, we found less and less bugs, and most of our testing ended up producing working results, partly because it was usually trivial for me to fix issues on the spot as well as fix it in the installers in the evening, and so most of our test cases also ended up working as implementations, as we were pressed for time. The same was true for the testing done by the other HISP India developer, and in fact, very few of the installations were carried out by the junior facilitators at all, mostly just the few clinics under 2 facilitators with slightly more computer experience. Despite the training we had done at the office, and the fact that the installations should be almost completely automatic, the senior facilitators preferred doing it themselves in case of problems.

4.3.6 Basic Upgrade Routines for DHIS2

After finishing the testing phase, I had a discussion with the local team about their further needs. The most important technical issue after having a way to easily install DHIS2 was, naturally enough, having an easy way to upgrade the software to later releases. While normally the “If it ain't broken, don't fix it”-principle for software in production use is applicable, there's compelling arguments to be made for being more relaxed in this sense when the system in question is a rapidly evolving open source software system, which is not yet feature complete. This was especially a problem in Kerala, as the local developers working on the reports needed for the clinics to report their data upwards to the district office, were changed almost daily(!) while we were there. The "final" version we were installing in the beginning had to be changed after a couple of days for example, which was unnecessarily time-consuming, and could be made significantly easier by having a small upgrade routine, instead of either installing the entire package again, or doing the changes manually.

As a side-note, that last part was a source of great frustration for both me and other members of the local team, since we more than once ended up having spent an entire day in the field installing the system on various PHCs and talking to the staff, and then coming back to the office and hearing our time had been wasted, it had to be done again due to changes in report formats or database setup.

It did however highlight a problem with DHIS2 that had not seen much research before, upgrading a DHIS2 installation is cumbersome unless you are familiar with the system. In short, the process is comprised of:

- shutting down the web application server
- possibly removing old files (depends on web application server in use)
- copying in the new war-file
- possibly running a set of commands on the database server to upgrade the database schema (depends on the version of DHIS2)
- starting the web application server up again

The observant reader may note that these steps do not cover upgrading the separate components DHIS2 is based upon (notably database and web application server), this was intentional, as our focus was on upgrading just DHIS2 itself.

Regarding the step marked "possibly", this stems from the fact that from Milestone Release 5 and upwards, each release of DHIS2 so far has also required changes to the underlying database schema. Ranging from the relatively simple act of changing a few column names in the M4 to M5 transition, to the following for the M9 to 2.0 final release:

```
drop table aggregateddatavalue;
drop table aggregatedindicatorvalue;
drop table extendeddataelement;
drop table importobject;
drop table importdatavalue;

create index crosstab on datavalue (periodid, sourceid);

alter table indicator modify column numerator text;
alter table indicator modify column denominator text;
```

After the system has been started at least once:

```
update datavalue set categoryoptioncomboid=(select categoryoptioncomboid
from categorycombos_optioncombos where categorycomboid=(select
categorycomboid from categorycombo where name='default'));
```

CodeExample 22: Database schema changes for DHIS2 M9 -> 2.0 transition

This code will drop several tables in the DHIS2 database, create a new index and alter two other table descriptions. This example also adds an additional layer of complexity, in that one of the SQL commands has to be run after the system has been started again.

While this may not seem very difficult from a technical standpoint, and indeed it is simple copy and paste if you know what you are doing, it is something that needed to be done in each and every clinic where DHIS2 was installed. In Kerala, this was far too complex for anyone but the senior system facilitators to do, meaning it will also be extremely time-consuming as these few people will need to travel around and perform the operation over and over.

An additional concern of ours was the risk of data-loss, the worst-case scenario of any upgrade-process. When doing changes to the data model, working directly on the database is required, which means that simple typing errors could result in losing valuable data. As computers are humans superior with regards to repeating the exact same task over and over, this is an issue that should be handled by an upgrade routine, in line with the project specification mentioned in chapter 1.1.

Consequently, there existed great motivation for the creation of tools intended to ease this transition from one release to the next. Again looking at Kerala, this could potentially allow the more junior facilitators to handle the upgrade process themselves, leaving the senior facilitators free to spend their time on more important issues. The junior facilitators would through their weekly routines pass by each clinic in their area and could simply do the updates when they were already there.

To simplify this process, we wrote a small script, *create_upgrader.bat*, to make it easy to create upgrade CDs so the local developers could make whatever changes necessary to the various part of the system, and then easily push these changes to the clinics:

```
@echo off

call variables.bat

mkdir %dhis2_dir%\upgrader

cls

echo ********************************************************
echo Creating upgrade-file, this may take a few minutes...

cd /d %dhis2_dir%
del /q %dhis2_dir%\upgrader\*.* 2> nul

tools\zip.exe -r -q upgrader\upgrade.zip tomcat\webapps\dhis.war db.sql dhis\ra

copy %dhis2_dir%\tools\upgrader.bat %dhis2_dir%\upgrader\upgrader.bat 2>
nul
copy %dhis2_dir%\tools\unzip.exe %dhis2_dir%\upgrader\unzip.exe 2> nul
copy %dhis2_dir%\dhis\autorun-upgrader.inf %dhis2_dir
%\upgrader\autorun.inf

echo.
echo Upgrader made. You can find it at %dhis2_dir%\upgrader\
echo ********************************************************
pause
```

CodeExample 23: Script for creating upgrade routines for DHIS2

Upon running this script from a updated dhis2 installation (based on the installation routine we produced in Kerala), a set of files would be placed in c:\dhis2\upgrader which can be recorded to a CD to produce a CD that automatically upgrades a DHIS2 installation upon insertion. The only thing that requires manual attention is the sql-file with the database schema changes, as this will be different for each DHIS2 version, and cannot as such be automatically captured.

The upgrade script itself looked like this:

```
set drive_letter=c
set dhis2_dir=%drive_letter%:\dhis2


cls
echo ********************************************************
echo.
echo This will upgrade any previous installation of DHIS2 on
echo %dhis2_dir%. If you dont want to upgrade it, please close this window.
echo.
echo ********************************************************
pause

echo.
echo Stopping Tomcat service.
net stop tomcat6

echo.
echo Removing old files
rd /q /s %dhis2_dir%\tomcat\webapps\dhis\
rd /q /s %dhis2_dir%\dhis\ra\

echo.
echo Unzipping, this might take a few minutes...
unzip -q -o upgrade.zip -d %dhis2_dir%
echo Unzipping done.

echo.
echo Updating mysql-database, this will take a few minutes...
%dhis2_dir%\mysql\bin\mysql -u root dhis2 < %dhis2_dir%\db.sql
del db.sql

echo.
echo Starting Tomcat again, this might take a few minutes...
net start Tomcat6

echo.
echo Upgrade Complete!
echo ********************************************************
pause
```

CodeExample 24: Script to upgrade DHIS2 installations in Kerala

This script requires a zip file called *upgrade.zip* to be present, which in turn needs to contain the new war file, new XML files for the local reports and an SQL file named *db.sql* with the changes to the database schema. All of this was of course created by the aforementioned *create_upgrader.bat*. When run, it would then perform the same actions mentioned in the check list at the start of the sub chapter, stopping the web application server, removing old files, unzipping the new files, import the new database schema and start the web application server again.

The intention was that these upgrade-scripts would make life easier on the facilitators when upgrades needed to happen, since there was after all around 80 clinics to push the changes to. Unfortunately, as the next chapter will describe, this was no success.

4.3.7 Upgrading DHIS2 Releases in Kerala

When we were implementing DHIS2 in Kerala in September/October 2007, we were installing version a pre-release version of DHIS2 2.0 - Milestone 8. Just two months later Milestone 9 was released, and there was a wish to upgrade to this version because of the additional features it provided:

> The reason to install/update M9 is we fixed the bugs in reports, bugs in Dashboard are fixed, ValidationAnalysis functionlity is added, customised dataentry functionality is added.
>
> *Mail 5: To me from a Trivandrum HISP developer, subject: Re: Exceptions with Installer, date 24.01.2008*

Again pointing out one of the challenges with using a non feature complete open source system in production. Ideally, once you have something set up and working, the need for continually updating the system would be small, but in the case of DHIS2 where each release so far has added new sought-after features, upgrading often has been important.

However, there were some issues with this transition, highlighting some problems regarding communication between development teams and local teams. While the local team had a fair grasp of how to set up DHIS2 for use, they were not 100% familiar with the solution we came up with for use in Trivandrum. Unfortunately, they went right ahead with the upgrade without much testing though, leading to a rather upset mail in January when they ran into problems. Excerpt:

Can you give me quick solution ,because the dataentry will become delayed in the field because of this exception.

Mail 6: To me from a Trivandrum HISP developer, subject: Exceptions with Installer, date 23.01.2008

In other words, the problems with the upgrade were leading to delays in the data-gathering process. This communication was done in private email, so when it turned out to be more complex than I could answer personally, I did what's usual in the project, and indeed OSS projects in general, namely asking the Kerala team to take it to the developer mailing list.

As circumstances would have it, that was not the way it should have been handled, as the Kerala-team never did, and I didn't catch the fact that nothing ever materialized on the mailing-list. This reluctance of the HISP Kerala people to contact the developers mirrors the experiences recorded by L. H. Øverland (2007) when working with implementing DHIS2 in Vietnam, and at this particular point in time severely hampered the Kerala project.

Consequently, when one of the other developers visited Kerala about a month later to test out the 2.0 beta, he was completely taken aback by the problems he was faced with:

My day starts by first visiting one of the PHCs. And the first punch I got is - "you came to test version 2.0 while we are still using M8 simply because we are unable to upgrade to M9!!!"

I asked why, their reply was "your installer is not working" "even contacting your member doesn't help......." "..........."

Mail 7: To DHIS-developer mailing list from a Core DHIS2 developer visiting Trivandrum, subject: [Dhis-dev] my day in kerala, date 08.02.2008

As it turns out, the problem was extremely minor, but it highlights the importance of knowing your system, and realizing not all components are completely similar. Earlier use of DHIS2 in Trivandrum had all been based on Jetty as the web application server, and naturally this was what the HISP India team was familiar with. However, as outlined above, we ended up using Tomcat for the installation in the 80 clinics, and one big difference between the two in practice, is how they handle war files at startup.

Since war files are basically zipped directory archives with specific content, there is essentially two ways of handling them for the web application server:

- The war file can be expanded at start-up each time (this is what Jetty does), or
- the war file can be expanded on first startup, and subsequent restarts will use the already expanded directory structure (the Tomcat way)

Basically, the first gives you added convenience in that redeploying a new version means simply replacing the war file and restarting, while the latter gives you slightly faster start-up time at the cost of having to delete the existing directory structure to ensure the new war-file is read. This implementation difference was the source of their problems, as the HISP India was not aware of it, and thus still ran M8 from an old expanded directory on a database prepared for M9, despite having correctly replaced the war file.

One final thing of note on this debacle would be the fact that the cause of the problems was identified and solved during the original implementation period in Kerala the autumn of 2007. The upgrade-scripts created then (see the previous chapter or Appendix C) correctly removed the old expanded war-file to ensure compatibility with Tomcat. However these scripts were one of the last things done during our stay in Trivandrum, and the instructions we gave the local team on their use was subsequently rather brief. So when the local team wanted to perform the upgrade in January 2008, they had the choice of trying their hands at creating an upgrade routine with a tool they had only seen in a short presentation, or upgrading the installations manually, a process they were after all quite familiar with. Unsurprisingly, they went with the latter, but by being unfamiliar with the new parts of the setup ended up with the problem described above.

4.4 Next Generation Installers

During the later stages of the Kerala Action Research project, when doing the evaluation and specifying of learning, it became apparent that at least the developers, and some of the system facilitators as well, were not completely comfortable with the system we had designed, as touched briefly upon in chapter 4.3.2. Claims that it seemed unfamiliar, and that more user interaction, which we had specifically designed to avoid, would have been better were brought up. In addition to this, further discussion had been done on the mailing list and among the core developers on the subject mentioned in the HISP developers quote at the beginning of chapter 4.3.3, the complexity of installing DHIS2. A more general solution was wanted, one that could be a model for further releases, not implementation specific, and that would seem familiar to most users.

4.4.1 Action Research cycles

This chapter will also underscore the cycle of action research as illustrated in 3.2.2. The learning specified from the phase of work done in Kerala was diagnosed, and further action based on this was planned, and then taken.

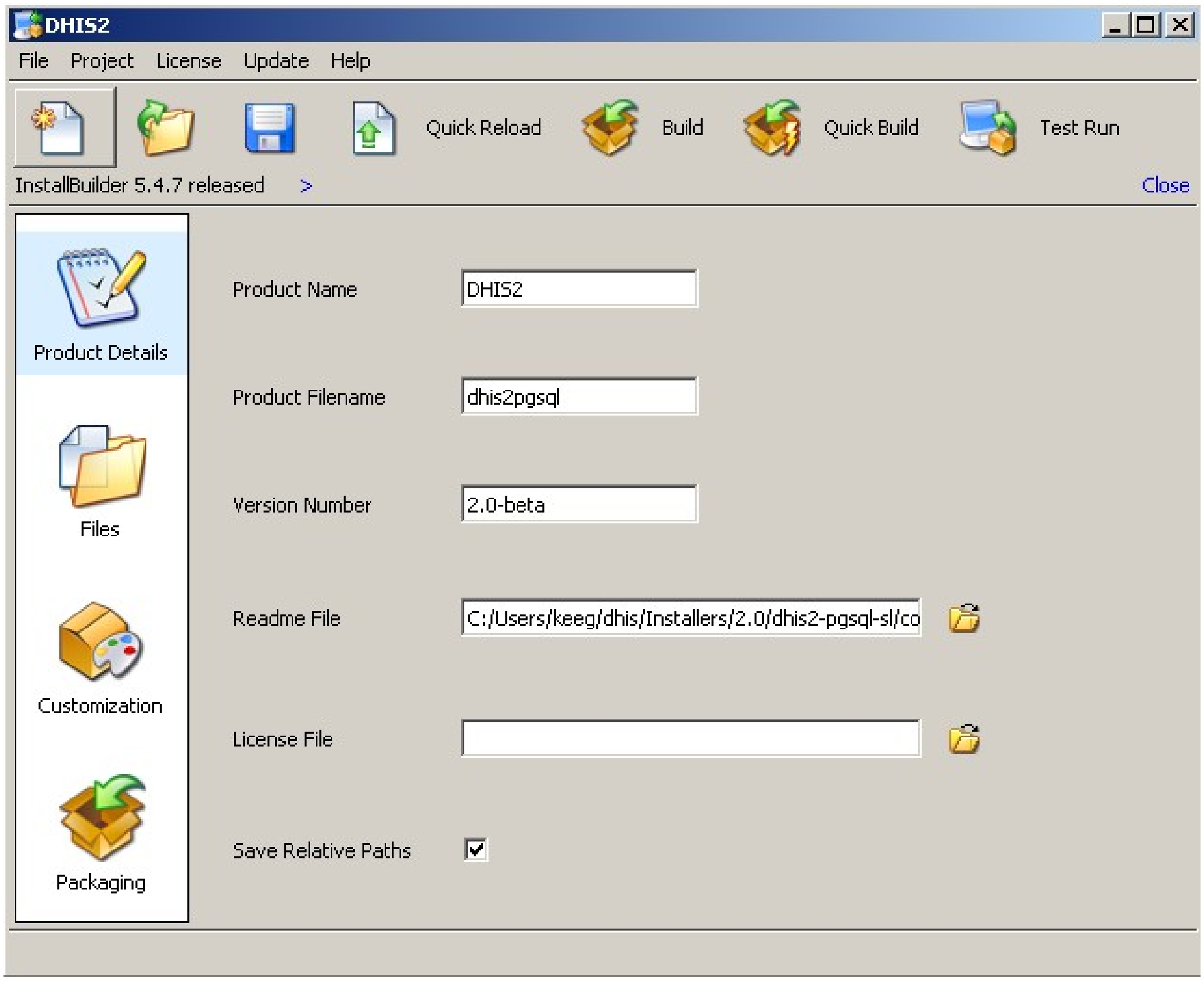
Apart from my work in Kerala, there had as mentioned been other attempts at creating installers for DHIS2 earlier. So another round of evaluation of these with the new focus and new potential user groups in mind was done, while also looking at other new options. For instance, NSIS and Installer2Go were both reviewed again with the new focus and experience in mind, and while both would likely have been adequate for the purpose, the most interesting find came from contact with another open source health-related system, OpenMRS (OpenMRS 2007). As part of their latest release, the OpenMRS project had used BitRock InstallBuilder (BitRock 2007), which is another tool for creating installers. Since there was interest in both DHIS2 and the OpenMRS camp at the time for cooperation, this alternative had intrinsic appeal, in that collaboration could save quite a bit of time.

Additionally, BitRock InstallBuilder seemed to be just as good an option the other alternatives, covering the important issues like easy to use graphical interface for windows installers, and a graphical interface for creating the installers themselves (this we had focused on as an easy to use interface would make it easier to allow new developers to continue using the system after we had finished), but it also had the option of creating Linux- and mac-installers. The latter was less interesting, but there was a growing usage of Ubuntu Linux (Ubuntu 2007) among people interested in DHIS2, so that possibility was very interesting.

4.4.2 BitRock

Regarding BitRock, licensing was an issue. DHIS2 itself is built upon exclusively open source software, and the underlying technologies needed (Tomcat/Jetty, PostgreSQL/MySQL and now Java) are also open source. However, BitRock InstallBuilder is not, but instead offers up free licenses for projects that are open source, much similar to JIRA (JIRA 2007), which was already in use by HISP to keep track of issues in DHIS2. So since there was precedent for non open source software used in connection with DHIS2 and BitRock would not be used to run DHIS2, simply to facilitate easier installation, we deemed that it would be unproblematic to use. The process of obtaining a license was also completely non-problematic when DHIS2 was explained to the BitRock staff.

The program itself is fairly easy to use and intuitive. Based on XML configuration files for defining how to create the installers, most of the possibilities can be configured from a graphical interface, see image below:

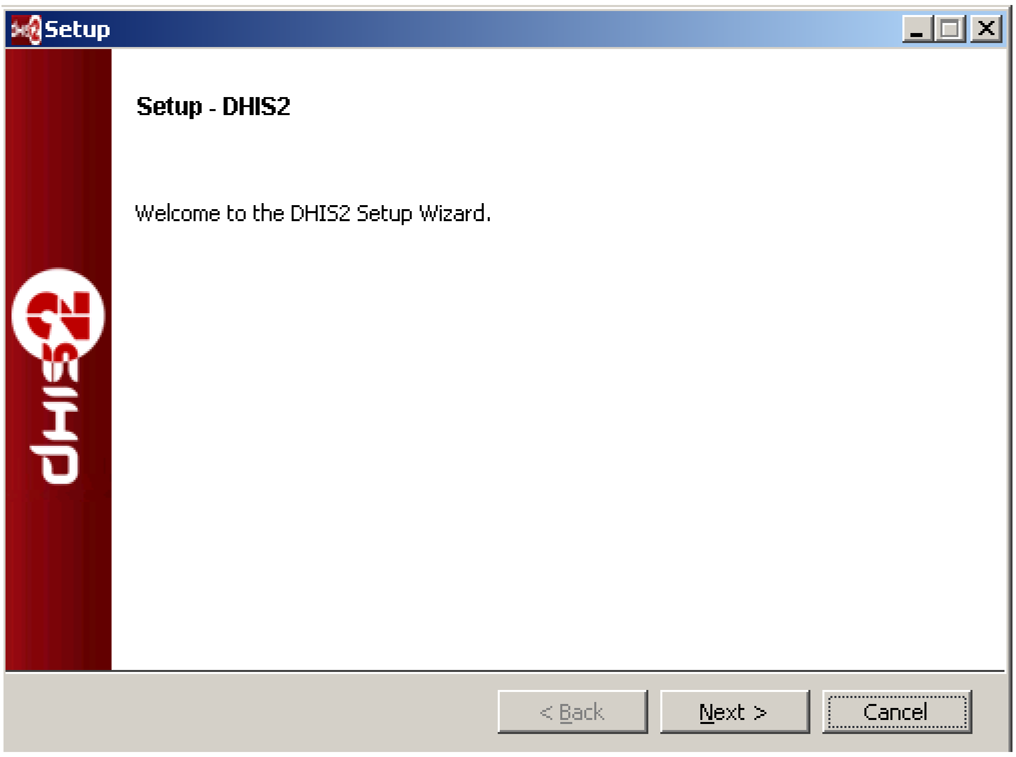


Product Name: DHIS2
Product Filename: dhis2pgsql
Version Number: 2.0-beta
Readme File: C:/Users/keeg/dhis/Installers/2.0/dhis2-pgsql-sl/co
License File:
Save Relative Paths: ✓

Illustration 11: BitRock InstallBuilder Graphical User Interface

However, not every function available, see the reference list included in the supplied documentation, are included in the graphical interface and some of the more advanced features our later iterations includes are among these, and thus had to be manually hacked into the XML. An example of this can be seen in Appendix E.

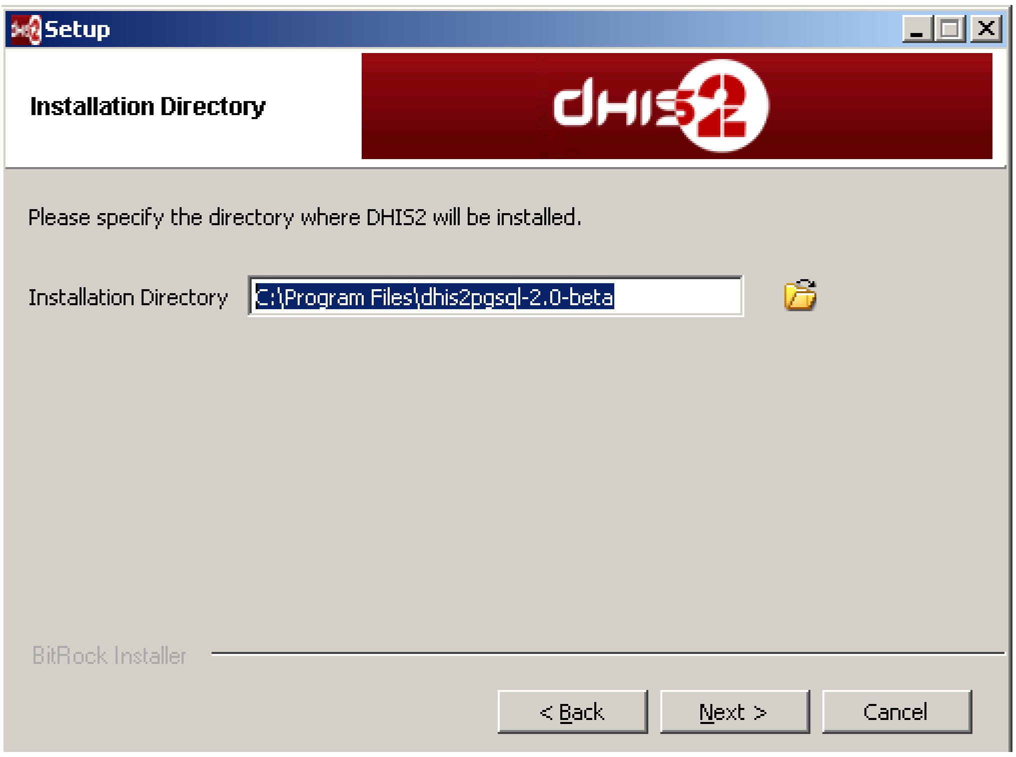
We could thus simply instruct BitRock to include the directory structure defined by the Kerala installations, but this time allowing BitRock to create shortcuts on the Start Menu instead of doing this manually, and define that post installation a script would be run. This script was a small rewrite of the scripts from the installer we made in Kerala, and the scripts used in the Installer2Go-project from Vietnam (for the installers featuring PostgreSQL instead of MySQL), and it was an easy way of setting up services, setting necessary environment variables etc. This way we had ready installers with minimal effort, that work like most «click-next-a-couple-times-then-click-finish» installation programs that windows users should be used to, some example screens are shown below:


Setup - DHIS2

Welcome to the DHIS2 Setup Wizard.

Illustration 12: First Installation Screen

After pressing next, the following is shown:



Please specify the directory where DHIS2 will be installed.

Installation Directory C:\Program Files\dhis2pgsql-2.0-beta

Illustration 13: Second Installation Screen

This part is documented in more detail at the HISP DHIS2 web pages, and in Appendix D.

Upon the release of these installers, I received some quite interesting feedback from one of the HISP developers in India over IM, expressing happiness that I had converted the installer into something that looked like more common Windows installers, and also a wish that this had been how the solution in Kerala had turned out.

Once the initial installers for Windows were created, creating similar installers for Linux was fairly uncomplicated. BitRock takes care of nearly everything, all that is needed is to split up the parts of the installation that is different for Windows and Linux, and create new script for post-install, starting/stopping and uninstallation.

The first was easy in our case, as all that was needed was to add one instance of MySQL/PostgreSQL for Windows, and one for Linux, the rest – Tomcat, DHIS2 itself – is pure Java and thus platform independent (to the extent that the Java Runtime is ported to the platform of course).

Creating Linux shell scripts to replace the Windows batch scripts was also simple, since the logic of what to do was already written, and shell scripts are usually easier to write than batch scripts if you know how to do both (shell scripts simply do not suffer from some of the problems batch-scripting do, notably control logic, you what is commonly referred to as the «GOTO-hell»). Shell script examples can be seen in Appendix H.

4.4.3 Open Source Health Systems Collaboration

One final point to note about the «sudden» need for easier installation of DHIS2 we experienced, was that DHIS2 developers were at the time trying for more collaboration with other projects. OpenMRS has been mentioned, but also OpenHealth (OpenHealth 2008) we had had meetings with, and easy installation of the different systems was a great aid in this process, as it allowed developers to spend less time on installation (like DHIS2, both OpenMRS and OpenHealth have complicated setup procedures). Also, an installer that could contain all of these projects, as they are built on almost identical technologies, was an interesting idea for people wanting to look at the different possible solutions for covering all aspects of their health system needs.

An additionally factor was how it provided developers from the different projects with easier ways to investigate the other projects and ease collaboration. As such, the installers were well received:

> Its great to be able to collaborate this way on a project that crosses so many boundaries!
>
> *Mail 8: Mail from OpenMRS developer to the OpenMRS mailing list, subject: Re: [OPENMRS-DEV] OpenMRS Installation tools, date 10.06.2008*

This idea also highlighted some problems with the new installers. Thus far, they had been of limited interest to most of the developers involved in DHIS2, as they already had it set up and working as they wanted, and had little interest in a separate installation. Once other projects were added into the installers though, the new installers suddenly became interesting to core developers as well, which led to our discovery of issues when the computer being installed to already had some of the components being installed. While the problem itself was not difficult to solve, and arguments could be made that it should even have been anticipated sooner, the only viable solution when keeping the installer self-contained required non-standard installation options, in this case port numbers and service names.

4.4.4 Upgrading Made Easy

Last, but not least, these installers gave the DHIS2 project the possibility to provide easy, generic upgrade-routines. So a second variant of the installer was created, intended to be run on machines that had already had DHIS2 installed by our installers, which would automate this process. This is one of the big advantages of having generic install routines, in that you know how things are set up, and thus how to change them (in this case, name of the underlying services, install-paths, etc.).

The model was of course the upgrade-scripts created for Kerala, but this time with the same graphical interface as the regular BitRock installers. These BitRock-created “upgraders” thus performed the exact same steps as outlined in chapter 4.3.4, but all of it automated, as the installers also set environment variables – here registry entries were also discussed as a possibility, but dismissed due to reasons of simplicity – dictating where DHIS2 was installed, and thus even this information was unnecessary for the user to provide.

Jansen et. Al (2006) define different ways of creating update routines for a software system, with one approach being each vendor creating their own updater, while another would be using a generic tool that through abstraction is intended for use with any software. Our solution falls somewhere in between, through use of a general tool for the update process itself, but heavily modified with scripts for our specific situation. Additionally, one of the drawbacks they associate with this type of update, the ability to roll back changes if necessary (which could potentially be extremely important if it turns out the update does not perform as expected), we circumvented by adding uninstall scripts for the “upgraders”. The last was greatly simplified by the DHIS2 installations modular structure, since parts can be upgraded separately.

After the first announcements of the new installers made with BitRock on the developer mailing list, a flurry of responses was posted. Most were variations over “nice work” or bug reports, but the response from the team in Kerala was perhaps the most interesting for this thesis.

Over IM they conveyed the feeling that these new installers would have been so much better than the installation routines created during the initial roll-out in Kerala. This came as somewhat of a surprise, considering the fact that installing DHIS2 with the Kerala-installer was less time-consuming and required less user interaction, suggesting that it would also be less error-prone. The difference lay in how the BitRock-installer presented them with a more familiar interface, providing the junior facilitators more confidence that they knew what they were doing. The senior facilitators were all familiar with the Windows XP command line, but for the juniors who usually only had a couple of months training in using computer whatsoever, this was strange and unfamiliar.

4.5 Geneva Conference

In the early part of February 2008, HISP was asked by the World Health Organisation (WHO 2007) if they could send two representatives to attend a conference in Geneva. The background for this was an earlier meeting around half a year prior where two of the project developers had presented DHIS2 at the WHO at their bequest, to determine whether DHIS2 was interesting in regards to a software solution WHO was planning on creating to be able to provide developing countries with a complete health IT package. At the time WHO had been interested, and the meeting in February was a follow up of sorts, but this time focusing more on putting the different components together. The author was invited as one of the two representatives, as at the time no one else had spent more time working on installing, setting up, and combining different DHIS2 solutions.

4.5.1 National Health Toolkit

The intention of WHO was as mentioned to be able to provide a complete health IT package, and the representatives at the conference came from different projects:

- DHIS2 was meant used for gathering information and statistics, as well as processing and presenting data, to enable the health workers, notably at the higher levels, to better understand the health situation in the district/region/country

- OpenMRS is «a community-developed, open-source, enterprise electronic medical record system framework intended to aid resource-constrained healthcare environments.". Thus it complements DHIS2 very well, in that together they form a complete software package, tracking health data from patient to the national health ministry, and both before and after the Geneva conference there was highly interesting work being done on integrating the two more closely.

- Inveneo is a non-profit organization producing «highly sustainable and affordable ultra low-power ICT infrastructure technologies designed specifically for organizations that provide vital services -- education, healthcare, economic development -- in remote and rural areas in the developing world." (Inveneo 2008) Thus far they had mainly been providing their hardware solutions for use in the education sector but WHO had expressed their interest in them for use in the health sector in developing countries.

Additionally there was quite a bit of talk of OpenHealth and integrating it as well, especially for their very interesting tools for reporting and viewing data.

4.5.2 Technical setup

The meetings I would characterize as fairly successful. It was quickly determined that none of the software had problems running on Inveneos systems, which had been half the reason for the conference in the first place, and that it technically should pose little difficulties integrating the various components. Of course, this came as no surprise to us regarding DHIS2, as we had had good (though admittedly a tad slow) experience from running it on almost as low-end machines during our stay in Kerala.

Additionally there were several presentations held, introducing the various systems to both the other projects, the people from WHO and at least one beneficial organization interested in possibly funding the project.

On the more technical side, Inveneo are running both Windows XP and Linux on their systems, but due to both performance and a wish from WHO to go with open source software as much as possible, we concentrated most our time on Linux. Here Inveneo run their own distribution, a customized Xubuntu Linux (Xubuntu 2008), which is again a customization of Ubuntu (Ubuntu 2008), with the main difference being the graphical interface, as Ubuntu uses Gnome (Gnome 2008) and Xubuntu uses XFCE (XFCE 2008). Additionally, the extra software of their own design they include with the distribution, are packed as .deb packages (a package format specific to Debian GNU/Linux and derivatives, which includes all variations of Ubuntu) they make available through an APT (APT 2008) software repository.

Since this did not go all that well with our regular install methods – though these did work flawlessly on their machines – we quickly made a similar package while we were there, for easy installation of Tomcat, the DHIS2 war-file and the necessary configuration files. The scripts created as part of this process can be seen in Appendix H. Thankfully it was very easy, since, through the magic of dependencies, you could simply have it depend on the underlying software necessary and that would be installed alongside it when requested. One drawback with this though, was that while apt is an excellent package management tool, it suffers, like other similar tools (Jansen et. al. 2006), from the inability to roll-back packages. This means that separate routines for backup and roll-back would have to be created for this way of distributing DHIS2 to be comparable to our other installers with regards to ease of use.

Chapter 5 Discussion and Conclusion

5.1 Information Infrastructure Growth

Coming back from the empirical data of my research, I will try to summarize how the content of the thesis so far is relevant to the topic of Information Infrastructure Growth, and the sub topic of Frontier Objects and installers. Of note regarding II growth, is how II can be viewed as a federation of information systems, and thus expanding the outreach of the IS is a way to grow the infrastructure. The theoretical framework for this discussion will draw upon several well known concepts from IT theory, all of which are outlined in details in chapter 2 and briefly recapitulated below:

As the information system used throughout the thesis as the basis of my research is a health information system, several articles specific to these types of system have been reviewed, including such issues as capacity building and the creation of local communities for improving sustainability of IS implementations. Because of the open source nature of the HIS in examination, issues more particular to OSS are mentioned as well, such as attracting developers and the differences between OSS and proprietary software regarding installation and upgrading of the system.

Boundary objects are also covered in detail, from their initial conception as a way to describe objects that have different meaning to different actors, moving between them as translators, through how they can aid in standardization of methods, to the expanded concept of Boundary Negotiating Artefacts as a tool for explaining objects that facilitate the pushing and establishing of boundaries. The latter also forms the basis for the concept of Frontier Objects, introduced in this thesis.

The theory introduced also covers the concept of Communities of Practice, as the common learning involved in communities within an open source project such as DHIS2, can be important when discussing the sustainability of an IS implementation, and can thus be very relevant to the II growth. Additionally, Boundary Objects can often be objects moving between communities of practice, providing another interesting point of view.

Finally, theory concerning Installed Base is covered, both how an IS must be fitted to the existing installed base during an implementation process, and how this installed base can be modified to ease further interactions such as upgrades and the like. "Knowledge as infrastructure" is also touched upon, the "installed base" of knowledge in a local implementation can be an important factor to consider.

The specific research questions I originally set out to answer for researching II growth, were as follows:

First Research Objective:
How can simplifying installation and upgrade routines benefit the adoption and maintenance of an open source health information system?

Second Research Objective:
How can standardization and rationalization help Information Infrastructure Growth?

These research objectives were researched using a mix of two different research methods, Action Research and Case Study. The author has for the past year been an active part of a global HIS project, and the ongoing action of creating upgrade and install routines for the IS has provided the empirical data for the research. Case Study of a local implementation effort, while also participating as an actor, has provided the brunt of the material.

5.2 Answering the Research Questions

5.2.1 First Research Question

How can simplifying installation and upgrade routines benefit the adoption and maintenance of an open source health information system?

Installation routines

As alluded to by HISP developers in Kerala (see Mail 3), the installer for DHIS2 was quite a timesaver. While the 30 minutes mentioned is not the most exact estimate of how much time earlier installations used to take, it is clear that less time was now required. However, the perhaps most interesting point is this part of the quote, *"But now by ur installed by just a single enter key everything will be installed",* indicating that we had accomplished our goal of simplicity in the installer, DHIS2 could successfully be installed with minimal user interaction, also by users with less technical skill.

However, our experiences with implementation in Kerala did not reflect this result, as most of the installations were performed by people who could have also installed DHIS2 manually, the only exceptions being a few installations performed by the more technically skilled junior facilitators. As the dozens upon dozens of installations, uninstallations and reinstallations on the HISP office computers and my own laptop can attest to, all of the junior facilitators were perfectly able to install and start up DHIS2, yet very few ever did outside the office.

At least part of this hesitancy to perform installations displayed by the junior facilitators, can be attributed to the weight placed upon including Linux in the local implementation effort, by the core HISP developers, mainly due to Keralas FOSS policy and virus issues. As mentioned in chapter 4.3.3, the added complexity and unfamiliarity introduced by the Linux part of the installation was far more confusing to the facilitators than its windows counterpart, which was quite natural considering that for many these were completely new concepts. Especially the problems with variable computer configuration (covered in chapter 4.3.3) caused issues, as there were too many possibilities to cover in a guide, and explaining all of them turned out to be impossible when the recipients of the explanation did not have the necessary background knowledge needed to understand it. And indeed, in the one case where a facilitator did not manage to complete the Linux installation on his own (see chapter 4.3.4), the Windows installation had still worked without a hitch.

The Linux installations thus caused us quite a bit of headache despite our best efforts to simplify the Wolvix GNU/Linux' Python based installation routines (see Code Example 20). Excerpts of the guide we made for the facilitators to aid in Linux installation can be seen in Appendix G, and is a stark contrast to the Windows installation detailed in 4.3.2. Because of this, it was until close to the end of the implementation phase hard to deem our Linux solution as being worth the effort, at least in the short run. But then the HISP developer doing implementations in parallel with me experienced a situation where Windows XP in a PHC was unusable due to viruses, and as such he simply told the staff to use Linux instead. While this may not justify its inclusion in the process by itself, there were after all other ways to handle the problem, it at least suggests that it was not without merit.

So the reasons why our installer was not as effective at enabling all facilitators to install DHIS2 as it could have been, seems to be social more than technical. The junior facilitators responsible for installations in the PHCs were in addition to having more computer experience, also the ones most confident in their own abilities, and the ones most in contact with me during training. This self-confidence combined with their higher technical skill, giving the senior facilitators more confidence in them, may explain why they alone of the junior facilitators actually “dared” to perform installations, despite the fact that training showed that they all had the ability to do it.

An important thing to note about the project in Kerala, is that there was as mentioned huge overhead in the implementation process, stemming from the fact that travel between PHCs was the most time-consuming part of the process. With this in mind, the effect of saving time on each installation is lessened, while a solution whereby more people could have been involved in the implementation would have had a much more profound effect on the time needed for completion.

However, spending less time on implementing an IS has a secondary effect in a situation such as the Kerala project. The people doing the rollout were also the ones responsible for training and supporting the level below them. When the installations were done by the developers and senior facilitators, this meant they were less available for the junior facilitators, while if the junior facilitators were doing the installation, the more time it took, the less time they would have to spend training and supporting the health staff at the PHC, and vice versa. In this manner, any time that could be shaved of installing could be converted directly into more time for training. While my example here has been installation, as this was the only phase I was present for, the principle applies to any form of maintenance or upgrade of the system as well.

In the beginning, it was my belief that Communities of Practice would be a helpful concept for discussing my empirical data, as it has been in other studies in the same project. As it turns out though, it was not as suitable as first anticipated, but some implications are still worth mentioning:

CoP theory revolves around the principle that much information in a community is passed around simply by sharing experiences or through word of mouth. Kimble et al (2001) goes further, and separates knowledge into “hard” and “soft” parts, where the hard knowledge is information that is written down and documented, while soft knowledge is the information that resides unwritten in the community, and which requires for example a new member to interact with the rest of the community to gain. In Kerala, there was a distinct lack of hard knowledge available for the health workers, and as such almost all their knowledge on use of DHIS2 came from interaction with each other and the facilitators when they came on their weekly visits, and thus more time available for these in the form of less time spent on installation/maintenance, would benefit the community as a whole in the sense that more information would be spread.

I would even go as far as to argue that had we been able to spread the installation out more than we did, both I and the other developers would have spent more time at the office, implicating more time for collaboration and dissemination of knowledge, and as such the failure of upgrades covered in chapter 4.3.7 might have been avoided. Had there been more time to discuss the setup we ended up with and how to upgrade it, the possibility exists that the HISP India developers would not have experienced the problems they did in the M8 to M9 transition. On the other hand, upon my departure there was an understanding that we had managed to go through the work thoroughly, and that the local developers had all the information needed to continue, so it may also have been as simple as a lack of communication.

Thus far I have only discussed how our installers worked in a local environment, where they were created and used for a specific task, but a fair amount of my research was done with a more general approach, researching how installers could growing an information infrastructure. DHIS2 is not a project with a high general appeal, and unlike most other open source projects cannot really expect to attract “random” developers due to the narrow field of interest. Thus our intent was never as much to provide ease of installation to the general public, as it was to be able to promote and showcase the system to potentially interested parties, such as international organisations, health ministries and other open source projects in the field of health IT.

Regarding the first of these, I have outlined our contact with WHO and my part there. However, considering these meetings in hindsight, it is clear that my work was not in any way essential for the process, nor did it have any great impact. The WHO were already interested, and is very likely to have remained just as interested without any form of easy installation. My research did save quite a bit of time in combining DHIS2 with the other software and platform that WHO was planning on using though, and the pre-meeting sharing of install packages may have led to the participants being more aware of how DHIS2 worked.

The topic of open source health systems collaboration is a lot more interesting in relation to my research however. Parallel to my research, work was being done to combine OpenMRS and DHIS2, as well as integrating OpenHealth with DHIS2. Both of these efforts of course required the developers involved to have the respective systems installed, and our combined installers lowered the bar for entry and increased possibilities for collaboration, as Mail 6 shows. They also allowed people who were not directly involved to easily install the projects and thus view the progress and offer insight and criticism, potentially leading to more bugs being discovered (Reymond 1999).

5.2.2 Second Research Question

How can standardization and rationalization help Information Infrastructure Growth?

My work on upgrade routines for DHIS2 is covered in detail in chapters 4.3.5, 4.3.6 and 4.4.4. The configuration possibilities for DHIS2 are huge, with so many different components to chose from when setting up a working system, and so standardization was a key factor in this process. Thankfully, only a few different configurations are normal, but with no knowledge of where or how the various components are located and set up, an upgrade-routine would either have to be very complex with a great amount of checks and tests, or it would have to require enough knowledge from the user that it would be self-defeating.

However, once you rigidly define where parts will be installed, how they will be set up, what the various services will be named, how the program will be run etc., the difficulty of implementing an upgrade-routine goes from very hard to surprisingly easy. Using the example of the M9 to 2.0 upgrade process, it is easy to create a routine that will:

- shut down the web application server
- perform the first steps to upgrade the database schema

- replace the war-file
- remove old files (see the problems with upgrades in Kerala, 4.3.7)
- start the web application server again
- perform the last steps to upgrade the database-schema

provided that you know the name of the database, the names of the services and the location of all the components. In this fashion, standardization enables total automation of the previously cumbersome and time-consuming upgrade process.

But perhaps the most important effect this standardization can have on a project like this, is how it enables central developers to create upgrade routines usable by any local community. While local adaptation is still possible, it should not be needed, because the setup should be identical in each clinic. And considering the issues described regarding updates to the installed systems in Kerala, the advantages of this should be obvious. There was a lack of understanding of how DHIS2 was configured, and the steps necessary to upgrade to a newer release. Had the upgrade routines later created for the project existed at that time, the problematic situation could have been avoided by e.g. a developer in Oslo doing the necessary work and simply providing the team in Kerala with an automated procedure.

Another effect of this is how being able to spread the labour required for upgrading the system to all the system facilitators, instead of each upgrade requiring the presence of a developer / senior facilitator, allows the people with more technical knowledge to put it to greater use by for example providing support where it is actually needed, as opposed to spending a lot of time with mundane repetitive tasks.

That said, the specific issue in Kerala, see chapter 4.3.7 and Mail 4 and 5, was a small detail (from a technical point of view, not in its ramifications), and it could most likely have been avoided just as easy by simply writing a guide detailing each step necessary to complete the upgrade. But seeing as how communications breakdown was part of the reason for why the problems existed in the first place, lessening the level of detail local developers need to worry about may very well be a good thing. Of course, it is important to not carry this to the extreme as you will then just end up with a situation where local authorities do not have the required knowledge in case something does go wrong after all. Though standardization of components and names should minimize the potential problems, it is impossible for global developers to account for every possibility, and in those cases local expertise is important.

Installed base

Mengiste and Nielsen (2006) have as mentioned discussed the importance of being installed base friendly to ensure successful implementation and persistence of an information system. Again expanding a bit on the concept of an installed base from the meaning it has in the business world, "installed base" here includes what already exists of hardware, software and personnel/knowledge. The article also mentions

(...)the installed base can also be draw upon and shaped by human agents(...) (Mengiste and Nielsen 2006)

which gives another interesting point of view regarding maintenance and upgrades, that continued success of an IS that is in a state of fairly rapid change, such as DHIS2, requires the installed base to accommodate this by facilitating upgrades etc.

The standardization discussed above can be seen as modifying the installed base to be "more friendly". For example, after a DHIS2 installation, the IB of the machine in question now includes the software required to run DHIS2, and subsequent releases of DHIS2 would be easier to install. Standardized names and variables would still be required for implementers to take advantage of it, but even without these a knowledgeable person would find the environment a lot simpler to work with for use with DHIS2 or other similar applications, such as OpenMRS or OpenHealth.

In addition to hardware and software, the expanded concept of installed base also encompasses the existing knowledge and personnel available. Using the personnel in Kerala as an example, we originally did our best to chose technologies that could leverage their knowledge as covered in chapter 4.3.2, and it is extremely interesting to note that deviating from this in our choice of Tomcat over Jetty, was part of the failure in the M8 to M9 transition. At least that exact problem is not likely would have occurred had our installers been more installed base – meaning catering to existing knowledge – friendly.

The issue of including Linux in our implementation as covered above is another instance where our choices did not fit with the installed base of personnel skill, and the fact that there was no installed base to capitalize on surely exacerbated the problems we had regarding the junior facilitators and installation.

Catering to the familiar

Continuing the concept of being installed base friendly in more general terms, familiarity was something that came up as my research progressed. As mentioned, with the advent of the BitRock-installers, the previous Kerala-installer was found quite lacking in comparison, by the very team it was designed for. A simpler and faster solution was perceived as less suited to the task, simply because it was less familiar to the users. Granted, the only response we got on the different installers were from those in the HISP India team who were a bit more used to computers, and had thus more experience with "standard" Windows installers, but the issue of familiarity is still an interesting one.

When choosing the interface for our installers in Kerala, great weight was placed upon simplicity to make it easy upon the end users, and leveraging earlier work to speed up the development process. While both are admirable goals in themselves, the ease with which the installer ideas were modified to fit within the BitRock installers and the feedback we received on these, suggest that perhaps laying more emphasis on producing something that would present the users with a more familiar interface would have been even more productive.

On the other hand, the unfamiliarity with the Windows part of our Kerala installers did not mean that they were in any way inefficient for their intended purpose, as the users would merely memorize the process – an easy task considering it was just one simple step – and accept how this installation differed from what they were more used to. The alternatives boil down to a trade off, familiarity for the users versus simplicity and speed in development, assuming the speed of installation is equal (here working under the assumption that non-human factors such as the choice compression algorithm is the largest factor, and that these will be equal regardless of installer type).

5.3 Frontier Objects

As alluded to in chapter 2.2.4, the concepts covered in my empirical data, installation and upgrade routines, do not quite fit within standard theories surrounding objects moving between groups and communities, and consequently I would like to coin the term “Frontier Object” to describe the work done in this thesis. The concept of Frontier Objects, which draws upon ideas from both Boundary Objects and Boundary Negotiating Artefacts, is well suited for a theoretical discussion of the empirical work done as part of this thesis.

This concept has of course evolved throughout the work on and review of my empirical data, and the discussion is thus split into two subchapters. The first details how it relates to BO, BNA and IO and covers the initial definition, while the second ties it more closely to the empirical data and outlines the final definition.

5.3.1 Object Theory

Viewing my empirical data, it is not clear cut how to discuss installers from a theoretical point of view. For example, the concept of an installation routine does not itself fit the criteria of a boundary object. Admittedly it is an object for transmitting information and knowledge between two communities of practice – developers and users being the obvious example – but it is the object contained within, DHIS2, which is the real boundary object, not the container. It is more the means to an end, than an end in itself.

Likewise it does not quite fit within the concept of a boundary negotiating artefact. While it does indubitably share some qualities, it is in conflict with others. Case in point, where Lee (2007) states that BNAs “*Can be largely sufficient for collaboration*”, an installation routine is not itself sufficient for collaboration, rather it is the object it transfers that is the focus of collaboration. On the other hand, an installer is, like BNAs most definitely an artefact capable of crossing boundaries and transmitting information across them. Additionally, it not only can, but is often intended to, negotiate and expand existing boundaries, and arguments could be made for how its main reason for existence would be the creation of new communities connected to the existing via the boundary object it transmits.

In the definition of BNA Lee (2007) states their fluidity is a defining characteristic:

Are fluid: (1) a boundary negotiating artefact can change from one type to another when the *context of use changes; and (2) a boundary negotiating artefact can sometimes also simultaneously be physically incorporated or transformed into another artefact;*

However, FO describes objects that are created to serve a specific purpose, and are therefore not susceptible to change, as this would deviate from the stated purpose, and indeed create a new object. Or in other words, while the object of transfer may very well be fluid in the meaning it conveys, it is hard to argue that the container has divergent meanings to different communities.

Regarding Intermediary Objects, the main problem (as touched upon in chapter 2.2.4) with using it as an idea for explaining my empirical data, is how it is focused upon the object as a representation:

Intermediary objects are also representations. They are either representations of the product or of the design process (Gantt charts, planning, etc.). (Boujut, Blanco 2003)

This idea of representation does not fit the objects that our installers constitute, as they are containers for transmitting an object more than they are descriptions of either design or object.

Frontier Objects do however share a very important property with Boundary Objects, namely the standardization of methods, as keeping information compatible between old and newly created communities in the case of FO is just as important as it is between groups in the case of BO. And, perhaps even more important, the effect standardization of methods has on increasing the reach of BO across divergent worlds (Star, Griesemer 1989, p407) is directly translatable to FO, as expanding their outreach is a simple and powerful way to increase their ability to move boundaries. A good example from chapter 4.4.4 is the way standardization allows for the creation of upgrade routines that can be used across communities without modification for possibly great savings on time.

So the initial definition of a Frontier Object was as follows:

1) Facilitate the pushing and establishing of boundaries
2) Serve a specific purpose, and are not susceptible to change
3) Employ standardized methods for compatibility

4) Has the same meaning across communities

5.3.2 Frontier Objects Defined

The Frontier Object that is the installer from Kerala did not empower the facilitators in the way intended as it, at least partway, failed to push the boundary of who was able to install DHIS2, as shown by how few system facilitators who were actually employing the installers, see chapter 4.3.5 and 5.2.1. Considering the summary of its usage pattern, a reasonable explanation is that it was too complex for its purpose. While the Windows part of the installer was suitably simplistic, and seemed to work as intended for the target users, the Linux part was not, for the reasons covered above, and this lead to the situation where the installer did not as much move the boundaries related to installation as it worked as a timesaver for those already enabled. However, the installer itself fits within my initial definition of FO, leading to the conclusion that the definition itself is flawed, or at least is not precise enough, and thus needs to be expanded upon.

Unlike Boundary Objects, Frontier Objects do not describe objects that cross the boundaries between communities, nor serve as translators/mediators between them, but rather objects that move these boundaries and possibly create new communities, much like Boundary Negotiating Artefacts. However, where Lee (2007) defines that BNA "*May seem ''effortful'' in use as opposed to effortless",* it is now apparent that a defining property of FO needs to be their simplicity in use, that they are effortless to the involved participants, as shown by how the difference in effort between the Windows and Linux part of the installation was so off-putting to the facilitators.

This concept of simplicity impacted other parts of my research as well, for example the sudden effortless way developers on different open source projects could install and view each others work (see chapter 4.4.3) helped open the way for closer collaboration and cooperation. Not that this would not have happened in either case, but as e.g. Mail 6 shows, the installers had some function in speeding up the process through lowering the barrier to entry.

One of the fundamental characteristics shared between Boundary Negotiating Artefacts and Frontier Objects, is how they facilitate the pushing and establishing of boundaries. In Lees

(2007) definition of BNA though, there is an added concept of division of labour, that was not present in my original definition of FO. Indeed, the article states

A great deal of boundary work is concerned with discovering, testing, and pushing of boundaries (e.g. attempting to modify division of labour)

In a way, this is exactly what has been the result of a lot of my research, the division of the labour required for implementing, upgrading or similar has been modified, work that previously had to be done wholly on the local level can now be done at least partly on the global level, etc. This division of labour also links back to the aforementioned barrier to entry, in that by leaving more of the work with installers upon the global team, there is a lesser need for knowledge and skill in the specifics of installation for each person setting it up. An side-effect of this again is how it undermines my original idea that Frontier Objects, like Boundary Objects aid in the transmission of knowledge, rather the fact is that FOs lessen the need for knowledge dissemination in the first place. The ability of even junior facilitators with little computer knowledge to install DHIS2 through our research, is testament to this.

Going back to the characteristics from last chapter, that Frontier Objects have the same meaning across communities was originally based on the idea that installers have the same purpose regardless of who the current user happens to be, and how this differs from the fluidity of meaning portrayed by BOs and BNAs. However, over the course of my research, it became clear that while the purpose of the installers themselves may not have changed, the perception of what the installers were doing, did. From easing implementation of an IS, through providing easy access to users, personal and more official alike, to aiding in collaboration between projects, the perception of the Frontier Object varied with application, directly conflicting with point 4 of the initial definition.

Thus I propose Frontier Objects as objects that

- facilitate the pushing and establishing of boundaries
- serve a specific purpose, and are not susceptible to change
- employ standardized methods for compatibility
- are simple and effortless in use

- lowers the barrier to entry/lessens the need for knowledge transmission

5.4 Conclusion

Despite quite a few problems and failures, mainly in the first part of my research, I would heartily recommend anyone developing an Information System, health related or otherwise, to look into working on installation and upgrade routines from an early stage. The benefits are more obvious in an open source setting such as the one described in this thesis (exposure and recruitment of developers being key), but the pros of easing installation, maintenance and management apply in general terms. However, a few caveats exist, outlined in the following.

5.4.1 Leverage the Installed Base

Echoing Mengiste and Nielsen (2006), I would recommend anyone doing further research on installers to place great emphasis on being installed base friendly, and in particular leverage the existing knowledge and skill. The biggest issues we faced in our implementation can all be at least partly attributed to choosing solutions that were not familiar to all the people involved. People generally do better when they feel confident that they know what they are doing, and small problems can be solved on the spot if the technology employed is known to the implementer, instead of ending up as larger problems.

Additionally, shaping the installed base – whether by standardizing software solutions or increasing the knowledge base through training – as appropriate in the process, is likely to produce great returns when it comes to maintenance or upgrades of the IS in question.

This friendliness is thus actually double-sided, in that it is preferable to be friendly to your users in their contact with installers and upgraders, but also friendly to the installed base.

5.4.2 Communication and Local Capacity is Still Key

A final note, while I recommend automation and standardization for installation and upgrade routines for increasing II growth, it is important to remember that no matter the level of automation there will still be people in each end of the system, and that good communication between them is essential. I have already explained how lack of communication lead to perhaps the biggest problems that Kerala has faced in its first half year of production use of DHIS2, and in those situations it is as mentioned crucial that the local developers have the required expertise, necessitating good communication of what is actually happening in the automated routines.

This is essentially the same thing covered in many previous HISP articles – concerning how building local capacity is essential for the sustainability of an IS (Øverland 2006; Fossum 2007) – but I wish to underscore that even if new developments, such as what has been covered in this thesis, allow more automation in installation and maintenance than previously possible and as such may lead to a perception that less local capacity and knowledge is needed, these lessons are not forgotten. Global to local aid may be improved upon in this manner, but it cannot replace local expertise. Coming back to the discussion on CoP, there is a lot of soft knowledge in most communities, and as soft knowledge is among the hardest to share without physical contact (Hemetsberger, Reinhardt 2004), local presence is important.

5.4.3 Simplicity

The arguments I put forth regarding simplicity as a defining characteristic of Frontier Objects is the final point I wish to underscore for anyone doing further research or work on either FOs or installers. Keeping everything as simple as possible has been a great boon for development – for my own sake and for lowering the barrier to entry for other developers – for acceptance of the work in the project and for its actual usefulness. This goes of course for both the interface to the potential user, and the choices regarding implementation and technologies.

However, simplicity can be drawn too far, as my comments on our initial installers in Kerala being so simple they ended up being unfamiliar to users show. While simplicity is a goal in itself, it should be moderated so that it does not conflict with the aforementioned issues of being installed base friendly, or drawing upon local capacity.

List of Acronyms

BO – Boundary Object

BNA – Boundary Negotiating Artefact

CD – Compact Disc

CoP – Communities of Practice

DHIS – District Health Information System

FO – Frontier Objects

FOSS – Free Open Source Software

FS – File System

GPL – GNU General Public License

HIS – Health Information System

HISP – Health Information Systems Programme

IB – Installed Base

II – Information Infrastructure

IO – Intermediary Object

IM – Instant Messaging

IS – Information System

IT – Information Technology

LLS – Linux Live Scripts

MPL – Mozilla Public License

MS – Microsoft

NTFS – New Technology File System

OS – Operating Systems

OSS – Open Source Software

USB – Universal Serial Bus

WHO – World Health Organisation

References

Avison D., Lau F., Myers M. and Nielsen P. A. "Action Research" *in Communications of the ACM, Vol. 42, No. 1,* January 1999

Ballintijn G. "A case study of the release management of a health-care information system" *in proceedings of the IEEE International Conference on Software Maintenance, ICSM2005, Industrial Applications track*, 2005.

Baskervill R. L. "Investigating Information Systems with Action Research", *Communications of the Associations for Information Systems, Volume 2, Article 19*, 1999 (Direct link: http://www.cis.gsu.edu/~rbaskerv/CAIS_2_19/CAIS_2_19.html)

Baskervill R. L. and Wood-Harper A. T. "A Critical Perspective on Action Research as a Method for Information Systems Research", *in Qualitative Research in Information Systems: A Reader* 2002 8:129-145

Benbasat I., Goldstein D. K., Mead M. "The Case Research Strategy in Information Systems Studies" *in MIS Quarterly,* 1987

Bonaccorsi A., Rossi C. "Why Open Source software can succeed" *Research PolicyVolume 32, Issue 7, , Open Source Software Development*, July 2003, Pages 1243-1258. (Direct link: http://www.sciencedirect.com/science/article/B6V77-48B5R73-2/2/674046d38fa92b909dc138d28561ce99)

Boujut J-F., Blanco E. "Intermediary Objects as a Means to Foster Co-operation in Engineering Design"*Computer Supported Cooperative Work 12: 205–219*, 2003

Boujut J-F., Laureillard P. "A co-operation framework for product–process integration in engineering design" *Design Studies Vol 23 No. 5* September 2002

Braa J., Monteiro E., Sahay S. "Networks of Action: Sustainable Health Information Systems Across Developing Countries", MIS Quarterly, pp. 337-362, 2004.

Brucker, Ø. F. “Internationalization and localization – A case study from HISP” Master thesis - 01.08.2007

Dubé L., Paré G., “Case Research in Information Systems: Current Practices, Trends, and Recommendations” *Cahier du GReSI no 01-12* August 2001

Erenkrantz J. “Release Management within Open Source Projects” *Proc. 3rd Workshop on Open Source Software Engineering, 25th Intern. Conf. Software Engineering*, Portland, OR, May 2003

Farrell J., Saloner G. “Installed Base and Compatability: Innovation, Product Preannouncements, and Predation” *The American Economic Review* Vol. 79, No. 5 (Dec. 1986), pp. 940-955

Fossum K. “Social Construction of Legacy Systems: A case study from a health information systems development project in Mozambique” Master thesis, June 2007

Gal U., Yoo Y., and Boland R. J. “The Dynamics of Boundary Objects, Social Infrastructures and Social Identities” *Sprouts: Working Papers on Information Environments, Systems and Organizations*, Volume 4, Issue 4 2005, pp 193-206. http://sprouts.case.edu/2004/040411.pdf

Hanseth O. “Knowledge as infrastructure”. In: C. Avgerou, C. Ciborra and F. Land, Editors, *The Social Study of Information and Communication Technology*, Oxford University Press, Oxford (2004).

Hanseth O., Monteiro E., Hatling M. “Developing Information Infrastructure: The Tension between Standardization and Flexibility” *Science, Technology, & Human Values, Vol. 21, No. 4*. (Autumn, 1996), pp. 407-426.

Hanseth O., Aanestad M. “Design as Bootstrapping: On the Evolution of ICT Networks in Health Care” *Methods of Information in Medicine* 42: 385-391. 2003.

Harvey F., Chrisman N. “Boundary objects and the social construction of GIS technology”

Environment and Planning A 1998, volume 30, pages 1683-1694

Hemetsberger A. , Reinhardt C. “Sharing and creating knowledge in Open Source community – The case of KDE” *In: The fifth European conference on Organizational Knowledge, Learning and Capabilities* (2004)

Jansen S., Brinkkemper S., and Ballintijn G. “A process model and typology for software product updaters” *In Proceedings of the 9th European Conference on Software Maintenance and Reengineering (CSMR 2005)*, Manchester, United Kingdom, Mar. 2005.

Kimble C., Hildreth P., and Wright P. “Communities of practice: Going Virtual” *In Knowledge management and business model innovation.,220-234* London: Hershey. 2001

Lakhani K., Wolf B.,Bates J., DiBona C. “The Boston Consulting Group Hacker Survey Boston”, The Boston Consulting Group, 2002

Lave J., Wenger E. “Situated Learning: Legitimate Peripheral Particiapation” 1991

Lee C. P. “Between Chaos and Routine: Boundary Negotiating Artifacts in Collaboration” *In H. Gellersen, K. Schmidt, M. Beaudouin-Lafon, W. Mackay (eds): ECSCW 2005: Proceedings of the Ninth European Conference on Computer Supported Cooperative work,* Paris, France, 18–22 September 2005

Lee C. P. “Boundary Negotiating Artifacts: Unbinding the Routine of Boundary Objects and Embracing Chaos in Collaborative Work” *Computer Supported Cooperative Work* (2007) 16:307–339

Madey G., Freeh V., and Tynan R. “The Open Source Software Development Phenomenon: An Analysis Based On Social Network Theory” *Americas Conference on Information Systems (AMCIS2002),* 2002

Mengiste S. A. and Nielsen P. “Scaling Health Information Systems in Developing Countries: On the Essence of Being Installed Base Friendly” Proceedings of the 29th edition of IRIS,. Helsingør, Denmark. (2006)

Nguyen T. N. “OSS For Health Care in Developing Countries: Comparative Case Studies of DHIS2 and Patient Based Systems In Ethiopia And Vietnam” Master Thesis – 2007

Nordal K. “The Challenge of Being Open – Building an Open Source Development Network” Master Thesis 31.06.2006

Pawlowski S. D., Robey D., and Raven A. “Supporting Shared Information Systems: Boundary Objects, Communities, and Brokering” *21st International Conference on Information Systems*, Atlanta, GA, Association for Information Systems. 2000

Preece, Rogers and Sharp “Interaction Design, beyond human-computer interaction” 2007

Raymond E. S. “The cathedral and the bazaar”. *Available at* *http://catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/* 1999

Rolland K. H. “Challenging the Installed Base: Deploying a Large-Scale IS in a Global Organization” *In: Hansen, H.R., Bichler,M.,Mahrer, H. (Eds.), Proceedings of the Eighth European Conference on Information Systems, Vienna*, pp. 583–590. 2000.

Schaumann J. “Pondering Live CDs” *Available at* http://www.netbsd.org/~jschauma/nblivecds.pdf 2006

Star S. L., Griesemer J. R. “Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39” *in Social Studies of Science*, Vol. 19, No. 3. (Aug., 1989), pp. 387-420.

Strandli S. A. “Exploring the use and effect of Intermediary Objects in a global Open source organisation” Master Thesis 2008

Wenger E. “Communities of practice and social learning systems” *Organisation* **7** 2, pp. 225–246 2000

Øverland L. H. “Global Software Development and Local Capacity Building: A means for

Internet references

APT 2008
http://www.debian.org/doc/manuals/apt-howto/

ASL 2008
http://www.apache.org/licenses/LICENSE-2.0.html

BitRock 2007
http://bitrock.com/products_installbuilder_overview.html

BSD 2008
http://www.opensource.org/licenses/bsd-license.php

Debian Live CD 2008
http://wiki.debian.org/LiveCD

Debian MySQL 2008
http://packages.debian.org/search?
keywords=mysql&searchon=names&suite=stable&section=all

DHIS2 CD/USB 2008
http://www.hisp.info:8080/display/RandD/CD+and+USB

DHIS2 Linux 2008
http://www.hisp.info/dhis2/wiki/Linux

DHIS2 Milestones 2008
http://www.hisp.info:8080/display/DHIS2/DHIS+2.0+Milestones

Dreamlinux 2007
http://www.dreamlinux.com.br/

Etch 2008:
http://www.debian.org/releases/stable/i386/release-notes/ch-whats-new.en.html
Release Notes for Debian GNU/Linux 4.0 ("etch"), Intel x86

Fedora Live CD 2008
http://fedoraproject.org/wiki/FedoraLiveCD

FS Driver 2008
http://www.fs-driver.org/

Gnome 2008
http://www.gnome.org/

GPL 2008
http://www.gnu.org/licenses/gpl-2.0.html

HCMC Installer 2008
http://www.hisp.info/confluence/display/HISP/HCMC+DHIS2+Installation

Hibernate 2007
http://www.hibernate.org/

Hibernate2 2007
http://www.hibernate.org/80.html
Databases supported by Hibernate

HSQLDB 2008
http://hsqldb.org/

Installer2Go 2007
http://www.dev4pc.com/installer2go.html
Installer2Go - powerful, easy to use, highly interactive tool for creating reliable windows installer packages that meet the Windows 2000/XP logo certification guidelines.

Inveneo 2008

http://www.inveneo.org/

JBoss 2008

http://www.jboss.org/jbossas/

Jetty 2007

http://jetty.mortbay.org/

JIRA 2007

http://www.atlassian.com/software/jira/

Kerala State 2007

http://www.technopark.org/downloads/ITPolicy-2007.pdf

KISS 2007

http://en.wikipedia.org/wiki/KISS_principle

Knoppix 2007

http://www.knoppix.net

Knoppix Remastering Howto 2007

http://www.knoppix.net/wiki/Knoppix_Remastering_Howto

Linux NTFS Project 2008

http://www.linux-ntfs.org/doku.php

LLS 2007

http://www.Linux-live.org/

Maven 2007

http://maven.apache.org/

Software project management and comprehension tool

Maven Jetty Plugin 2008

http://mojo.codehaus.org/jetty-maven-plugin/usage.html

MPL 2008

http://www.mozilla.org/MPL/MPL-1.1.html

Mysql 2007

http://www.mysql.com/

MySQL AB :: The world's most popular open source database

MySQL Download Page 2008

http://dev.mysql.com/downloads/mysql/5.0.html#downloads

OpenHealth 2008

WHO funded project, powering e.g. this website: http://www.who.int/whosis

OpenMRS

http://openmrs.org/wiki/OpenMRS

OpenOffice.org 2008

http://www.openoffice.org

OSS 2008

http://www.opensource.org/docs/osd

Postgresql 2007

http://www.postgresql.org/

PostgreSQL: The world's most advanced open source database

Puppylinux 2007

http://www.puppylinux.org/

Resin 2008

http://www.caucho.com/

Slax 2007

http://www.slax.org

Simpleslax 2007

http://scattershot.wickedtribe.org/

Sun War 2008

http://access1.sun.com/techarticles/simple.WAR.html

Tomcat 2007

http://tomcat.apache.org/

Ubuntu 2008

http://www.ubuntu.com/

Ubuntu Live CD 2008

https://help.ubuntu.com/community/LiveCD

WHO 2008

http://www.who.int/

World Health Organization

Wolvix 2007

http://wolvix.org/

Wolvix Remastering Documentation 2007

http://wiki.wolvix.org/UsingModules

Wolvix Review 2007

http://on-disk.com/cms/index.php?wiki=Wolvix110

XFCE 2008

http://www.xfce.org/

Xubuntu 2008

http://www.xubuntu.org/

Zenwalk 2007

http://www.zenwalk.org/

Zenwalk Remastering Documentation (2007):

http://www.zenwalk.org/modules/tinycontent/index.php?id=20#mozTocId354787

Appendix A DHIS Linux Live CDs

This is a copy of the documentation located at http://www.hisp.info/dhis2/wiki/WolvixDHIS

A.1 Creating a Wolvix Live CD containing the DHIS software

I started up from a Wolvix Cub CD (the Cub version is a lot smaller than the Hunter version and I at first thought it would contain all we need, but in retrospect it would've been easier to start with Hunter and trim it down) and copied the contents of /mnt/live/mnt/hdb/ (where hdb is my cdrom) to my harddisk:

```
cp -af /mnt/live/mnt/hdb/ /remaster/
```

Alternatively you could just copy the contents of the CD when booted into a different operating system, the contents of /mnt/live/mnt/hdb/ is simply a mirror of what's on the CD, but being in the liveCD environment ensures you have all the necessary tools.

Then I needed to change this CD-image to include java, mysql, tomcat and the DHIS war-file. Since I already had a .deb archive lying around on my debian installation, I simply used deb2lzm (a tool to convert debian .deb archives to lzm-modules, the modules all newer SLAX versions use) on that:

```
deb2lzm jdk-1-5.deb java-1.5.lzm
```

If you don't have one lying around (who does?), just download one from http://java.sun.com, any of the packages should be easily convertable with either tgz2lzm, rpm2lzm or deb2lzm.

Mysql was a bit more tricky, since I couldn't find any recent mysql-module, so I simply grabbed the servers-module from a recent SLAX Server Edition (http://www.slax.org/download.php), and got mysql from there:

```
lzm2dir servers.lzm tmp/
```

This extracts the servers lzm-module to a directory tmp/. I then looked through tmp/var/log/packages/:

```
ls tmp/var/log/packages/
```

to see which packages this module contained, and removed everything but mysql like this:

```
ROOT=tmp/ removepkg dhcpcd sendmail bind ...
```

Once mysql was the only package left, I repackaged it:

```
dir2lzm tmp/ mysql.lzm
```

(Note that this is the easiest way to trim down wolvix. Unpack a module, see what subpackages are inside, remove the ones you don't want, and repackage.)

At this point I actually added this one module and remastered Wolvix and rebooted from this remaster. It's by far the easiest way to add the DHIS2-tables to the mysql.lzm. After the reboot you will have mysql running, and then you simply do as described on http://www.hisp.info/confluence/display/DOC/Installing+DHIS+2 to add the DHIS2-tables:

```
mysql -u root -p
```

(prompts for a password, the default of which is "toor", the same as the default Wolvix root-password, should be changed...) Once you're in mysql:

```
mysql> create database dhis2 default character set utf8 collate
utf8_general_ci;
mysql> grant all on dhis2.* to dhis@"localhost" identified by '';
```

Then add the changes to the module:

```
lzm2dir /remaster/wolvix/modules/mysql.lzm tmp/
cp -af /var/lib/mysql/* tmp/var/lib/mysql/
dir2lzm tmp/ /remaster/wolvix/modules/mysql.lzm
```

I also grabbed the latest Slackware Openoffice.org package from http://www.slacky.eu (http://www.slacky.eu/repository/slackware-11.0/office/openoffice/2.2.1/en-us/), and converted it to a lzm-module:

```
tgz2lzm openoffice-2.2.1.tgz openoffice-2.2.1.lzm
```

These three modules I then added to the directory where wolvix looks for additional user-added modules:

```
cp openoffice-2.2.1.lzm /remaster/wolvix/modules/
cp mysql.lzm /remaster/wolvix/modules/
cp java-1.6.lzm /remaster/wolvix/modules/
```

(This is one point where starting with Hunter would have helped, it already has an OpenOffice? lzm-module, which I didn't see till I'd repackaged the Slackware tgz...)

Then I figured out which wolvix module contained users and home dirs and edited that one a bit:

```
lzm2dir /remaster/wolvix/base/002_core.lzm tmp/
```

Wolvix normally has just root added as a user, which is not really all that great for our purposes, so I added a normal user. When already booted into a wolvix environment, you can just add a user like on any normal Linux system, and it'll update /etc/passwd & /etc/shadow accordingly, as well as create /home/username. So I added a user dhis:

```
adduser dhis
```

and copied the changes back to the base module so these would show up next time:

```
cp /etc/passwd tmp/etc/passwd
cp /etc/shadow tmp/etc/shadow
cp -a /home/dhis/ tmp/home/
```

Then I added tomcat. This required making a lzm-module from scratch, which is a lot easier than it sounds. Simply make a base directory to work in

```
mkdir tmp
```

and make the necessary folders under this:

```
mkdir -p tmp/etc/rc.d/
mkdir -p tmp/opt/
```

Unzip the latest tomcat (6.0.14 at the time of writing):

```
cd tmp/opt/
unzip apache-tomcat-6.0.14.zip
mv apache-tomcat-6.0.14 tomcat
```

I then added the hibernate.properties file described at http://www.hisp.info/confluence/display/DOC/Installing+DHIS+2 under tmp/opt/tomcat/lib/. I also added this startup script (which was really easy since tomcat comes with such nice bash scripts) in tmp/etc/rc.d/rc.tomcat, and modified the wolvix startup scripts to start and stop tomcat as appropriate:

In /etc/rc.d/rc.M add

```
# Start the Tomcat server
if [ -x /etc/rc.d/rc.tomcat ]; then
  /etc/rc.d/rc.tomcat start
```

```
fi
```

and in /etc/rc.d/rc.0 add

```
# Stop the Tomcat Server
if [ -r /var/run/tomcat.pid ]; then
  /etc/rc.d/rc.tomcat stop
fi
```

(these files are located in 002_core.lzm, so you need to change that as described above).

At this point, you could remaster the cd and have a working liveCD. Assuming you did as I did in this guide, i.e. you have a copy of wolvix with the added/modified modules in /remaster/, all you have to do is this:

```
cd /remaster/wolvix/
./make_iso.sh /somewhere/you/want/a/big/iso/wolvix-dhis.iso
```

and then burn this iso to a cd using your favorite OS/recorder.

But to make it all slightly more accesible, I did a bit more.

First I edited the desktop and the menu/panels slightly. This is really simple, just make the changes you want, and copy it back into the 002_core.lzm as described above. (Short recap, boot into your wolvix remaster, change the desktop however you want it, then unpack 002_core.lzm somewhere using dir2lzm, and copy /home/dhis/ into the same directory, and repack 002_core.lzm)

I also changed how wolvix starts up. Normally wolvix boots to a X login screen, with only one user available for logging in with, "root", with the password "toor". As already mentioned in the guide I added a user called "dhis", so if you followed the guide above you could log in as that person now and you'd have tomcat etc available. However, I wanted an automatic login, since this is a live CD. (If "security!!!" is the first thing you think when reading auto-login, good, but remember that this is a live CD. If you can reboot a computer from a CD, you can do whatever you want with it anyway, which is why I feel auto-login is acceptable.)

If you've tried Zenwalk you'll see it has automatic login to a normal user, which it uses gdm to accomplish. And gdm and kdm actually seem to be the only desktop managers who provide this, and both require some libraries which would make our wolvix-remaster bigger, which we don't really want. So I cheated. A lot. I added the "noslim" boot option to /boot/isolinux/isolinux.cfg, which stops wolvix from starting it's desktop manager, Slim:

```
APPEND changes=wolvixsave.xfs max_loop=255 initrd=/boot/initrd.gz
ramdisk_size=6666 root=/dev/ram0 rw vga=791 splash=silent
```

to

```
APPEND changes=wolvixsave.xfs max_loop=255 initrd=/boot/initrd.gz
ramdisk_size=6666 root=/dev/ram0 rw vga=791 splash=silent noslim
```

I then edited /etc/inittab to automatically login the user dhis on virtual console 1 (the following is loosely based on http://ubuntuforums.org/archive/index.php/t-31310.html):

```
1:2345:respawn:/sbin/getty 38400 Linux
```

to

```
1:2345:respawn:/sbin/getty -n -l /bin/mylogin 38400 Linux
```

where /bin/mylogin is a small C program I added to the 002_core.lzm module, which only does the following:

```
int main() {
execl( "login", "login", "-f", "dhis", NULL);
}
```

So whenever wolvix starts, it skips the desktop manager, but it automatically logs in "dhis" on virtual console 1.

(From my understanding, this should've been enough, but it still asked for a password, so on the current remaster, the password for "dhis" is actually empty to ensure it logs in automatically. It's on the TODO-list...)

Then I changed his .bash_profile (again, you have to add this to the 002_core.lzm module) to include the following:

```
if [ -z "$DISPLAY" ] && [ $(tty) == /dev/vc/1 ]; then
startx
fi
```

meaning that it automatically starts x when logging in on virtual console 1, unless X is already running ($DISPLAY set).

This last part is full of dirty hacks and tricks, so I really doubt this is how a more final version of this remaster should look. However, it works(tm) now, which is good enough for seeing if this is something that will be useful to anyone.

A.2 Changing the live CD to include a modified Hard Disk installer

(To understand this part you should likely have read the section on creating a live CD)

I've done some more changes to Wolvix to make it suitable as an install-cd for a costumized Linux for use in Kerala. The changed iso is not available on the net anywhere yet.

The main changes were some changes to the desktop, and modifications of the install-script that comes with Wolvix so it'll ask less questions (in other words, make some assumptions since we know a bit about the hardware here) and be easier to use, as well as changing mysql and tomcat to recognize the way we install these there, and use those data if available.

Next I added a trivial change to the core-package, I included a file called core/home/dhis/Desktop/hdinstall.desktop, which contains:

```
[Desktop Entry]
Name=Install to Hard Disk
Comment=Installs Linux and DHIS2 to the Hard Disk
Exec=gksu python /usr/sbin/wolvixinstall.py
Terminal=false
Type=Application
StartupNotify=true
Encoding=UTF-8
```

Basically it's a desktop link directly to the hard-disk installer.

Then I modified this installer a bit. The package it's located in is base/200-wolvix-control-panel.lzm. Unpackaging this to a folder wcp I then modified wcp/usr/sbin/wolvixinstall.py and wcp/usr/sbin/mkbootloader.py. These are python scripts, and fairly easy to read as well if you have some python knowledge. The main things I did to the installer was to remove the warning at the start, fix up the UI a bit and change some defaults. It now defaults to installing a bootloader, and installing to command line (since just booting auto-logs in the user "dhis" and starts a graphical desktop. This we might want to change for an actual install, auto-booting into a user is convenient but not very secure... Also, I fixed the script so it correctly populates the swap and root partition dropdown menus and selects sensible defaults. At the end I also added a line to remove the hdinstall-link on the desktop from the finished install. I also changed the bootloader-script to set proper names on the boot options by default, guessing by filesystems on the partitions. Finally I repackaged this.

One last thing I did that separates this from the earlier live CD, is the need to configure how mysql and tomcat start up, so they will use the same data as the windows installation. For this purpose, I made a very small module including just one file, *rc.dhis2*. This is a shell-script that will try to find a windows-install of DHIS2:

```
#!/bin/sh

WIN_DIR=""

for type in "hd" "sd"; do
  for drive in "a" "b" "c" "d"; do
    for part in "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"; do
      #echo "Trying /mnt/$type$drive$part/dhis2/"
      if [ -d /mnt/$type$drive$part/dhis2 ]; then
        WIN_DIR=/mnt/$type$drive$part/dhis2
        #echo "Found windows install at $WIN_DIR"
      fi
    done
  done
done
```

Now we need to change rc.tomcat and rc.mysql to include the data found by this script. Mysql changes (added these lines to rc.mysqld):

```
WIN_DIR=""
# see if there's a windows install on the machine
if [ -r /etc/rc.d/rc.dhis2 ]; then
  . /etc/rc.d/rc.dhis2
fi

if [ ! -z $WIN_DIR ]; then
  if [ -d $WIN_DIR/mysql/data ]; then
    DATADIR=$WIN_DIR/mysql/data
  fi
fi
```

Tomcat changes (added these lines to rc.tomcat):

```
        # See if we can find the war-file on a windows partition
        echo "Looking for dhis.war file..."
        rm -rf /opt/tomcat/webapps/*
        WAR_FILE=/opt/tomcat/dhis.war
        if [ ! -z $WIN_DIR ]; then
          if [ -r $WIN_DIR/tomcat/webapps/dhis.war ]; then
```

```
    WAR_FILE=$WIN_DIR/tomcat/webapps/dhis.war
  fi
fi
ln -s $WAR_FILE /opt/tomcat/webapps/dhis.war
echo "Using $WAR_FILE"

# see if we can find the ra-folder on a windows-partition
rm -rf /root/dhis/*
RA_FOLDER=/opt/tomcat/dhis/ra/
if [ ! -z $WIN_DIR ]; then
  if [ -d $WIN_DIR/dhis/ra/ ]; then
    RA_FOLDER=$WIN_DIR/dhis/ra/
  fi
fi
ln -s $RA_FOLDER /root/dhis/ra
```

These changes have to be repackaged back into mysql.lzm and tomcat.lzm respectively of course.

This page is about the installer we've made for easily installing all the software necessary to run DHIS2 on Windows XP. An alternative version with a gui installer can be found at DHIS2 Windows Installer

Please help out defining the Installation Dialog

(This should be a list of links, like an index. Someone teach me how to do that...)

1. Getting the installer

2. Installation procedure

3. How the installer works

4. How to quickly make a new install/upgrade CD

5. Creating an install CD including the Linux-installer 6. Comments about choices

Appendix B Kerala Installers Documentation

B.1 Easy installation of DHIS2 on Windows XP

This page is about the installer we've made for easily installing all the software necessary to run DHIS2 on Windows XP. An alternative version with a gui installer can be found at DHIS2 Windows Installer

1. Getting the installer
2. Installation procedure
3. How the installer works
4. How to quickly make a new install/upgrade CD
5. Creating an install CD including the Linux-installer 6. Comments about choices

B.2 Getting the installer

We've tried to make the installer as easy as possible, so hopefully even someone with very little computer knowledge can complete it. Here's the installation guide we've made for use here in Kerala: http://folk.uio.no/oysts/dhis/dhis2-installer-guide.pdf

That guide assumes you have a CD with the installer on it already though, so if you don't:

- download the installer from http://folk.uio.no/oysts/dhis/dhis2.zip
- alternatively, if you want a working CD, download http://folk.uio.no/oysts/dhis/dhis-wolvix.iso, this contains the Linux-installer as well though, so it's a lot bigger
- unzip it

Now you will have a directory called "dhis2_installer" containing three files:

```
autorun.inf
dhis2.zip
installer.bat
unzip.exe
```

Burn these four files to a CD or copy them to an USB pen drive or something similar. If you're installing DHIS2 on the same machine you just downloaded to, you can of course simply run the install from the folder you just made.

B.3 How the installer works

When you're ready to install, insert your CD and it will automatically run *installer.bat* if your computer is set up to autoplay CD's. This is accomplished by the file *autorun.inf*, which contains simply:

```
[autorun]
open=installer.bat
```

This tells windows to automatically run the *installer.bat* script when the CD is inserted. If windows it not set up to autoplay CD's, you will have to run this script manually.

It will by default be installed to *c:\dhis2*, and this folder will contain everything needed, except for a couple of shortcuts that will be installed elsewhere. If you want to change where it is installed to (please don't if you're also planning to use the Wolvix Linux DHIS installation), you have to edit the top of the installer.bat script, and change the variables drive_letter and or dhis2_dir. For example, installing to *d:\foo* instead of c:\dhis2 you'd change

```
set drive_letter=c
set dhis2_dir=%drive_letter%:\dhis2
```

to

```
set drive_letter=d
set dhis2_dir=%drive_letter%:\foo
```

(The rest of this page assumes the location c:\dhis2)

Contents of the installer as of this writing:

- *c:\dhis2\dhis* - contains some shortcuts and the kerala reports in a subfolder ra
- *c:\dhis2\firefox* - the Mozilla Firefox web browser...
- *c:\dhis2\java* - jdk 1.5.0_12 (jre should likely work but...)
- *c:\dhis2\mysql* - mysql 5.0.45
- *c:\dhis2\tomcat* - apache tomcat 6.0.14
- *c:\dhis2\tools* - various tools for the installation, mainly bat-scripts

All we've done with these packages is adding a hibernate.properties file to the tomcat installation (*tomcat\lib\hibernate.properties*) as well as an updated war-file including the reports made here in Kerala, and importing the kerala database into mysql. Since none of this needs any kind of installation but can be run from this location, the installer is simply a matter of copying the entire DHIS2 folder to the new computer. To accomplish this we simply zipped down the DHIS2-folder and wrote a small batch script that "installs" DHIS2 by unzipping this file and setting some variables. In more detail it does the following:

- stops and removes old mysql and tomcat services (this is to ensure the services running are running our versions in case of multiple installation. The installer is not nice to previous installations, you've been warned)
- unzips the installation to *c:\dhis2*
- sets JAVA_HOME to *c:\dhis2\java* and USER_HOME to *c:\dhis2*
- saves the installation-path to disk (in case you change it, there's an option for that at the top of the installer-script)
- deletes any existing DHIS2 user, and then adds it again with no password, and sets a password for the user you are installing as
- copies the shortcuts to DHIS2 (http://127.0.0.1:8080/dhis/) to the allusers folder, meaning that every user logging on to the machine gets the shortcut on their desktop and their start menu
- creates a service for mysql and tomcat so they start when the computer starts up
- starts the services

The tools-folder contains quite a few scripts you might or might not find useful:

- *create_installer.bat*: This script takes the contents of the DHIS2-folder and creates a new installer (i.e. 4 files) in *c:\dhis2\installer*. If you want to change the contents of the installer, say upgrading tomcat, just do it, make sure it works, run this script, and you have something you can install on a new machine again.
- *create_upgrader.bat*: This script creates a folder *c:\dhis2\upgrader* which contains 4 files. Burning these to a CD will result in a CD that will automatically run, and upgrade the DHIS2-install on the machine. It will update the war-file and the ra-folder, as well as the database (without removing any old data).
- *install_services.bat*: Just installs mysql and tomcat as services. Not terribly useful, same lines of code as the installer-script really.
- *installer.bat*: Described in detail above...
- *open_firefox.bat*: Starts Firefox with dhis up.
- *remove_services.bat*: Removes the mysql and tomcat services (stopping them first)
- *restart_services.bat*: Stops and starts the mysql and tomcat services
- *start_services.bat*: Starts the tomcat and mysql services

- *stop_services.bat*: Stops the mysql and tomcat services (notice a pattern?)
- *uninstaller.bat*: Uninstalls the DHIS2 folder (i.e. deletes it), and removes the desktop shortcuts etc. This should be improved upon a bit, since it won't actually be able to remove the *c:\dhis2\tools* folder since it is running from there. But it gets rid of most stuff, including removing services.
- update_database.bat: This script is to not have to know anything about mysql to update the database. If you have a mysql export, simply place it the tools\new_database folder as "db.sql" and run this script. It will import it for you. (Yeah, this is not terribly useful, but still...)
- *upgrader.bat*: This script will upgrade an existing dhis2 install (war-file, database, reports), like this:
    - Stop tomcat
    - Remove the old ra-folder (*c:\dhis2\dhis\ra*) and the old unpacked war-file (*c:\dhis2\tomcat\webapps\dhis\*)
    - Unzip the zip-file (replacing the war-file, adding a new ra-folder, and extracting the exported database)
    - Update the database and remove the exported file
    - Start tomcat
- *variables.bat*: This simply sets the variables %drive_letter% and %dhis2_dir%. If you installed in another folder by editing the installer-script before running it, this is where the location will be stored. All the other scripts run this first so they should be independant of where the install is actually located.

B.4 Creating an install CD including the Linux-installer

This part assumes you've read how to create a windows-installer. It is also extremely simple.

You need a CD with the latest version of the Linux-installer. Now, make a temporary folder (e.g. *remaster*), and copy the contents of the CD there. Then the folder should contain two folders and 4 files, like this:

```
boot/
wolvix/
autorun.inf
dhis2.zip
installer.bat
unzip.exe
```

Replace the 4 files with the 4 from c:\dhis2\installer that you made by running *c:\dhis2\tools\create_installer.bat*.

Finally, run *remaster\wolvix\make_iso.bat*. This will create an iso-image *c:\dhis-wolvix.iso*, so make sure that *c:\* has around 700MB of free space. This iso-image you can make a CD from using any cd-recording program.

If you want to create the iso-image somewhere else, you can simply run

```
make_iso.bat c:\somewhere\else\name_of_iso.iso
```

instead.

B.5 Comments about choices

We chose to go with Tomcat instead of Jetty, based on some wierd behaviour of Jetty on different machines here. Running the database server and the webapp server as services is by far the cleanest way of running them at startup (we want the machines in the clinics ready for running DHIS2 right away, this is mainly what they'll be doing), and MySQL, Jetty and Tomcat all come with solutions for setting them up as services. However, on some of our slower test-machines (but also from very rarely on our faster ones) Jetty would sometimes fail to start at startup of windows. It was always the same problem, it timed out waiting for a respons from the JVM. This turned out to be pretty damn hard to debug, since it seemed completely random whether or not Jetty would start. So since Tomcat worked flawlessly, and we couldn't find any significant difference in the two, we ended up just going with Tomcat.

As for database-server, it wasn't much of a choice. MySQL was what was in use and known here in Kerala, so that's what we went with. And it hasn't given us any trouble whatsoever.

Regarding Java, we had to go with 1.5 of course, but we also chose to include the entire JDK instead of just the JRE, "just in case". The extra space requirement on the hard-disk is insignificant on the machines in question, and the effect on the install is also negligeble. There is still plenty of space on the install CD, even with a complete Linux-install and all necessary software for running DHIS2 on both OS's.

Appendix C New Kerala Installers/Upgraders

C.1 How to use the DHIS2 installer to make a new install-cd, or an upgrade-cd

You need to have an installation of the DHIS2-package, which is easiest to accomplish by simply running the installer. This will give you everything you need in *c:\dhis2*.

C.2 Preparation

Now you can modify this installation how you want it.

- If you want a new war-file, replace *c:\dhis2\tomcat\webapps\dhis.war*, and remove the folder *c:\dhis2\tomcat\webapps\dhis*. (Note, this will require you to stop the tomcat service first).
- Changing the xml-files for the reports is as easy as replacing the folder *c:\dhis2\dhis\ra* with your new ra-folder.
- Updating the database depends on which database you have been changing:
    - If you have worked directly on the database located in *c:\dhis2\mysql* then you don't need to do anything (skip to Creation).
    - If you have been working on a different mysql-install, you will need to export it, and then import it into the database located in *c:\dhis2\mysql*:

If you want to completely replace the old database, i.e. there is nothing in *c:\dhis2\mysql* you need to save, you can simply do a

```
mysqldump dhis2 > db.sql
```

from the database you have modified.

If you want to keep the data in the old database, you need to run

```
mysqldump --no-defaults --no-create-info --insert-ignore --add-drop-
table=false -u root dhis2 > db.sql
```

Either way you will have a file called db.sql. Copy this file to *c:\dhis2\tools\new_database\*, and then run the script *c:\dhis2\tools\update_database.bat*. (Note, the mysql-install located in *c:\dhis2\mysql\* needs to be running as a service at this point) After this is done you can remove *c:\dhis2\tools\new_database\db.sql*.

This part could also be done with the import-export-tool in DHIS2.

An important thing to remember is which mysql-database is currently running on the computer if you are using two different installs.

C.3 Creation

Now you are ready to make your own installer or upgrader.

To create a new installation-package, run the script *c:\dhis2\tools\create_installer.bat*. This will create a folder *c:\dhis2\installer\*, which contains four files. Burning these to a CD will result in a CD you can put into a new computer and it will start the installation of DHIS2.

To create an upgrade-package, run the script *c:\dhis2\tools\create_upgrader.bat*. This will result in a folder *c:\dhis2\upgrader\*, which contains four files. Burning these to a CD will result in a CD you can put into a new computer and it will start an upgrade of DHIS2, meaning that it will copy in the changes you did to the machine, and update the database without losing existing data.

Appendix D BitRock Installers Documentation

D.1 BitRock installers howto

(throughout this document, ${installdir} is wherever the user choses to install to, and is substituted by the bitrock installer to the correct value)

D.3 Quick example - changing the war-file:

- Install DHIS2 using an old installer (don't start DHIS2 from that installation)
- Replace ${installdir}\tomcat\webapps\dhis.war with the new one (must be called the same)
- Copy either http://www.hisp.info/~oysts/dhis-mysql.xml or http://www.hisp.info/~oysts/dhis-pgsql.xml to ${installdir}, depending on whether you are changing a postgres or a mysql package
- Start bitrock and open the xml-file
- Press build
- The new install-file is located in the bitrock output folder, which is unfortunately platform dependant
    - On Windows Vista it's: c:\Users\username\Installbuilder\output
    - Under Linux it's <bitrock-install-folder/output/

D.3 BitRock

We ended up using BitRock (http://bitrock.com/products_installbuilder_overview.html) as it was the most easy to use we found in the short time we had. Like confluence, it is not open source itself, but is free to use for open source projects. I've been in contact with them, everything is in order. Simply download the Professional version, and register it with the license file you can download here: http://folk.uio.no/oysts/dhis/license.xml

BitRock itself is a GUI-program for windows that is fairly easy to use. I've simply created an installer that creates 4 items in a DHIS2-folder on the start-menu: Readme, Start DHIS2, Stop DHIS2 and Uninstall DHIS2 - which all do as the names suggests. The installer also copies in four (five) folders in a user-specified location (defaults to c:\Program Files\<installer-name-and-version>\, e.g. c:\Program Files\dhis2pgsql-M9\). These four are *config*, *jdk* (optional) *mysql/pgsql*, *tomcat* and *tools*.

Of these four (five), *config* contains the README file and the hibernate.properties file. *tomcat* and *mysql/pgsql* contain the respective program installations, slightly modified as mentioned below. *jdk* (optional) contains a Java JDK (1.6.0_1 at the time of writing). *tools* contains four scripts and an exe file, *setx.exe*. That exe-file is used by the installer to set some environment variables. The scripts are:

- brinstall.bat - this is run by the install-program after all files are copied in. It sets up mysql/pgsql and tomcat services
- brstart.bat - this is the script run by the "Start DHIS2" icon on the start menu. It starts the tomcat and mysql/pgsql services, and opens a webpage to DHIS2
- brstart.bat - this is the script run by the "Stop DHIS2" icon on the start menu. It stops the tomcat and mysql/pgsql services.
- brstop.bat - This is the script run by the uninstall program before it removes the files. It stops and removes the tomcat and mysql services and deletes some files the uninstaller doesn't catch.

The project file I used for the various installers can be found as mentioned above (quick example section).

Simply place this in a folder (*c:\remaster* for instance) which also contains the four folders mentioned above, and you can modify the installer to your hearts content. If you only want to change the war-file and make a new installer, you only need to replace *c:\remaster\tomcat\webapps\dhis.war* with the new one, start BitRock and open the project file in *c:\remaster\dhis2.xml* and press "Build". It will then create a new installer, which will be placed in *My Documents\Installbuilder\output\*

D.4 Design

We chose to implement the installation by having a completely selfcontained install, only requiring a modern web browser such as Internet Explorer 7+, Opera or Firefox, which we believe is reasonable. You'll also need your own install of java if you chose not to download the version that includes java. MySQL/PostgreSQL and Tomcat will be run as services started on demand. These services we installed as dhis2mysql/dhis2pgsql and dhis2tomcat to ensure that they do not interfere with any earlier installations. This also required us to change which ports these were running on, and they now run on 3307 and 8180 for mysql and tomcat respectively. (For the time being, I haven't done the same for postgres, since there's issues with combining openhealth and DHIS2 with a non-standard port for postgres) Details follow:

Changing the tomcat port number is done by changing the port-numbers in tomcat\conf\server.xml

Since mysql would also be running on a different port, we needed to change how tomcat connected. This is done by changing ${installdir}\config\hibernate.properties:

```
hibernate.connection.url = jdbc:mysql://localhost:3307/dhis2?
```

so that it now uses our mysql server, running at port 3307.

Changing mysql to run on a different port was fairly simple as well. The only tricky part to remember was to add a [dhis2mysql] section in my.ini, so that when it was installed as a service called dhis2mysql, it would read the correct parameters.

Under this we changed the port number:

```
port=3307
```

Mysql also had hardcoded lines for the data-directories in *${installdir}\mysql\my.ini*, this was fixed by changing it to the following:

```
basedir=@BASEDIR@
datadir=@DATADIR@
```

and then having the installer do a text-substitution on *${installdir}\mysql\my.ini* after the install is completed, changing *@BASEDIR@* to *${installdir}\mysql* and *@DATADIR@* to *${installdir}\mysql\data*.

Last, but not least, the installer will set a couple of environment variables:

- DHIS_PROPERTIES_FILE - lets DHIS2 know which hibernate.properties file to read when starting up.
- DHIS2DIR - location of the install, used for upgrades (we prefer not messing with the registry...)
- JAVA_HOME - set to the location of the provided java, ${installdir}\jdk (assuming you downloaded an install with java...)

D.5 Some notes on wierd things I had to do to make everything work:

It works with both java 1.5 and 1.6, but tomcat has some wierd issues with 1.6. I had to copy msvcr71.dll from jdk\bin\ to tomcat\bin\ to get it to work (see http://waspflash.blogspot.com/2007/04/setting-up-tomcat-6.html for instance).

Appendix E Sample BitRock XML

Sample XML-file from the BitRock installers.

```
<project>
   <shortName>dhis2pgsql</shortName>
   <fullName>DHIS2</fullName>
   <version>2.0-beta</version>
   <readmeFile>config/Readme.txt</readmeFile>
   <leftImage>config/dhis2-logo-rotated.png</leftImage>
   <logoImage>config/dhis2-logo.png</logoImage>
   <splashImage>config/dhis2-logo.png</splashImage>
   <postInstallationScript>${installdir}/tools/brinstall.bat</postInstallationScript>
   <postInstallationScriptArguments>${installdir}</postInstallationScriptArguments>
   <showPostInstallationScriptResult>1</showPostInstallationScriptResult>
   <preUninstallationScript>${installdir}/tools/bruninstall.bat</preUninstallationScript>
   <preUninstallationScriptArguments>"${installdir}"</preUninstallationScriptArguments>
   <postUninstallationActionList>
      <removeDirectoryFromPath>
         <path>${installdir}\jdk\jre\bin</path>
      </removeDirectoryFromPath>
      <deleteEnvironmentVariable>
         <name>DHIS2DIR</name>
         <scope>system</scope>
      </deleteEnvironmentVariable>
      <deleteEnvironmentVariable>
         <name>DHIS_PROPERTIES_FILE</name>
         <scope>system</scope>
```

```
    </deleteEnvironmentVariable>
    <deleteEnvironmentVariable>
        <name>JAVA_HOME</name>
        <scope>system</scope>
    </deleteEnvironmentVariable>
</postUninstallationActionList>
<enableRollback>1</enableRollback>
<saveRelativePaths>1</saveRelativePaths>
<windowsExecutableIcon>config/dhis2.ico</windowsExecutableIcon>
<componentList>
    <component>
        <name>default</name>
        <description>Default Component</description>
        <canBeEdited>1</canBeEdited>
        <selected>1</selected>
        <show>1</show>
        <folderList>
            <folder>
                <description>Program Files</description>
                <destination>${installdir}</destination>
                <name>programfiles</name>
                <platforms>all</platforms>
                <actionList>
                    <addEnvironmentVariable>
                        <name>DHIS2DIR</name>
                        <scope>system</scope>
                        <value>${installdir}</value>
                    </addEnvironmentVariable>
                    <addEnvironmentVariable>
                        <name>DHIS_PROPERTIES_FILE</name>
                        <scope>system</scope>
                        <value>${installdir}\config\hibernate.properties</value>
                    </addEnvironmentVariable>
                    <addEnvironmentVariable>
                        <name>JAVA_HOME</name>
                        <scope>system</scope>
                        <value>${installdir}\jdk</value>
                    </addEnvironmentVariable>
                    <addDirectoryToPath>
                        <insertAt>end</insertAt>
                        <path>${installdir}\jdk\jre\bin</path>
                    </addDirectoryToPath>
                </actionList>
                <distributionFileList>
                    <distributionDirectory>
                        <origin>config</origin>
                    </distributionDirectory>
                    <distributionDirectory>
                        <origin>pgsql</origin>
                    </distributionDirectory>
                    <distributionDirectory>
                        <origin>tomcat</origin>
                    </distributionDirectory>
                    <distributionDirectory>
```

```
                <origin>tools</origin>
            </distributionDirectory>
            <distributionDirectory>
                <origin>jdk</origin>
            </distributionDirectory>
        </distributionFileList>
    </folder>
</folderList>
<startMenuShortcutList>
    <startMenuShortcut>
        <comment>Uninstall ${product_fullname}</comment>
        <name>Uninstall ${product_fullname}</name>
        <runInTerminal>0</runInTerminal>
        <windowsExec>${installdir}/${uninstallerName}.exe</windowsExec>
        <windowsExecArgs></windowsExecArgs>
        <windowsIcon></windowsIcon>
        <windowsPath>${installdir}/</windowsPath>
    </startMenuShortcut>
    <startMenuShortcut>
        <comment>Starts postgresql, tomcat and then DHIS2</comment>
        <name>Start DHIS2</name>
        <runInTerminal>0</runInTerminal>
        <windowsExec>${installdir}/tools/brstart.bat</windowsExec>
        <windowsExecArgs></windowsExecArgs>
        <windowsIcon>${installdir}/config/dhis2.ico</windowsIcon>
        <windowsPath></windowsPath>
    </startMenuShortcut>
```

```
            <startMenuShortcut>
               <comment>Stops the services</comment>
               <name>Stop DHIS2</name>
               <runInTerminal>0</runInTerminal>
               <windowsExec>${installdir}/tools/brstop.bat</windowsExec>
               <windowsExecArgs></windowsExecArgs>
               <windowsIcon>${installdir}/config/dhis2.ico</windowsIcon>
               <windowsPath></windowsPath>
            </startMenuShortcut>
            <startMenuFileShortcut>
               <comment>Readme file</comment>
               <filePath>${installdir}/config/Readme.txt</filePath>
               <name>Readme</name>
               <runInTerminal>0</runInTerminal>
               <windowsIcon></windowsIcon>
            </startMenuFileShortcut>
            <startMenuShortcut>
               <comment>GUI for managing PostgreSQL databases</comment>
               <name>pgAdmin</name>
               <runInTerminal>0</runInTerminal>
               <windowsExec>${installdir}/pgsql/bin/pgAdmin3.exe</windowsExec>
               <windowsExecArgs></windowsExecArgs>
               <windowsIcon></windowsIcon>
               <windowsPath></windowsPath>
            </startMenuShortcut>
         </startMenuShortcutList>
      </component>
```

```
  </componentList>

  <finalPageActionList>

    <runProgram>

      <program>${installdir}/tools/brstart.bat</program>

      <programArguments></programArguments>

      <progressText>Do you want to launch ${product_fullname} now?</progressText>

    </runProgram>

  </finalPageActionList>

  <parameterList>

    <directoryParameter>

      <name>installdir</name>

      <description>Installer.Parameter.installdir.description</description>

      <explanation>Installer.Parameter.installdir.explanation</explanation>

      <value>${platform_install_prefix}/${product_shortname}-$
{product_version}</value>

      <default>${platform_install_prefix}/${product_shortname}-$
{product_version}</default>

      <allowEmptyValue>0</allowEmptyValue>

      <ask>yes</ask>

      <cliOptionName>prefix</cliOptionName>

      <mustBeWritable>yes</mustBeWritable>

      <mustExist>0</mustExist>

      <width>40</width>

    </directoryParameter>

  </parameterList>

</project>
```

Appendix F Linux Live CDs Experiences

This list can also be found at: http://www.hisp.info/dhis2/wiki/Linux

F.1 Dreamlinux

http://www.dreamlinux.com.br/english/index.html Debian/Morphix based.

Review

Can you spell OS X clone? Seriously though, the XFCE skin looks great. Being based on neither Knoppix or Slack/Linux-live this distro comes with it's own remastering tool, which is the most user-friendly I've seen. It's developed to be usable by anyone, and it seems to be...except that after removing some packages the new iso was larger than the old one, too large to fit on a cd... Also, it had problems with discovering my USB-stick, most likely because I had a SATA disk and linux detects USB, SATA and everything SCSI as SCSI-stuff (smart eh?).

F.2 Knoppix

http://www.knoppix.org/

http://www.knoppix.net/wiki/Knoppix_Remastering_Howto

Perhaps the best known live CD. Crammed full of software (there's even a DVD version with every OSS project known to man on it...), it needs heavy stripping to be useful. Might want to consider a smaller remaster, need to find one that's kept up to date though. Remastering process is fairly easy. Best I've seen at detecting hardware and doing the right thing with it.

F.3 Feather

http://featherlinux.berlios.de/

Remastering feather

Based on Knoppix, basically a heavily-stripped down version. Fairly decent desktop once you get it up, clean and simple. Requires the user to answer some questions regarding X and keymap etc. before starting though, while "regular" Knoppix simply boots up. I'm more in favor of the latter, with proper costumization (i.e. we make one liveCD for users in India, one for South Africa etc). Also bit lagging in versions in regards to Knoppix, might not matter much though.

F.4 SLAX

http://www.slax.org

Not much to say, like Knoppix it seems good at hardware detection and is decent at setting things up. Would need some fair editing/remastering to be optimal though. Seems as easy or easier than Knoppix to remaster, though it lacks the same amount of detailed instructions on how to reconfigure KDE etc. (but that shouldn't be too much of a hassle)

F.5 ...and Simpleslax

http://scattershot.wickedtribe.org/

It's simple alright. The least fancy liveCD of all of these, the modified Fluxbox it's running would need to be edited quite a bit to make it easier to use. Face it, unless you've used linux for a while, right-clicking empty space on the desktop to start applications is not intuitive. People generally expect a start-menu or icons, either on the desktop or the bottom bar. On the upside it only requires 192 MB of RAM to put everything in RAM, which makes loading times blazing fast. Remastered in the same way SLAX is, so just adding a costum XFCE and the modules needed to run DHIS might be a good bet to get a really slim DHIS cd.

F.6 Zenwalk

http://www.zenwalk.org/

Based on Slackware. More complicated to remaster than Knoppix-variants due to additional software expected to be in ZenWalk? modules. Out of the box desktop is pretty spot on in regards to how such a distro should be though. Clean and simple. (edit after checking out SLAX) it seems the SLAX packages are interchangable with the zenwalk ones (why the fsck do you wanna rename them then, stupid zenwalk), so that makes zenwalk a way more likely candidate. It basically fixes what "problems" standard SLAX has for our purposes.

Addtional notes: Remastering the packages themselves is fairly easy. The desktop config however is trickier, as it is located in "06_xapps.lzm/etc/skel". Likely easiest to modify a desktop as you want it, and then copy the resulting home directory to a remaster of this package.

F.7 Wolvix

http://wolvix.org/

Favorable review

Another Slackware/Linux Live clone. Looks ok, but couldn't really find anything going for this one to put it ahead of Zenwalk, considering they're basically the same thing and zenwalk would need less work when remastering...

F.8 GoblinX

http://www.goblinx.com.br/en/index_home.htm

Either my boot media is fubar, or the latest goblinx release is corrupt. Trying again later. Second try fubar as well, so the Goblinx Mini 2.0.0 release is corrupt (md5sum correct and no errors on the CD etc.) But screw it, it's just a SLAX remaster anyway, and from the webpages, either regular SLAX or Zenwalk are better choices.

F.9 NimbleX

http://www.nimblex.net/ Uses SLAX modules (.mo) Provides fast version of KDE

The Custom Nimblex tool is a nice way to create a costum linux live-CD before even downloading, but it lacks a few of the modules needed for DHIS-cd, so a full remaster would have to be done. Can't quite see the point of Nimblex over any other Slackware/Live linux variant then, again, Zenwalk or SLAX seem more suited.

F.10 Arch

http://www.archlinux.org/

This one is more complex to remaster than the other variants, for no appearent gain. In addition it's mostly optimized for i686, which means a lot of the older hardware we are targeting with this project wouldn't be able to run it, so I think we can rule this one out.

F.11 DSL

http://www.damnsmalllinux.org/

Due to it's design-choices not really suitable for our needs. It's bigger brother DSL-N (Damn Small Linux, Not!) is still in development, which means going for a more stable liveCD is likely better.

F.12 Puppy

Grafpup has security, mysql and JRE packages. General intro (old): http://www.desktoplinux.com/articles/AT6545156120.html Roll your own: http://puppylinux.net/puppy-unleashed.htm Java: http://www.puppylinux.org/wikka/GuestTooJavaInstaller Using SLAX modules

F.13 Vector Linux

http://www.vectorlinux.com/

Long, distinguished history of targeting low end hardware. Unfortunately, little to no info regarding live-CD remastering, or any info at all on the live-CD...

F.14 Linux Live

http://www.linux-live.org/

Basically turns any linux distro into a live-CD. Probably way too much hassle though, since hardware auto-detection is the hard part of making an all-round CD like this...

F.15 Sabayon

http://www.sabayonlinux.org/ Gentoo based Review August 2007

F.16 PCLinuxOS

http://www.pclinuxos.com/ Increasingly popular, RPM based and Mandriva inspired http://www.tuxmachines.org/node/15844

F.17 Kate OS

http://www.kateos.org/ "very high efficiency, security, reliability and low system requirements. Support for common multimedia included." Review

3 CDs puts this more in the "full" category. Also, download links don't work.

JRE, Mysql ++

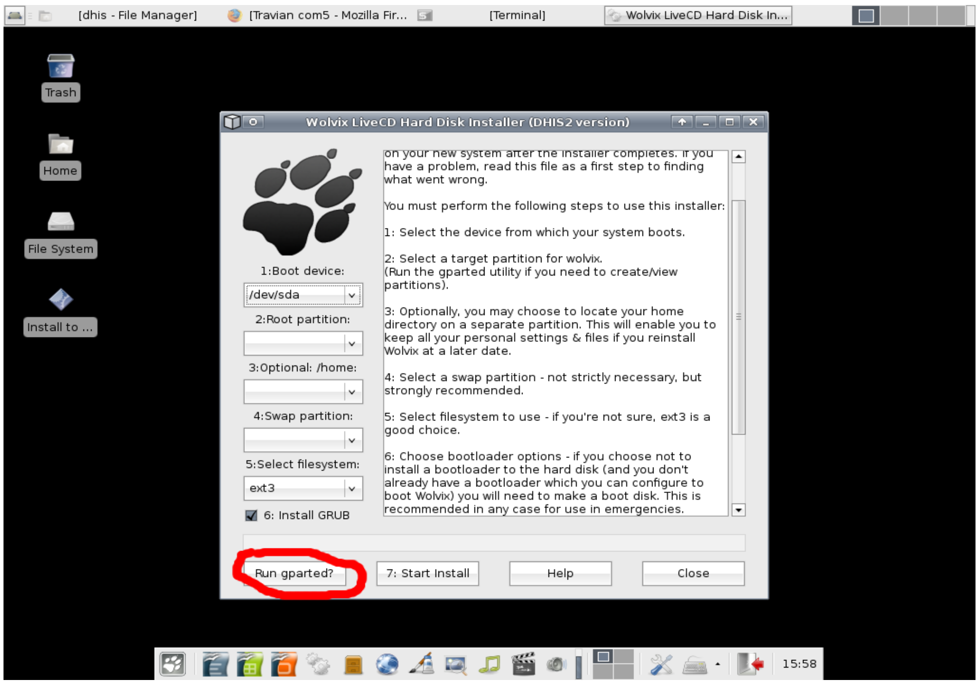
No documentation on how to remaster/change the liveCD, which makes it a hard sell. Was originally based on Slackware, but has left that behind. Seems to use a slightly modified live linux variant too, so can't really see why this is preferable to Zenwalk or SLAX. Their own comparison page more or less states the same as well.

Appendix G DHIS2 Installation Guide

This chapter is an excerpt of the guide we made for the system facilitators in India, and is only included to illustrate the difference in difficulty in the Windows vs. Linux installation procedure:

<snip>

3. Press the link labeled "Run gparted":



Wolvix LiveCD Hard Disk Installer (DHIS2 version)

on your new system after the installer completes. If you have a problem, read this file as a first step to finding what went wrong.

You must perform the following steps to use this installer:

1: Select the device from which your system boots.

2: Select a target partition for wolvix.
(Run the gparted utility if you need to create/view partitions).

3: Optionally, you may choose to locate your home directory on a separate partition. This will enable you to keep all your personal settings & files if you reinstall Wolvix at a later date.

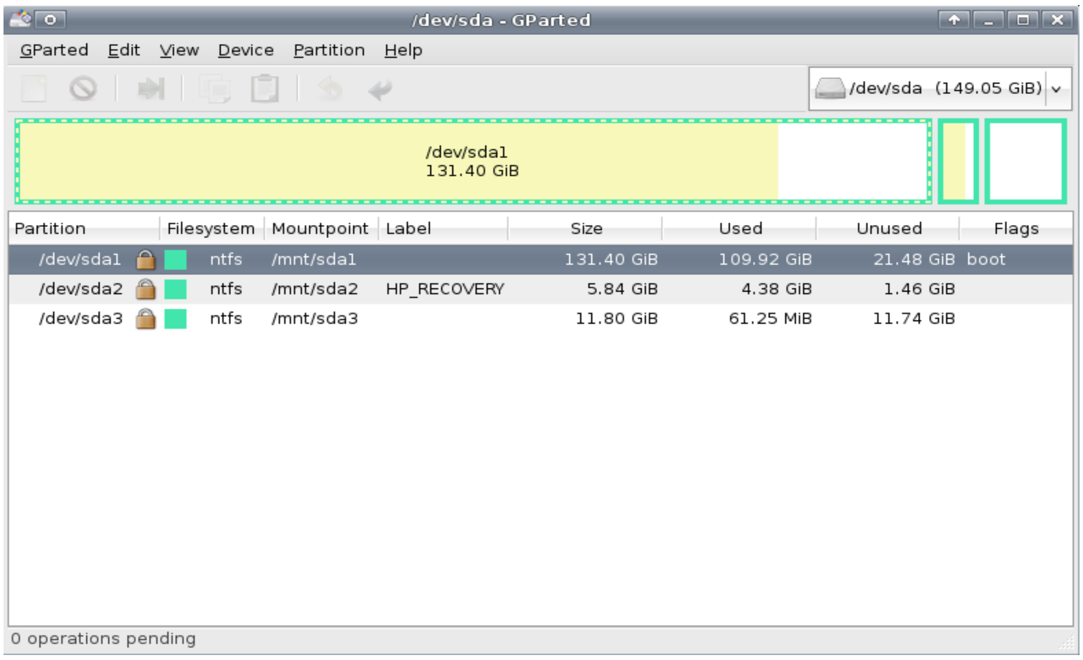
4: Select a swap partition - not strictly necessary, but strongly recommended.

5: Select filesystem to use - if you're not sure, ext3 is a good choice.

6: Choose bootloader options - if you choose not to install a bootloader to the hard disk (and you don't already have a bootloader which you can configure to boot Wolvix) you will need to make a boot disk. This is recommended in any case for use in emergencies.

1:Boot device: /dev/sda
2:Root partition:
3:Optional: /home:
4:Swap partition:
5:Select filesystem: ext3
✓ 6: Install GRUB

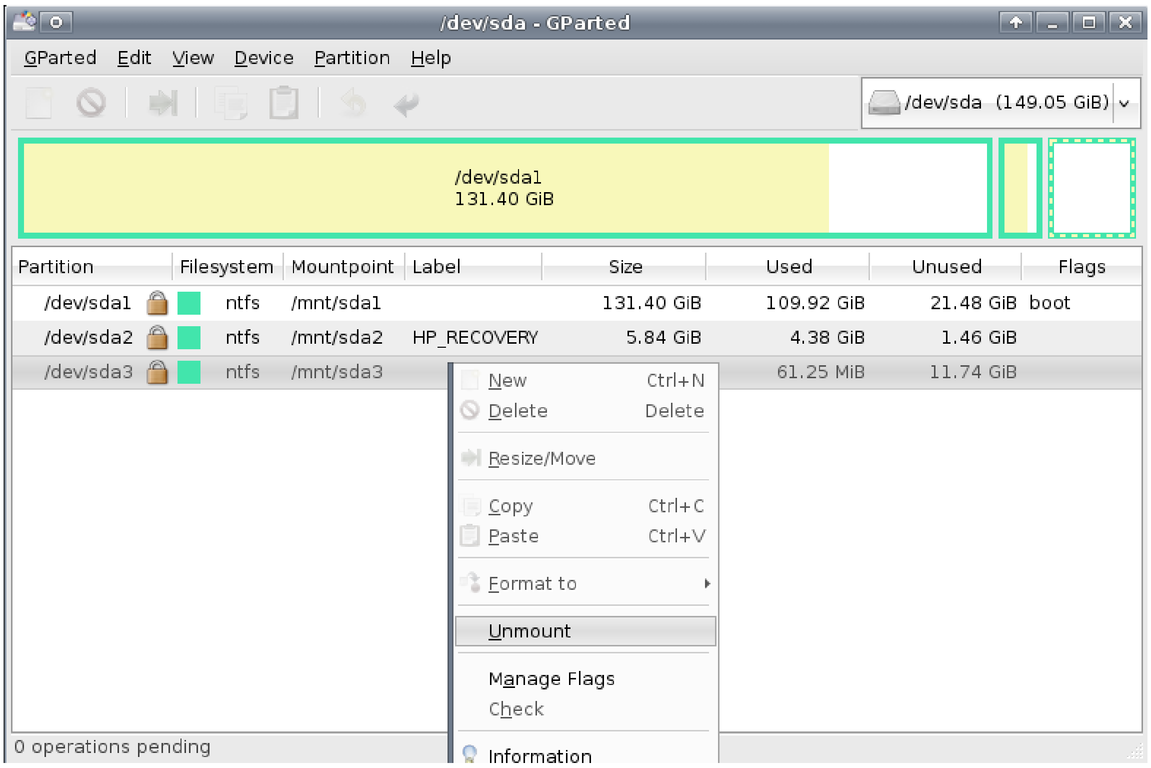
The partition manager ("gparted") will start, which should look like this (the numbers on your screen will be different)



Partition		Filesystem	Mountpoint	Label	Size	Used	Unused	Flags
/dev/sda1	🟩	ntfs	/mnt/sda1		131.40 GiB	109.92 GiB	21.48 GiB	boot
/dev/sda2	🟩	ntfs	/mnt/sda2	HP_RECOVERY	5.84 GiB	4.38 GiB	1.46 GiB	
/dev/sda3	🟩	ntfs	/mnt/sda3		11.80 GiB	61.25 MiB	11.74 GiB	

0 operations pending

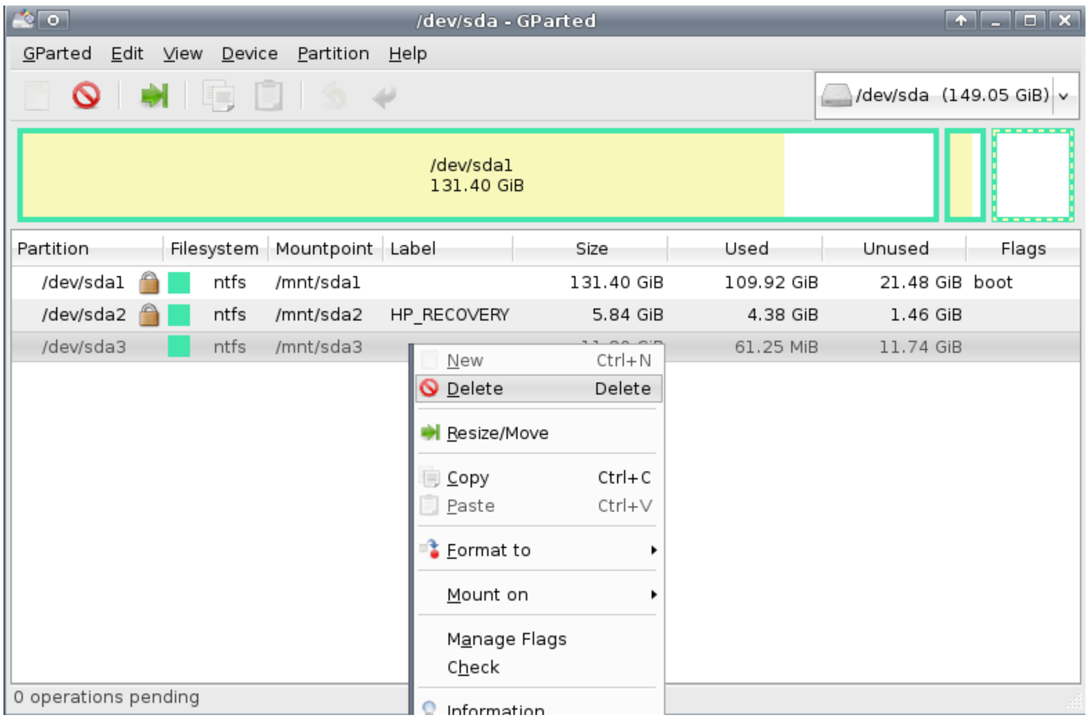
4. Now right-click the line that has “sda3” or “hda3” as part of it's name, and press “Unmount”:



Partition		Filesystem	Mountpoint	Label	Size	Used	Unused	Flags
/dev/sda1	🟩	ntfs	/mnt/sda1		131.40 GiB	109.92 GiB	21.48 GiB	boot
/dev/sda2	🟩	ntfs	/mnt/sda2	HP_RECOVERY	5.84 GiB	4.38 GiB	1.46 GiB	
/dev/sda3	🟩	ntfs	/mnt/sda3			61.25 MiB	11.74 GiB	

0 operations pending

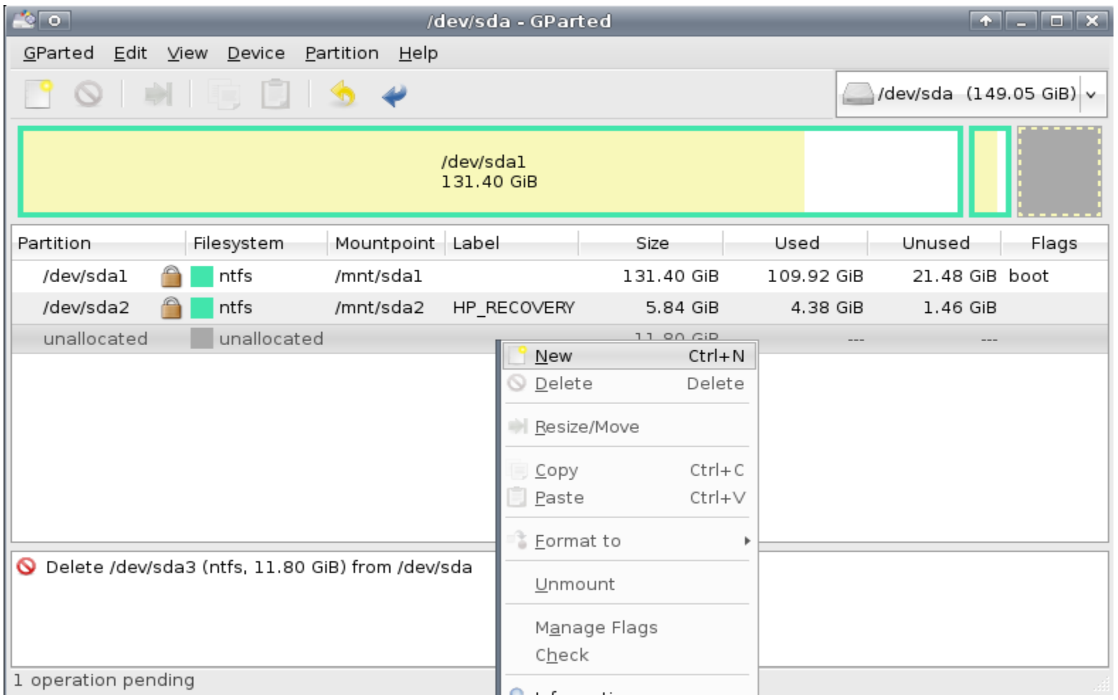
5. The image will now be “greyed out” for some seconds and then it will return, but now the “lock” on the line you selected in the last step will be gone. Right click the same line again, and press “Delete”:



Partition		Filesystem	Mountpoint	Label	Size	Used	Unused	Flags
/dev/sda1	🟩	ntfs	/mnt/sda1		131.40 GiB	109.92 GiB	21.48 GiB	boot
/dev/sda2	🟩	ntfs	/mnt/sda2	HP_RECOVERY	5.84 GiB	4.38 GiB	1.46 GiB	
/dev/sda3	🟩	ntfs	/mnt/sda3			61.25 MiB	11.74 GiB	

0 operations pending

4. Right click the line saying “unallocated” and press "New":

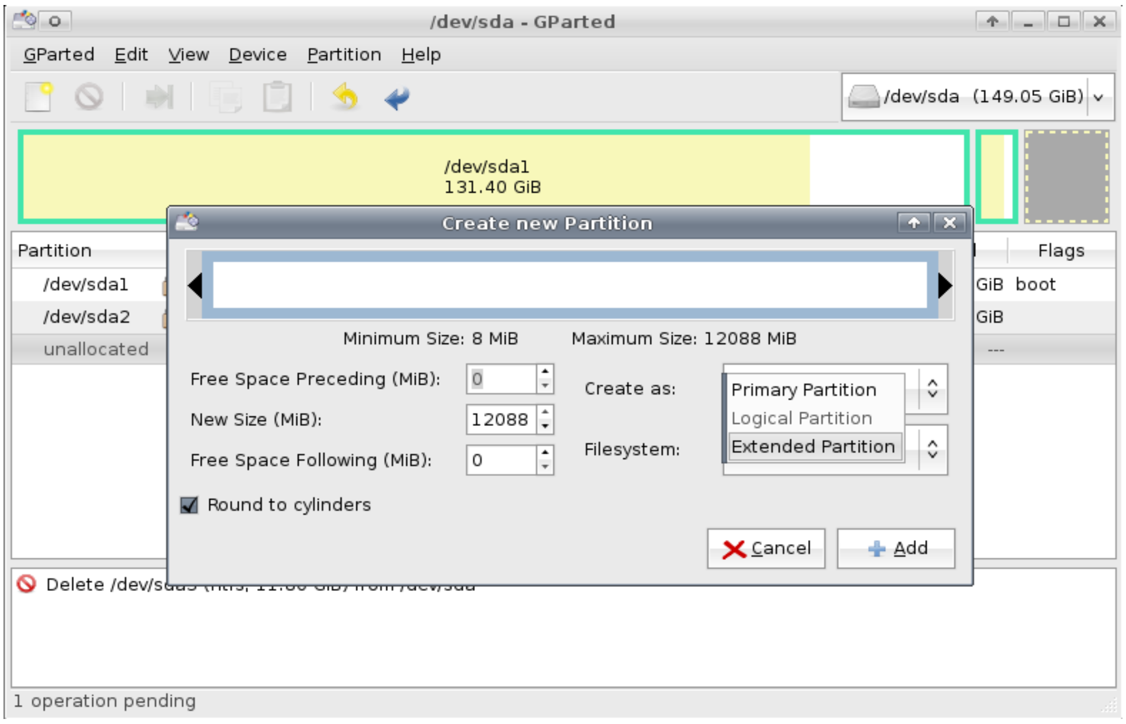


Partition	Filesystem	Mountpoint	Label	Size	Used	Unused	Flags
/dev/sda1	ntfs	/mnt/sda1		131.40 GiB	109.92 GiB	21.48 GiB	boot
/dev/sda2	ntfs	/mnt/sda2	HP_RECOVERY	5.84 GiB	4.38 GiB	1.46 GiB	
unallocated	unallocated			11.80 GiB	---	---	

Delete /dev/sda3 (ntfs, 11.80 GiB) from /dev/sda

1 operation pending

You will get a popup window, which should look something like this (the numbers will be different):



Minimum Size: 8 MiB Maximum Size: 12088 MiB

Free Space Preceding (MiB): 0

New Size (MiB): 12088

Free Space Following (MiB): 0

Create as:

Filesystem:

☑ Round to cylinders

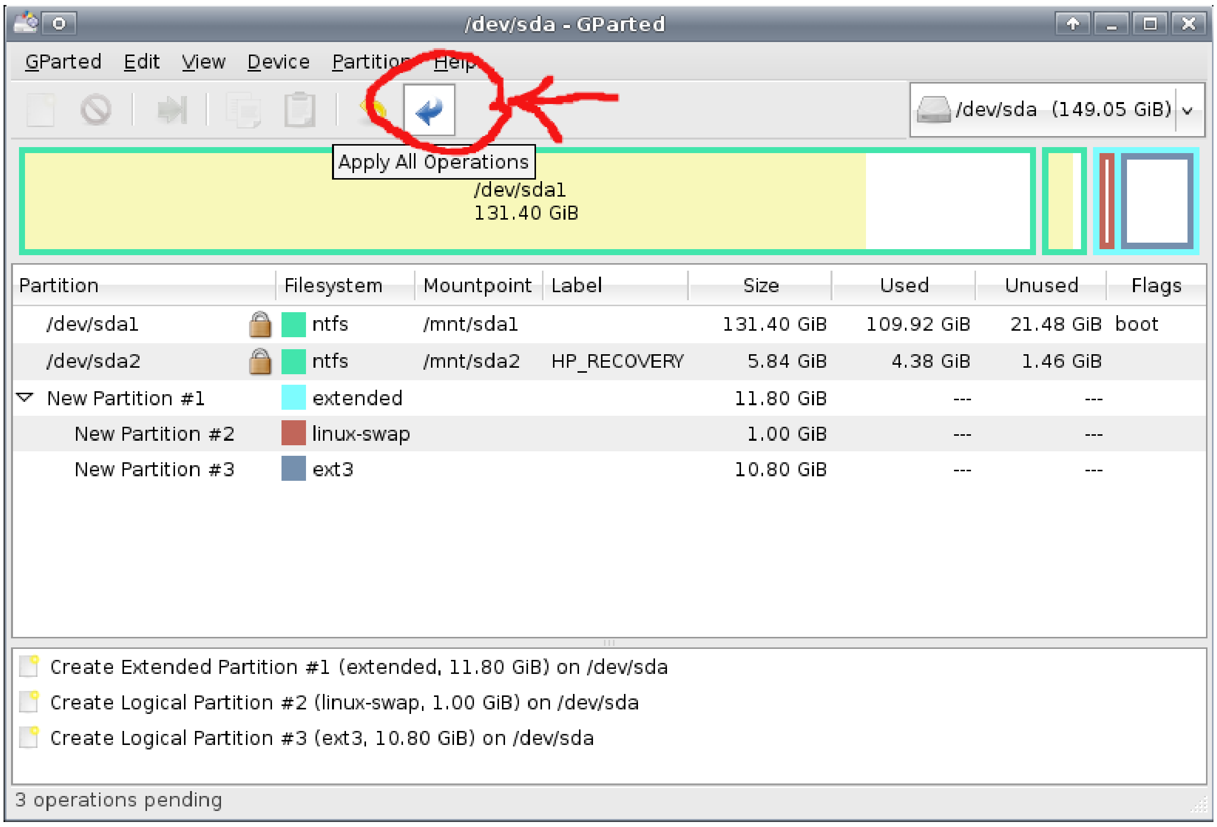
Delete /dev/sda3 (ntfs, 11.80 GiB) from /dev/sda

1 operation pending

<snip>

12. Now the screen should look like this, press the Apply button (the button with the blue arrow,

marked here with a red circle and red arrow):



Partition		Filesystem	Mountpoint	Label	Size	Used	Unused	Flags
/dev/sda1	🔒	🟩 ntfs	/mnt/sda1		131.40 GiB	109.92 GiB	21.48 GiB	boot
/dev/sda2	🔒	🟩 ntfs	/mnt/sda2	HP_RECOVERY	5.84 GiB	4.38 GiB	1.46 GiB	
New Partition #1		🟦 extended			11.80 GiB	---	---	
New Partition #2		🟥 linux-swap			1.00 GiB	---	---	
New Partition #3		🟦 ext3			10.80 GiB	---	---	

```
Create Extended Partition #1 (extended, 11.80 GiB) on /dev/sda
Create Logical Partition #2 (linux-swap, 1.00 GiB) on /dev/sda
Create Logical Partition #3 (ext3, 10.80 GiB) on /dev/sda
3 operations pending
```

Appendix H Scripts for Linux BitRock installers

Post install script for a MySQL based Ubuntu BitRock installer:

```
#!/bin/sh

if [ `id -u` != 0 ]; then
    ln -sf "$1/tools/linuxstart-user.sh" "$1/tools/linuxstart.sh"
    ln -sf "$1/tools/linuxstop-user.sh" "$1/tools/linuxstop.sh"
else
    # we're root ^^
    deluser dhis2mysql
    deluser dhis2tomcat
    adduser --system --no-create-home --group dhis2mysql
    adduser --system --no-create-home --group dhis2tomcat
    chown -R dhis2mysql.dhis2mysql "$1/mysql/data"
    chown -R dhis2tomcat.dhis2tomcat "$1/tomcat/"
    cp $1/dhis/dhis2.desktop /usr/share/applications/
    cat > /etc/init.d/dhis2mysql << EOF
#! /bin/bash
export dhis2_dir="$1"
EOF
    cat "$1/tools/dhis2" >> /etc/init.d/dhis2mysql
    chmod 755 /etc/init.d/dhis2
    ln -sf /etc/init.d/dhis2 /etc/rc2.d/S50dhis2
    ln -sf /etc/init.d/dhis2 /etc/rc0.d/K50dhis2
    ln -sf /etc/init.d/dhis2 /etc/rc6.d/K50dhis2
    ln -sf "$1/tools/linuxstart-root.sh" "$1/tools/linuxstart.sh"
    ln -sf "$1/tools/linuxstop-root.sh" "$1/tools/linuxstop.sh"
fi
```

Starting DHIS2 on Ubuntu if installed as a regular user:

```
#!/bin/sh

# get dhis2dir
abs=$(cd "$(dirname "$0")"; pwd)
dhis2dir=$(dirname $abs)
echo $dhis2dir

# for the war-file...
export DHIS_PROPERTIES_FILE=$dhis2dir/config/hibernate.properties

# set JAVA_HOME for tomcat...
#java=`which java`
#java=`dirname $java`
#export JAVA_HOME=`dirname $java`
export JAVA_HOME=$dhis2dir/jdk/

# start mysql
cd $dhis2dir/mysql/
./bin/mysqld_safe --port=3307 --default-storage-engine=innodb &
sleep 3
```

```
# start tomcat
cd $dhis2dir/tomcat
./bin/startup.sh &
sleep 2

# start firefox
firefox 127.0.0.1:8180/dhis/ &
exit 0
```

The following is the start/stop script for use in the national health toolkit package made for the meeting at WHO in February 2008.

```
#! /bin/bash

### BEGIN INIT INFO
# Provides:          nhtoolkit
# Required-Start:
# Required-Stop:
# Default-Start:     S
# Default-Stop:      0 6
# Short-Description: Starts tomcat and the included projects (dhis2,
openmrs, openhealth etc.)
### END INIT INFO

export toolkit_base="/opt/national-health-toolkit/"
export DHIS_PROPERTIES_FILE=$toolkit_base/config/dhis-hibernate.properties
export OPENMRS_RUNTIME_PROPERTIES=$toolkit_base/config/openmrs-
runtime.properties
export JAVA_OPTS="$JAVA_OPTS -Xmx256M"

# set JAVA_HOME for tomcat...
java=`which java`
if [ -z $java ]; then
  echo "No java found!"
  exit 1
fi
java=`dirname $java`
export JAVA_HOME=`dirname $java`

function start {
  echo "Starting tomcat..."
  sudo -u nhtktomcat "$toolkit_base/tomcat/bin/startup.sh" &
}

function stop {
  echo "Stopping tomcat..."
  sudo -u nhtktomcat "$toolkit_base/tomcat/bin/shutdown.sh" &
}

case "$1" in
  start)
      start
        ;;
  stop)
      stop
        ;;
  restart)
      stop
      sleep 10
      start
        ;;
  *)
        echo "Valid arguments are start, stop and restart..."
        ;;
esac
```