SUPPORTING FORMAL EXPRESSION OF ELIGIBILITY CRITERIA IN CLINICAL TRIALS

The implementation and evaluation of the Eligibility Criteria Builder

Master thesis

by

Bjørge Næss

Submitted in partial fulfilment of the requirements for the degree of

"Master in Information Science"

June 2009

UNIVERSITY OF BERGEN

Department of Information Science and Media Studies

ABSTRACT

A clinical trial is a study that assesses the effectiveness and safety of a new drug or treatment. To be able to generalize from the findings of the study, the clinical trial requires a representative sample of the target patient population. The target population is defined in terms of eligibility criteria that clearly describe the characteristics of patients enrolled in the study. The eligibility criteria are stated in natural language in its own section in a protocol document, which serves as the plan and detailed description of any prospective clinical trial. Having the eligibility criteria expressed as natural language has several drawbacks. First, it can lead to ambiguities and different interpretations among clinicians responsible for enrolment of patients into the study, which consequently may affect the safety of patients. Second, it provides no means of automatic eligibility checking against patient databases and electronic patient journals. The process of determining the eligibility of each patient therefore becomes a resource demanding and time consuming task. Formally defined computer interpretable eligibility criteria could potentially improve safety of involved patients and efficiency in patient enrolment. This thesis presents the Eligibility Criteria Builder that aims to provide a simple and pragmatic way of defining these rules using a user-friendly graphical user interface. The evaluation of the prototype indicated that the target users in general were positive to, and clearly saw the need for, a tool like this. The evaluation also pointed out weak spots and areas of improvements for the proposed prototype.

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

1 INTRODUCTION

A clinical trial is a study that assesses the effectiveness and safety of a new drug or treatment. To be able to generalize from the findings of the study, the clinical trial requires a representative sample of the target patient population. The target population is defined in terms of eligibility criteria that clearly describe the characteristics of patients enrolled in the study. In current clinical trials, the eligibility criteria are stated in natural language in its own section in a protocol document, which serves as the plan and detailed description of the prospective clinical trial. Natural language opens for ambiguities and different interpretations among clinical researchers. This is illustrated in an example by Chow & Liu (2004, pp. 611-612), where they refer to a protocol having an inclusion criterion that requires patients between 18 and 65 years of age without being clear on whether or not patients of 18 years of age would be considered eligible. If the eligibility criteria in a clinical trial are vague and ambiguous, it can potentially have serious impact on the safety of involved patients (i.e. the risk of including an ineligible patient due to misinterpretation). Furthermore, it directly affects both the external validity (generalizability) and reproducibility (i.e. the likelihood of arriving at the same findings in another, similar future trial Chow & Liu (2004)) of the findings of a clinical study.

When recruiting patients in a clinical trial that requires an estimated two-thousand subjects, the initial group of potential subjects may be multiple times larger in order to find the two-thousand that fulfil the eligibility criteria (for rare diseases the number may reach 10 000). The eligibility of each potential subject must be assessed by a qualified clinician. The information needed to assure the eligibility may be available from different sources like paper based patient journals, patient databases, lab tests or case report forms. For each patient, a clinician must look up information found in these different sources to verify that *all* inclusion criteria are met by the patient. For each exclusion criterion, the same task is repeated, to ensure that *none* of the exclusion criteria are met. Only after such thorough investigations can the patient be determined as eligible for participation and be enrolled into the clinical trial. Thus, the process of manually determining the eligibility of thousands of potential eligible patients becomes a resource demanding and time consuming task.

Having the eligibility criteria expressed in natural language also has other potential drawbacks. First, it can lead to ambiguities and different interpretations among clinicians

responsible for enrolment of patients into the study, which consequently may affect the safety of patients. Second, it provides no means of automatic eligibility checking against patient databases and electronic patient journals. So, in addition to being a time consuming task that requires a lot of resources, manual eligibility determination may also, at worst, affect the validity of the findings and the safety of patients involved in the study. Accordingly, formally defined computer interpretable eligibility criteria could potentially improve both the safety of involved patients and efficiency in patient enrolment.

Enrolled participants can be classified as ineligible at any stage in an ongoing clinical trial if their medical condition changes. Also, changes in the medical state of former ineligible participants can result in a change in the participant eligibility classification. In the former, it is of particular importance that clinical trial staff is informed about the change in eligibility status. Similarly, in the latter, it can be of considerable value to enrol these eligible "newcomers" as participants in the study.

Further, being able to express the rules for eligibility in direct conjunction with how the collected patient data will be stored in the ongoing study will give way for validity checks of enrolled patients, and could also enable monitoring of change in any patient-characteristics that may influence eligibility status.

Utilization of computer interpretable eligibility criteria would, however, require the needed information to be available from digitalized sources like electronic patient journals, electronic case report forms and digital lab-tests. As this is the goal of the OMEVAC project (see own description in 1.1), this thesis fits into the vision of a future where these digital resources will be available.

Within the frame of the OMEVAC project, a prototype has been developed to explore the possibility of providing an interface for stating eligibility criteria in an unambiguous machine-interpretable fashion as opposed to the current practice of expressing these criteria in natural language. This thesis presents the Eligibility Criteria Builder that aims at providing a simple and pragmatic way of defining these rules using a user-friendly graphical user interface. The evaluation of the prototype showed that the target users in general were positive to, and clearly saw the need for, a system like this. However, the evaluation also pointed out weak spots and areas of improvements for the proposed prototype.

1.1 The OMEVAC project

This thesis is a part of the OMEVAC project (Open Mobile Electronic Vaccine Trials) which is funded by the Norwegian Research Council (NFR). OMEVAC is led by the Centre for International Health (CIH), an interfaculty institution at the University of Bergen. One of the stated goals of the OMEVAC project is to move from a paper-based way of conducting clinical trials to exploit the possibilities given by computers and electronic capture devices. One of the achievements will be patient databases and electronic medical records. This prospect paves the way for automatic patient selection based on computerized eligibility criteria.

The Eligibility Criteria Builder prototype and the findings of this study will be a contribution to the OMEVAC project intended to support and improve the conduct of clinical trials in the developing world.

1.2 Target users

Formulating eligibility criteria requires knowledge about the diagnosis towards which the intervention is directed, any biochemical properties and compatibility of the intervention (known adverse reactions) and patient characteristics. For instance, some patients may have hypersensitivity or fatal adverse reactions to a drug. Obviously, these patients should not be exposed to it, and they should therefore never be included in the study.

The users of such a system are those responsible for administering and planning the execution of a clinical trial. The prototype must be easy to use and should not require the user to have programming skills, but at the same time be flexible enough to suit a wide range of different clinical trials. This means that the prototype must be data-model independent in order to able to cover the multitude of different properties that different clinical trials might have.

1.3 Research question

The aim of this project is to explore the feasibility of using a more programmatic, and thus formal, way of expressing inclusion and exclusion criteria for planned clinical trials. In order to be able to achieve this, a prototype of a web based application has been iteratively developed and evaluated.

The guiding research question for this project is:

How can a data model independent software tool be developed to support users to formally define criteria for patient eligibility in a clinical trial?

1.4 Terms

In this thesis, the terms "tool", "system" and "prototype" are all used interchangeable to describe the Eligibility Criteria Builder.

1.4.1 Clinical trial

A clinical trial can be defined in many ways. Friedman, Furberg, & DeMets (1996, p. 2) define a clinical trial as "(…) a prospective study comparing the effect and value of intervention(s) against a control in human beings". Piantadosi (1997) provided a simpler definition of a clinical trial as "an experimental testing of a medical treatment on human subjects" (quoted in Chow & Liu, 2004, p. 1). The characteristics of a clinical trial are described more in depth in section 2.1.

1.4.2 Study protocol

The study protocol is the document that details how a clinical trial is to be carried out and how the data are to be collected and analyzed (Chow & Liu, 2004). A widely adopted definition is provided by the International Conference on Harmonization (ICH) of Technical Requirements for Registration of Pharmaceuticals for Human Use: "a document that describes the objective(s), design, methodology, statistical considerations and organizations of a trial." (May, 1996, p. 6).

1.4.3 Eligibility Criteria

Eligibility criteria are defined in a separate paragraph in the study protocol. It is a list of requirements that a person must meet in order to be determined as eligible for enrolment into a clinical trial. "These criteria include demographic characteristics, prior or current diagnoses, laboratory-test results, subjective symptoms, physical findings, current or prior medications, and drug allergies" (Tu, Kemper, Lane, Carlson, & Musen, 1993, p. 341).

1.4.4 Inclusion Criteria

Inclusion criteria are a list of requirements that a patient must meet in order to be enrolled in a study. A patient must meet *all* inclusion criteria in order to be considered eligible.

1.4.5 Exclusion Criteria

The exclusion criteria are conditions that would disqualify the patient for enrolment in the study (i.e. known hypersensitivity to the proposed drug or treatment). A patient who meets *any* of the exclusion criteria will be considered ineligible for enrolment.

1.4.6 Patient recruitment

Patient recruitment refers to the process of recruiting patients to a clinical trial. This can be done by advertising in media, querying a large database to find potentially qualified people, or by medical practitioners reporting.

1.4.7 Patient enrolment

This is the activity of enrolling patients to an ongoing study. In order to be enrolled, a patient must meet the eligibility criteria.

1.4.8 Eligibility checking

Eligibility checking is the process of determining whether a patient meets the eligibility criteria.

1.5 Structure of the thesis

As this chapter has given an introduction to the project and its background, chapter two gives a more in-depth description of its context and other related approaches that utilize information technology in supporting the conduct of clinical trials. In particular other attempts that aim to formalize the study protocol, or crucial parts of it, are described. Chapter three provides discussion of the methodological approaches employed in this thesis, both in terms of research methodology and development methodology. In chapter four, the development process is described, together with a presentation of the resulting prototype.

Chapter five describes the evaluation design and presents the findings along with a discussion of their implications. Chapter six reviews and concludes the project and suggests ideas for future work.

2   RELATED WORK

This chapter presents a more in-depth description of a clinical trial and related approaches that utilize information technology in supporting the conduct of clinical trials.

2.1   What is a clinical trial?

A clinical trial is often understood as testing the effects of a drug, but the term can be broadened to include almost any type of intervention meant to improve human health, like running an attitude campaign to increase awareness about HIV/AIDS, or introducing mosquito nets as a means to prevent the spread of malaria. It is the fundamental activity of finding the answer to the question "Does the treatment work as intended?" In order to answer that question, a thorough study has to be carried out, usually involving a control group receiving placebo or most effective alternative treatment, as well as precise monitoring of the medical condition of involved patients.

Clinical trials are classified in four phases from I to IV (Chow & Liu, 2004; Friedman, et al., 1996). In the first phase (Phase I study), the treatment is usually tested for the first time on a very small group of healthy patients. The goal of a Phase I trial is to assess the safety of the treatment (i.e. determine a safe dosage and discover possible side-effects). Phase II trials involve a somewhat larger group of subjects (100 - 300). As in phase I studies, the goal is to assess the safety of the treatment and discover side-effects, but in addition, phase II trials also try to answer how well the treatment works. A Phase III trial is a comprehensive study administered to a significantly larger patient group (> 1000). After a treatment has passed a Phase III trial, it is authorised for prescription and approved for sale. The treatment will however continue to be monitored in long-term Phase IV trials. Because phase I-III studies are conducted over a shorter time span, the possible long-term side effects of the treatment cannot be fully known in these phases. Phase IV trials therefore continue to collect data about the treatment, to further assess its safety, effects and effectiveness.

All the clinical trial phases need subjects who meet the eligibility criteria. Each trial phase starts with a preparatory stage in which patients are recruited, eligibility status checked, and then enrolled (given they are determined as eligible).
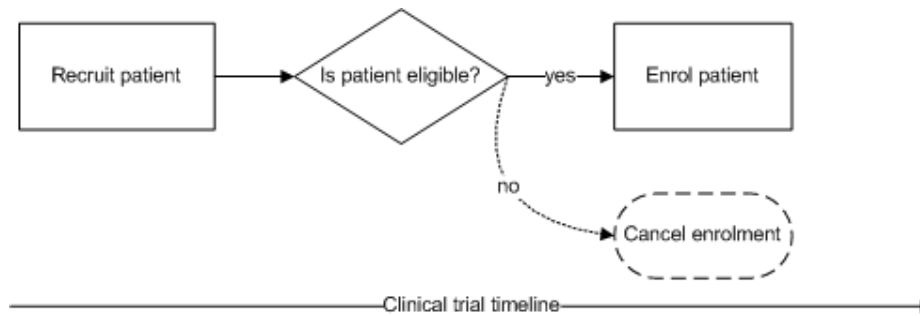
Figure 1: The recruitment and enrolment steps in the preparatory stage of a clinical trial

The process of finding people who are qualified to be participants in a clinical trial is divided in three main steps (Figure 1). First, a group of potentially qualified people are recruited as candidates for participation. The recruitment process typically involves contacting hospitals, advertising in media or querying a patient database to look for patients that may be considered candidates. Then, a more careful analysis of each patient's eligibility for the given study is performed by a study manager. If a patient does not meet the eligibility criteria, enrolment of the patient is cancelled.

A formalization of eligibility criteria may be useful in both the patient recruitment and the eligibility determination stage. If a large database of patient data and patients' medical history is available, formal eligibility criteria can be used to search for potential participants. In the eligibility criteria determination stage, formal eligibility criteria may not be used alone to determine a patients eligibility, but rather as an indicator that guides the study manager into making a qualified decision on whether to enrol the patient or not.

2.2   Clinical Trials Information Systems (CTIS)

A clinical trial information system (CTIS[1]) is a system that supports the management and conduct of a clinical trial. According to Oliveira & Salgado (2006), a "(...) CTIS must be understood as multi-modular systems because of the multiplicity and heterogeneity of the tasks that are part of the clinical trials cycle". The authors describe protocol authoring tools and patient eligibility determination tools as examples of active research areas that falls within this understanding of a CTIS. Most commercial systems today are what can be

---

[1] Sometimes referred to as Clinical Trials Management System (CTMS)

described as Clinical Study Data Management Systems (CSDMSs), because they are "essentially concerned with the delivery of valid and accurate data in conformity with the Good Clinical Practice (GCP) guidelines" (Oliveira & Salgado, 2006, p. 386). The focus on data management, as opposed to clinical trial management in a wider, organizational sense is a noteworthy difference between a CSDMS and a CTIS.

Even though clinical trial software exists, the existing systems are either based on proprietary data formats, forcing the clinical researchers to use (often expensive) software from a specific vendor or require highly experienced technical personnel (i.e. require consultants from the vendor itself) to operate. OpenClinica is an example of a feature-rich open source CTIS that supports management of all stages in a clinical trial, but it support any means of defining formal eligibility criteria. OpenMRS is an open source medical record system targeted for the developing world. It is implemented and used in production settings in several African countries today. However, it is not classified as a clinical trial management system, and offers no protocol authoring features. OpenMRS includes a cohort builder to select patients based on different criteria, which will be given more attention in section 2.6.

2.3   Protocol representation

Clinical Data Interchange Standards Consortium (CDISC) is a global, open, multidisciplinary, non-profit organization that has established standards to support the acquisition, exchange, submission and archive of clinical research data and metadata[2]. CDISC leads the Biomedical Research Integrated Domain Group (BRIDG) project, which is a collaborative initiative between different health research organizations that brings together different standards communities to clarify the semantics of clinical research across pharmaceutical, regulatory and research organizations (Fridsma, Evans, Hastak, & Mead, 2007, p. 130). Its main goal is harmonization of different standards of clinical data to provide interoperability and establish a shared semantic understanding in order to enable exchanging, sharing and systematic analysis and integration of clinical trials data (Fridsma, et al., 2007). The output of the project is the BRIDG model, which merges different standards through the process of harmonization.

---

[2] http://www.cdisc.org/about/index.html, Accessed May 29, 2009

A subgroup of BRIDG, the Protocol Representation (PR) group works on developing a standard structured protocol representation so that protocol information can be repurposed across multiple clinical research documents, databases, and systems from study start-up through reporting and regulatory submissions (Willoughby, et al., 2007). The PR group aims at creating a structured representation of inclusion/exclusion criteria part of the protocol (Willoughby, et al., 2007).

The BRIDG model presupposes that the eligibility criteria check is carried out by a clinician and does not support any way of assisting the selection of valid subjects. Whether a patient is to be selected or not is stored as an integrated part of the BRIDG data model, represented by the "PerformedEligibilityCriterion" class. This class has a "questionCode" attribute which points to a coded question (i.e. "Is the subject at least 18 years old?") and a Boolean (yes/no) "requiredAnswer" attribute (see Figure 2). This means, however, that information that might already be recorded with the system will not be re-used at the time of the actual patient selection. If the patient is already registered in the database, its age information is most likely also stored. So the answer to this question actually represents redundancy as the same information could be retrieved using the condition "patient.age >= 18".

PerformedEligibilityCriterion Attributes

Attribute	Type	Notes
questionCode	public : CD	The complete text of an individual question/criterion on the eligibility checklist of a protocol. For example, Is the subject at least 18 years old?
requiredAnswer	public : BL	The reply necessary to include/exclude a potential subject on a study.
displayOrder	public : INT	The sequence or position of a component in a list of question or data items.
notApplicableIndicator	public : BL	Specifies whether the specific eligibility criterion is not applicable to this participant.

Figure 2: How the BRIDG model represents eligibility criteria (BRIDG Release 2.1, p. 59) [3]

At the time of writing, no concrete proposal exists on how a structured representation of eligibility criteria that re-uses data that may already be collected, can be achieved. The Eligibility Criteria Builder may constitute such a proposal.

---

[3] Excerpt from the Static Elements Report.rtf file in BRIDG_Release_2.1_Package.zip, downloadable from http://gforge.nci.nih.gov/frs/?group_id=342

2.4 Patient selection

As most commercial systems focus on the data management part of clinical trials, few of them tend to incorporate additional features to automate complex procedures that traditionally are done manually by a clinician. Oliveira & Salgado claims that "More advanced features like patient recruitment, eligibility checking, treatment allocation and adverse events reporting, are seen in only one or two commercial systems" (2006, p. 386), but these systems are not named.

However, the idea of automating eligibility checking of patients using a software tool is not new. Tu, et al. (1993) developed a language for expressing eligibility criteria in a machine-readable way (e.g. HIV+ = true) OR (AIDS = true). For each criterion, a corresponding template had to be written for translation into a valid database query. This means that the criterion itself is not directly usable in finding matching patients. For every new criterion added, a corresponding query has to be defined. In the Eligibility Criteria Builder, the criterion becomes the query, and no additional mapping is needed.

Ohno-Machado, Wang, Mar, & Boxwala (1999) developed a support system for clinical trial eligibility determination to help patients or their care providers to find ongoing studies of which the patient may be eligible (Ohno-Machado, et al., 1999). This situation is somehow different from the situation where a clinical study coordinator looks for potential participants by querying a database of patients, but the mechanisms used to match patients against eligibility criteria would be the same. In this study, the eligibility criteria were taken from unstructured protocol documents and manually translated into machine-interpretable statements using the Arden Syntax (see section 2.5). "The translation of the original free-text criterion descriptions (...) into a machine-interpretable representation was largely a manual process performed by informatics fellows and faculty in our laboratory" (Ohno-Machado, et al., 1999, p. 341). The intention with the Eligibility Criteria Builder is that this translation can be done by a study manager without the help from a programmer.

Fink, et al. (2004) summarize a comprehensive amount of previous approaches to support both finding relevant trials for a patient, and finding potential patients for a clinical trial. However, the majority of these approaches are a matter of developing tools for entering the specific data needed to determine a patient's eligibility status and holds the form "Has the patient had any previous cases of disease x?". Thus, the development of such tools may

become a one-time activity valid only for a specific trial and does not take into consideration that the data (or at least data structure) needed for answering such a question may be available in an existing, available patient database.

In OpenClinica, patient selection is a manual process as there is no feature of defining the inclusion/exclusion criteria as rules that will later be used in automatic selection of eligible patients. In the current version of OpenClinica (2.5) patients are marked enrolled to a study *after* they are determined by a clinician as eligible.

2.5   Defining rules in healthcare - the Arden Syntax

The Arden Syntax is a long-lived syntax specification for defining rules that applies in healthcare practice. It is used in defining Medical Logic Modules (MLMs), which is single rules that contains data and logic to help a clinician to make a decision at the point of care. It is intended to result in computer-readable rules that can be used to trigger alerts, reminders or suggestion about a patient at the time of encounter with a clinician. However, an MLM includes a part that references local, institution-specific healthcare data. Also, with Arden, a separate compiler must be written in order to integrate it with institutional data. As a consequence, in order to make a MLM written for one institution fit in to the context of another is a complex and time consuming task that requires highly skilled personnel. Attempts have been made to extend the Arden syntax to cope with the problem referred to as the "curly braces" problem (as local linkages are defined inside curly braces). The curly braces problem is still a present issue without any clear and immediate solution, and will most likely continue to be so until a globally accepted EMR standard is implemented and adopted across medical institutions.

Another shortcoming of the Arden syntax is the lack of object-oriented features. The available data are limited to primitive (boolean, integers), and scalar (arrays, strings) only. Related pieces of data cannot be logically organized together as objects with attributes. Neither can data be accessed and references to as properties in a directed graph of objects with properties (objects as vertices and properties as edges). Attempts have, however, been made to introduce some object oriented features to the Arden syntax. (Jenders, Corman, & Dasgupta (2003) suggests the READ AS <object type> statement that instead of returning a list of primitives from the database, returns each matching row as an object where each column is mapped to their corresponding properties of the object type. This approach may

ease some of the shortcomings of the Arden syntax, but it does not solve the curly braces problem as it still requires references to a site-specific database. Thus, there is still a need for further decoupling of the database and how data is stored physically and the rules that access this data.

Another attempt to improve the Arden syntax closer to the approach described in this thesis is the one by Choi, Bakken, Lussier, & MendonÇA (2006). Here, the authors argue that while the Arden syntax provides a mechanism to represent machine readable procedural knowledge, it is less suited for direct use by humans. The human readability aspect of MLMs has received little attention, but is nonetheless important of the "essential role of clinical experts in the development and maintenance of MLMs" (Choi, et al., 2006, p. 221). The Eligibility Criteria Builder is intended to produce rules that represent similar logic on a format that is intuitive and easier to understand and work with for a non-programmer.

2.6 Retrieving patients

The OpenMRS Cohort builder (Figure 3) provides the possibility to search for a group of patients based of different criteria as personal data, demographics, drug orders and encounters. The searches can later be combined to create a cohort, represented as a subsection of patient Ids (a cohort is a population sample that share similar characteristics, i.e. age and/or citizenship). The cohorts are saved as a one-time cross-section of the matching patients and cannot be re-used to find matching patients that may be added to the database in the future.

Figure 3: The OpenMRS Cohort builder

The OpenMRS cohort builder does not support conditions spanning more than one attribute in the object graph. For example, searching for patients whose country of the preferred address is "Uganda" would pose a great challenge. You can require the citizenship to be Ugandan, but that is something different, as the patient's current address may be somewhere else. The user interface of the OpenMRS cohort builder is also tightly coupled with the OpenMRS system itself, making it non-generic and thereby less suited for integration with other data models.

2.7 Adverse Event Reporting

As there are inclusion/exclusion criteria, there are also rules for adverse event reporting in a clinical trial. The caAERS system is a open source tool developed and maintained by the

Cancer Biomedical Informatics Grid (caBIG) intended to support the definition and reporting of adverse events that can occur in a cancer trial (Whippen, Deering, & Ambinder, 2007).

Figure 4: The caAERS rule editor

The caAERS system (Figure 4) is web based, and provides some of the same features as the Eligibility Criteria Builder. It is however limited in functionality in some respects. For example, only subsets of root-level attributes are available for use in conditions, grouping of conditions is not allowed and the only allowed logical operator is AND. On the other hand, it provides richer functionality as it allows for custom actions based on rules. In the Eligibility Criteria Builder, the rules are used for either "Include" or "Exclude" actions. They could, however, easily be used to trigger other types of actions, such as adverse event reporting.

3   RESEARCH METHODOLOGY

In the same way as a clinician intervenes in humans in order to solve or prevent a problem (i.e. diagnosis) faced by their patients, this thesis proposes an intervention to solve a problem faced by clinicians in their everyday work. This kind of "intervention research" is not uncommon in social sciences and is dominated by two major methodological approaches, namely action research and design science (or design-based science).

3.1 Action research

The term action research was first coined by the German-American psychologist, Kurt Lewin in his paper "Action Research and Minority Problems" from 1946. Lewin described action research as "a comparative research on the conditions and effects of various forms of social action and research leading to social action" (Lewin (1946) as quoted in Susman & Evered (1978)). Action research originated as an approach to integrate research findings with practice in order to help the resolution of a social problem. Improvement and involvement are key features of action research. There is, first, the improvement of a practice of some kind; second, the improvement of the understanding of a practice by its practitioners; and third the improvement of the situation in which the practice takes place (Robson, 2002). This claim by Robson indicates that the improvement comes first, and then the knowledge is generated on the basis of how the improvement affected the surrounding environment.

In action research, the researcher does not behave as a passive observer, but rather enter into the situation as an active participant working in close cooperation with the involved people to introduce a solution to the problem. The solution to the problem will be offered based on knowledge gained from initial observation of the problem area. The situation is then evaluated, to identify how the change affected the studied situation and the knowledge gained through evaluation is the outcome of action research. Both knowledge about the different aspects of the problem area and the effects of the introduced change are of high value.

3.2 Design research

In design research, the intervention is in the form of an new and innovative artefact that seeks to extend the boundaries of human and organizational capabilities (Hevner, March, Park, &

Ram, 2004).

Design science has its origin in the Information Systems (IS) field (with its roots in artificial intelligence) and emerged as a counterpart to the predominant explanatory and predictive research traditions. Simon (1996) argued that design research is about finding out *how things ought to be* (Adikari, McDonald, & Collings, 2006). This may sound similar to the principles of action research which purpose is also to find a solution to a problem faced in a domain of interest.

Design research does not only ask what can be done to improve the situation and how the improvement ends up affecting the surrounding environment. It is just as much about describing the actual design process. By asking "how can we best create the solution Y in order to solve the problem X in the best possible manner", design research also emphasizes the knowledge gained during the process that led to the solution.

3.3 Action research vs. Design research

Both action research and design research view the researcher as an intervening part with a subjective opinion, not merely a neutral, objective observer as in traditional explanatory and predictive sciences.

The distinction between design research and action research is not obvious. In fact, it can be argued that the two approaches share so many characteristics that they can be considered as similar to one another (Järvinen, 2005). Design research and action research differs, however, on some key points. Design research is usually researcher-initiated and grounded in theory. This point is elaborated by Wang & Hannafin (2005): "Before conducting design-based research, researchers (...) examine literature and available design cases, and identify gaps to ensure the value of the research (Edelson, 2002) and to identify existing problems and issues" (Wang & Hannafin, 2005). In action research, on the other hand, the research is usually initiated by practitioners in an immediate problematic situation. Theory, if it exists, is then used to propose a solution to the problem, and the proposed solution is in turn evaluated to build theory.

Put simply, design research acts upon theory, while action research theorizes upon practice. For example, design science requires significant literature review and theory generation, uses

formative evaluation as a research method and utilizes many data collection and analysis methods widely used in quantitative or qualitative research (Orrill, Hannafin, & Glazer (2004) quoted in Wang & Hannafin (2005)). Action research generates theory grounded in action by applying theory in diagnosing situations and developing interventions, and by evaluating interventions to test the underlying theory (Andriessen, 2006, p. 5).

In both cases, the purpose is to contribute both to practice and theory through iterations of acting and evaluation (Andriessen, 2006; Järvinen, 2005; Rapoport, 1970).

3.4 Choosing a research methodology

Because of the close resemblance of action research and design research as shown by Järvinen (2005), it is not very meaningful to claim that this thesis belongs to one or another of these approaches. Rather than classifying the methodological approach of this thesis as either action research or design science, I would argue that it uses elements from both of them. It may belong to action research in the way that its motivation arose from a practical problem faced by practitioners, rather than from a thorough literature review leading to a discovery of a gap in previous research. It may belong to design research in the way that the development of the solution (prototype) is not merely a means to an end, but just as much a way of enabling the researcher to learn about the real world, how the artefact (prototype) affects it, and how users appropriate it (March & Smith, 1995).

3.5 Development methodology

Sommerville defines a prototype as "an initial version of a software system that is used to demonstrate concepts, try out design options and, generally, to find out more about the problem and its possible solutions" (Sommerville, 2007, p. 409). There are two common prototyping approaches: evolutionary (or incremental) prototyping and throw-away prototyping (Avison & Fitzgerald, 2003, pp. 89-90). The latter is usually a quick non-functional (or limited in terms of functionality), mock-up or sketch of the intended system, used to visualize and ease the understanding of the system or specific features of it that may be different to imagine otherwise. Throw-away prototyping is quick (the user can typically see a sketch of system and test some functionality in a couple of days), but has, however according to Avison & Fitzgerald (2003), a few limitations. Most notably, it does not scale

well and is unsuitable or difficult to integrate with operational systems. Another shortcoming is the problem of spending time developing something that never will be used later. When the customer (or end-user) decides that the prototype is "good enough", the prototype is thrown away, and the final system is being implemented. Herein lays the key difference between evolutionary prototyping and throw-away prototyping; in evolutionary prototyping, the development is considered "done" when customer satisfaction is met, while in throw-away prototyping that is when the real development starts. Nothing is thrown away using the evolutionary approach, except features that may have been disclosed as unnecessary through the user-evaluation. Evolutionary prototyping is about constantly evolving the prototype so that it steadily moves towards a finished product.

The development of the Eligibility Criteria Builder followed an evolutionary prototyping approach, in which the development took place in three iterations (or increments), with each iteration delivering an improved version of the prototype. The Eligibility Criteria Builder should still be regarded as a prototype as it used to demonstrate a concept and find out more about a problem and its possible solutions, which fits well with Sommerville's definition of a prototype as initially quoted in this section.

3.6 Usability evaluation

What is usability? The most common definition of usability is the one given in the ISO standard 9241:

"The effectiveness, efficiency and satisfaction with which specified users achieve specific goals in particular environments"

The different components of this definition, effectiveness, efficacy and satisfaction are further explained by Dix, Finlay, & Abowd (2004):

o Effectiveness refers to the accuracy and completeness with which specified users can achieve specified goals in particular environments
o Efficiency refers to the resources expended in relation to the accuracy and completeness of goals achieved
o While satisfaction is the comfort and acceptability of the work system to its users and other people affected by its use

According to Dix et al., usability evaluation has three main goals: (1) to assess the extent and accessibility of the system's functionality, (2) to assess users' experience of the interaction, and (3) to identify any specific problems with the system (Dix, et al., 2004, p. 319). The first goal is about finding out to which extent the system solved the task it was intended to solve from a users perspective. The second goal is concerned about how the users experience the interaction with the system. A clear definition of usability and the goals of usability evaluation enable the development of different techniques to assess the extent to which the software meets the user requirements.

Several evaluation techniques exists and choosing between them is a matter of available resources, at what stage in the development lifecycle the evaluation takes place, nature of study (laboratory/field), level of objectivity needed, and type of measure required to name a few (Dix, et al., 2004). It is impossible to do all types of evaluations and there exists no evaluation technique that covers all possible aspects of the studied subject. Therefore, choosing an evaluation technique is always a trade-off and a matter of selecting the most appropriate technique within the limitations of the study.

At the end of the first development iterations, usability evaluations were conducted by taking notes while going through the features of the new version.

The characteristics of the Eligibility Criteria Builder are:

(a) It is a web-application
(b) It is intended to support a complex task in order to solve a real-life problem faced by clinical researchers
(c) It is a prototype, in an early development stage, meant to test a concept/idea

Because of (a) it can easily be administered to people around the world. This affects availability of test-users that potentially can be recruited from all over the "connected" world. On the other hand, because the target users are clinical researchers (b), it limits the group of potential participants to those with this background (ideally, they should also have experience from conducting clinical research in the developing world), i.e. it is not a matter of asking fellow students (which are easily accessible) for their participation.

Because of (c) it is not possible to observe the system in a completely natural setting. The evaluation tasks are artificial, so the participants have to imagine themselves building the

given eligibility criteria tasks for an imaginary study. This excludes field study as a feasible technique in evaluating the prototype.

The time constraint posed by a master's thesis and the possible remoteness of available participants meant that the best available option for assessing the usability of the prototype was using a questionnaire. A combination of the Systems Usability Scale (SUS) and open questions were used. See section 5.2 for a description of SUS.

4 SYSTEM DESIGN AND IMPLEMENTATION

This chapter includes a description of the development process in the design research and the incremental evolutionary prototyping approach that was adopted.

As opposed to throwaway prototyping, evolutionary prototyping involves continuous refinement of the system. The prototype was developed in three iterations, each resulting in a set of improvements from the previous one.

4.1 Initial requirements analysis

The initial requirements analysis was based on conversations with different clinical researchers with experience in the planning of clinical trials. The conversations took place during the OMEVAC kick-off meeting in Kampala, Uganda. The requirements was then refined and described in more technical terms in the next section.

4.1.1 General requirements

A system enabling clinical researchers to formulate a list of computer-interpretable eligibility criteria would require interaction with an existing data model and available resources (demographics data, observational data, case report forms, etc.). The prototype should not be bound to a specific data model, but rather developed as a generic tool that could be adapted to any data model with as little effort as possible. During development and evaluation, there was however a need for an example data model to which the system was tested against. Therefore, the prototype was developed using the OpenMRS API[4] as an example of what the data model might look like. The OpenMRS data model is based on the data model of the Regenstrief Medical Record system which has been in production use more than 30 years and contains patient demographics, drug orders, observations, patient encounter forms, and a concept dictionary (Mamlin, et al., 2006). It is written in Java and made freely available by download from the OpenMRS website as a Java Archive (jar) file ready to be integrated in other Java

---

[4] API stands for Application Programming Interface, and is an abstraction of the underlying relational database model. The OpenMRS data model therefore refers to the OpenMRS API throughout this thesis

based systems.

4.1.2 Initial functional requirements

A rule editor which allows the user to do the following:

- o Define variables
- o Define variables based on aggregated data
- o Define variables based on expressions or other variables
- o Use variables in logical expressions
- o Declare variables based on available resources (patient journals and case report forms)
- o Continuous assisting the user in the process of formulation by:
    - ▪ Instant hinting of available options or possible next steps (i.e. selecting a patient blood type property).
    - ▪ Storage and retrieval of expressions or rule sets for later use.

4.1.3 Non-functional requirements

- o A graphical user interface enabling easy formulation of eligibility criteria
- o Expressions should be valid in the given context (with the same resources available)

4.1.4 Required resources

The system will need access to at least one of the following type of resources:

- o Electronic Patient Journals
- o Electronic Case Report Forms

4.1.5 Expressiveness

The expressions formulated in the system should be ranging in complexity from simple to complex where the most simple could be requiring "patients to be above 18 years old" to more complex, expressions spanning across collections based on selection criteria on the collection items (i.e. patient has an observation where the question is "are you pregnant?" and the answer is "yes"). Because a patient can have several observations, the collection selection criterion is "question is "are you pregnant"". In addition, the system should enable the basic

logical operators AND and OR, and parentheses (or groups) that can be negated.

4.2 System design

The prototype was developed iteratively following an incremental, evolutionary prototyping approach. The prototype was continuously improved in accordance to feedback given by potential users including Jørn Klungsøyr at Centre for International Health at the University of Bergen.

The idea of the eligibility criteria builder emerged from a plan of developing a more general study protocol management tool. A formalization of the eligibility criteria section in the protocol was identified as a significant improvement to the current practice. At the same time, it is a great challenge because it deals with formalization of natural language and represents written statements in natural language in a computer-understandable format without compromising the semantics of the written statements on the way. In addition, a system like this should be usable for study managers who may not be experienced programmers. For a skilled programmer, it is undoubtedly easier to formulate criteria in a programming language syntax using a text-editor or the supporting features provided by an IDE (Integrated Development Environment). The idea of developing the prototype as an IDE plug-in was explored in the beginning, but later rejected because of the great complexity of this approach and because of the programming knowledge required from the user.

4.2.1 Original design - sketch
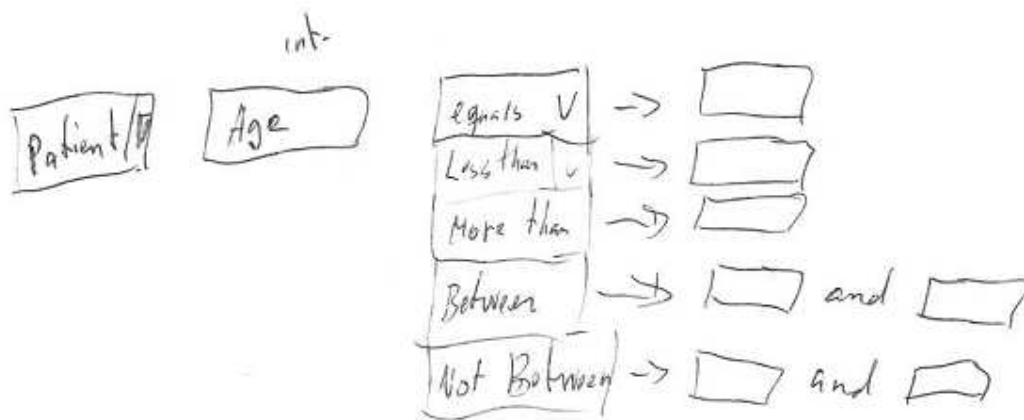
Figure 5: Early conceptualization of the prototype on a piece of paper. Each rectangle represents a selection field.

As shown in Figure 5, the sketch illustrates what happens when the user selects the age property of the patient class. Because the age is an integer value, the list of possible operators reflects those valid for numeric values. The last empty rectangle would be where the user typed the value(s) for the condition.

4.2.2 The programming language approach

The idea of developing the prototype as an integrated development environment (IDE) was considered as a feasible alternative in the beginning. A mock-up was therefore developed using the Netbeans Platform[5], which is a framework for developing Java based desktop applications built on top of the Netbeans IDE application. Basically, it works by selecting the essential features needed in one's custom application, and extending these functions as desired. This means that the mock-up could quite easily be founded on the IDE features (i.e. syntax highlighting) of the Netbeans platform. The language selected for the Eligibility criteria syntax was a simplification of the Groovy[6] language. A Groovy language definition

---

[5] http://platform.netbeans.org

[6] http://groovy.codehaus.org/

25

file for the Netbeans platform was described in a blog entry by Geertjan Wielenga[7], and based on that, a mock-up was implemented on the basis of the initial requirements.
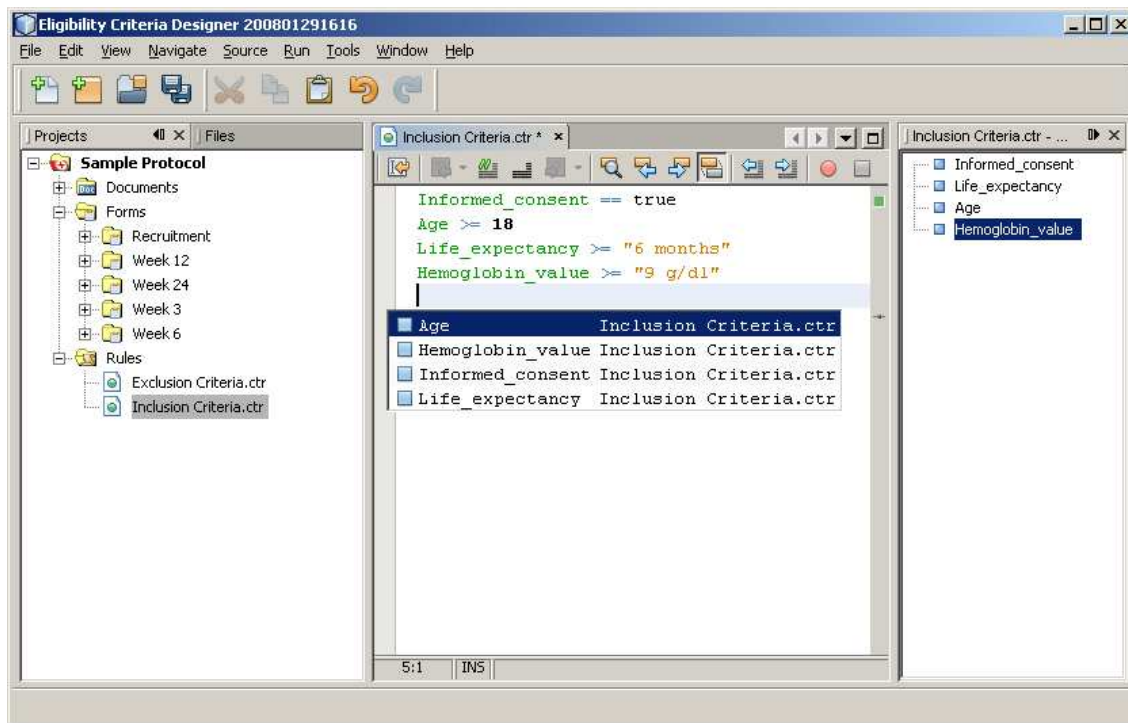
Figure 6: Mock-up using the Netbeans platform

As seen in Figure 6, the user can state its eligibility criteria in an integrated development environment (IDE). The project explorer to the left shows the different parts of the protocol, with the rules for eligibility organized in its own section/folder as separate files with the CTR file type (clinical trial rule).

Editing a file with the CTR extension will make the editor view in the middle apply the appropriate syntax colouring scheme and perform syntax validation according to the defined language definition file. Each criterion is stated on a new line in as conditional expressions that evaluates to true or false. The dropdown list below the cursor shows the valid properties available for user selection.

The right pane in Figure 6 lists the criteria in the current edited criteria file for easy navigation between them.

---

[7] http://blogs.sun.com/geertjan/entry/how_to_write_a_groovy

The programming language approach was however found to have several drawbacks:

- Heavyweight application framework: This meant longer compile-time and less rapid development of new features.
- Complex application framework: The Netbeans code base is comprehensive and much more time was spent on figuring out how the different parts worked (and making them work *together*) rather than focusing on implementing the actual required features.
- End user availability and client requirements: The resulting application was about 10 MB of size, and had to be downloaded and installed on the client computer.
- Higher technical knowledge required by the end user. This approach would demand the end user to hold at least some programming knowledge.

The programming language approach might have been less restricted and provide a great deal of freedom for the user (i.e. placing the cursor anywhere in the file and start typing), but because of the reasons given above, a decision was made to change the approach from a traditional desktop application development, towards developing a web-application with a less complex dropdown-based user interface.

4.2.3 Web application using Grails

Grails is a web application framework that facilitates rapid web development, and was selected because of its reputed ease of implementation and suitability for integrating with existing Java libraries (like the OpenMRS API).

"(...) Grails, which is an open source framework that aims to simplify Web development. Grails is written in Groovy, a dynamic, object-oriented language that runs on the JVM (Java Virtual Machine). Because Groovy interoperates seamlessly with Java, Grails can leverage several mature Java frameworks." (Richardson, 2008)

Using Grails and developing the prototype as a web-application facilitated rapid development of new features and put less system requirements on the user as it can then be opened and run in a browser from anywhere.

First prototype: System architecture and user interaction

In the first prototype using the web application approach, the architecture infrastructure was

established together with a preliminary version of the user interface.

On an architectural level, the prototype works by the user interface (UI) in the front-end listening for user actions (i.e. a mouse click on a button) and giving appropriate responses to those actions. If the action requires information from the server, the front-end sends a request to the server, asking for the needed information. The server responds to the client-initiated request and returns the appropriate data which is examined by the front-end and presented to the user through the user interface. Figure 7 shows a model of the system architecture.
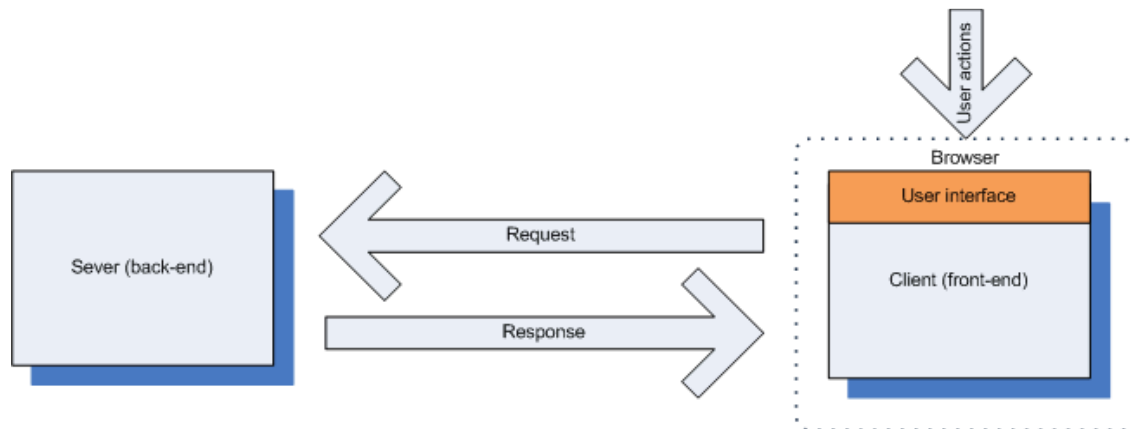
Figure 7: Overall system architecture

Apart from the system architecture, an early version of the user interface was implemented in this iteration. The ability to provide the user with hints on what properties and what operators were available for selection were considered as of high priority in the requirements analysis. As the approach moved away from the notion of the prototype being an IDE facilitating simplified programming, so some of the requirements initially stated were not as relevant anymore, resulting in the requirements being re-prioritized. Those assisting the user in the formulation of the criteria were put in higher priority than those dealing with the notion of variables. This is reflected in the user interface, as shown in Figure 8

The OpenMRS data model was used as an example for testing with the properties of its Patient class as the starting point for each criterion. However, as the aim was to develop the prototype as generic as possible, direct dependencies between the front-end and the OpenMRS library were avoided.
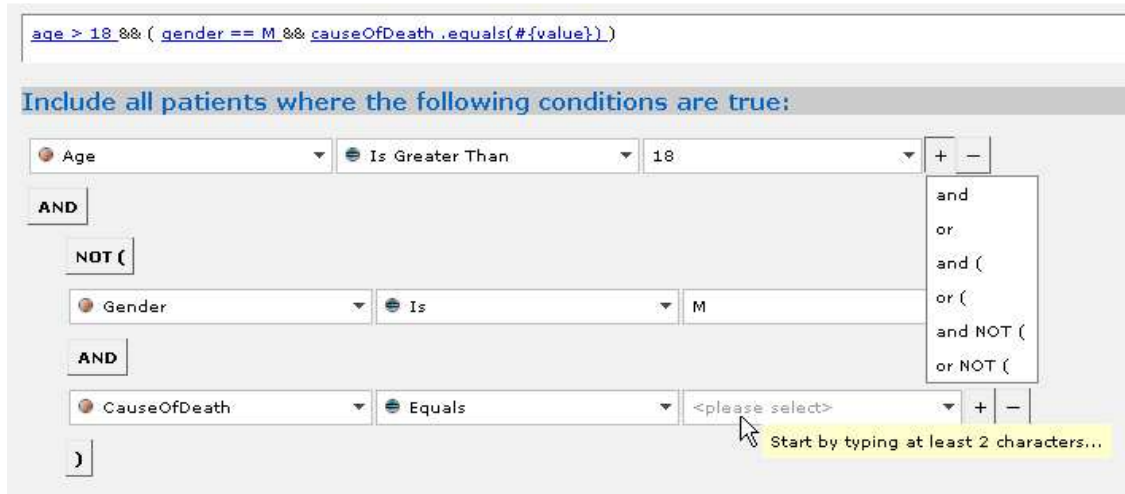
Figure 8: The first web-application prototype

Figure 8 shows three properties of the OpenMRS Patient class used in an expression. Each line has a dropdown box for choosing the Patient property, a dropdown box for selecting a valid comparison operator, and an input box for entering the value to compare with. The list of valid comparison operators depends on what property is selected. For instance, the property "Age" refers to an integer value, and therefore the user can choose between all valid integer operations. "CauseOfDeath" is a concept, and the user is therefore given the opportunity to search for valid concepts stored in the database. The user can type in at least two characters and the system will list all the concepts that start with these two characters. After the end of each line, the user is given the option to add a new condition (plus button) by selecting the logical operator (AND/OR) to use in between, or deleting the condition by clicking the button with a minus sign on it.

The corresponding logical expression is shown in the upper box in Figure 8. This reflects the expression formulated in the UI as source code in the Groovy language.

Second prototype

In this iteration, the user interface was improved in terms of stability, visual impression and functionality. Improvements include the ability to perform basic operations like storage, retrieval and deletion of expressions (as specified in the requirements analysis), naming expressions and viewing them in XML format. The user can select any of these operations from buttons on the introduced toolbar, as seen in Figure 9. End-user documentation was also

29

included in this version.
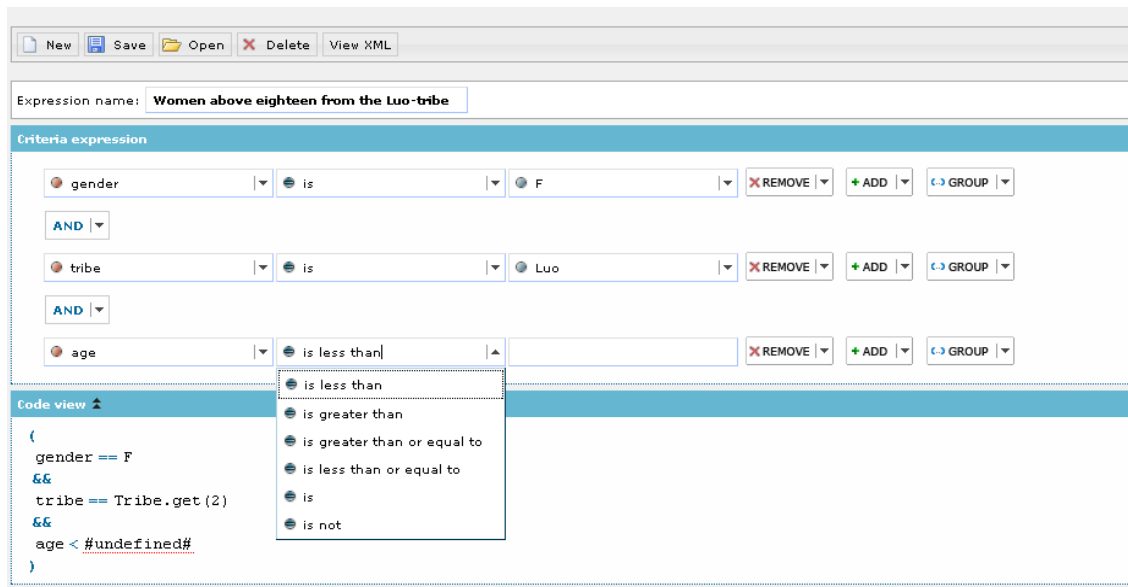
Figure 9: The user interface from the second iteration

The buttons at the end of each line were improved with icons, and options for adding new groups were placed in a separate button. Also, the code view was moved below the Criteria expression, to put the workbench in focus (see Figure 10 for the names of the different parts of the UI).

For this iteration, tasks covering the range of functionality offered by the prototype were designed in order to test the prototype. The test tasks for this iteration were the same as the one used in evaluation (see section 5.3.5).

The tests were done by Jørn Klungsøyr at Centre for International Health. He revealed several usability issues:

- It was impossible to remove a group. If the user made a mistake by clicking the add group button, there was no way to remove this expression group again.
- It was not possible to add a group immediately after another group. This had to be done by first adding a group, then adding two other groups after the expression inside this group.
- Most notably, if one started by adding a group, there was no way of adding a new expression after it.

Final prototype

Based on the feedback from the second iteration tests, the final prototype was developed.

It was suggested to add the "remove, add and group" buttons before or after a group-parenthesis. This led to the development of the Expression Toolbar (see Figure 10), responsible for adding and removing expressions and expression groups before or after both single expressions and expression groups.

This resulted in these three buttons appearing rather more frequent than before, and to tone down their visual impression and dominance in the GUI, these buttons were reduced to icons only (with a describing tooltip text) and made transparent when inactive (opaque on mouse over). A more thorough description of the user interface in the final prototype follows.

The final user interface

The user interface in the final prototype now consists of four different parts. A toolbar, an information pane, the expression workbench and a code view. The toolbar offers basic operations for the creation, saving, re-opening and deletion of expressions. For demonstration purposes, an option to download and view an XML version of the current expression is also available. Lastly, the toolbar has a button that opens the user-documentation in a new window. This makes the documentation available to the user at any time.
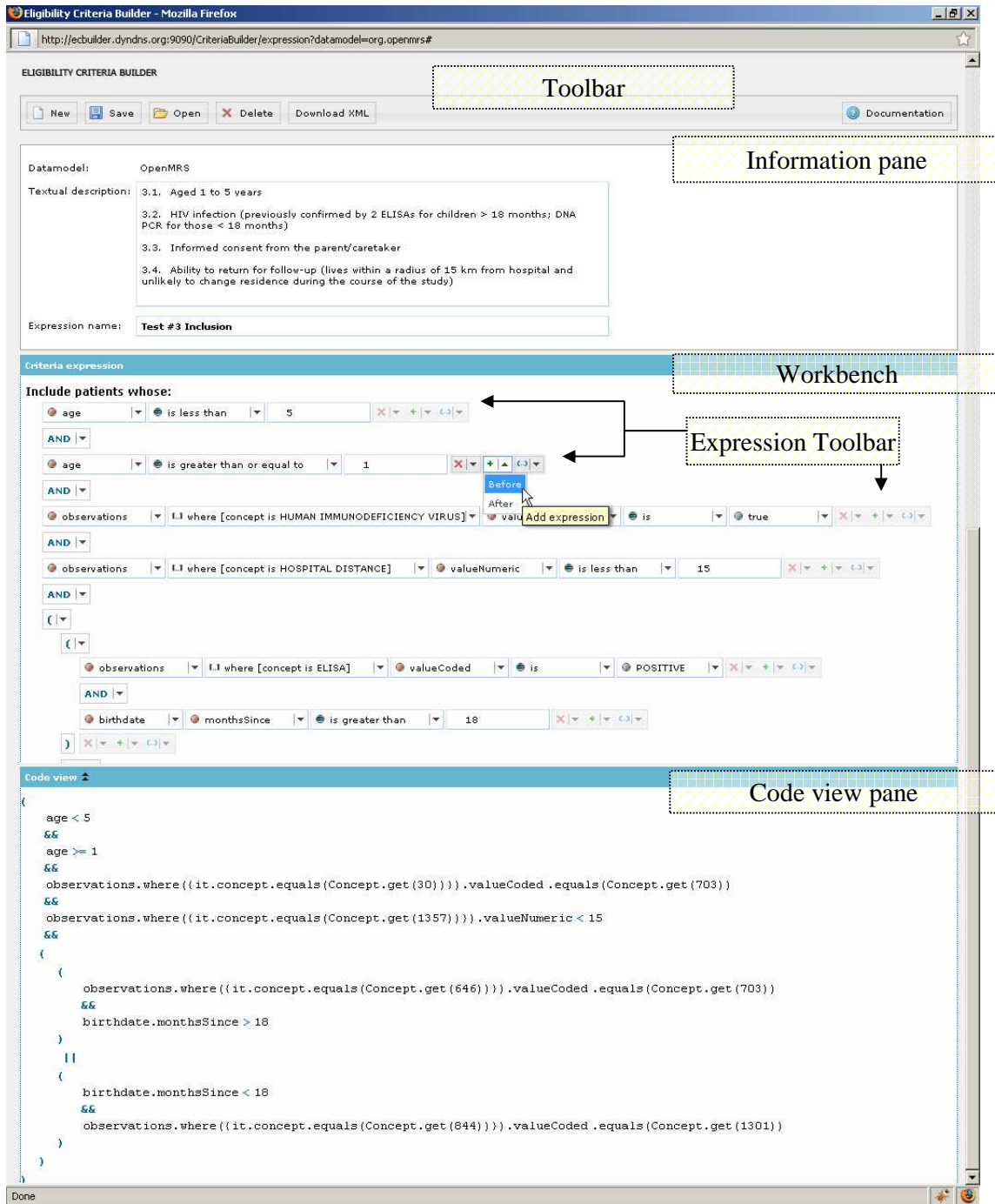
Figure 10: The final prototype and its different parts

The information panel contains information about the current data model. During the tests, this panel also included task-number and criteria type selection boxes and a field for the textual description of the criteria expression. See section 5.3.5 for more information about the user tasks used in evaluation.

The workbench pane is where the user formulates his/her criteria by building the expression piece by piece. When creating a new expression, three separate dropdown-boxes are shown. The first is where the user selects the property to use in the condition. When the user selects a property from the first dropdown, the next one will be updated to contain the valid comparison operators for the data type of the selected property (remember that a property is actually a get-method with a return-type). For example, if the property type is an integer, the next selection box will display a selection of comparison operators valid for integers (i.e. less than, greater than... etc.). If, on the other hand, the selected property returns a complex data type, like a patient or person, the comparison-operator selection box will also include the properties of the return type (i.e. birthdate for a person). If the user selects one of these properties, a new dropdown will be added before the comparison operator selection box, containing the chosen property, and the comparison operator will be re-populated with comparison operators valid for the data type of the selected property (in the case of birth date: "is before", "is after", etc.).

The last field is always the value in the expression. The value can be either primitive, (numbers, strings) or complex (referring to actual instances, i.e. the user with id=2).

The registered lookups, as described in section 4.3.2 - Lookups, determines what values should be suggested as alternatives for the different data types.

When the user is finished with the first condition, he/she can go on by adding another condition by clicking the "Add" button to add a single condition or the "Group" button to add a grouped condition. Before adding a new condition, the user must decide whether to add the condition before or after the current. When the condition is added, the user also needs to select the logical operator (either AND or OR) between them. When adding a grouped condition, the user can also optionally negate the newly added group (i.e. put a NOT before it).

The code view is for demonstration purposes only, showing the current expression as a Groovy expression.

Creating a new expression from scratch

To create a new expression, the user clicks the "New" button on the toolbar.

Along with a field for entering the expression name, three blank fields will appear in the workbench (see Figure 10). The first field gives a list of all the properties associated with the root-class as defined in the backend.

The user will need to give the expression a name by filling in the "Expression name field" This will typically be a textual description of the criterion the user would like to create. In order to force the user to provide a name for the expression, a dialog box will appear if he/she tries to save the expression without yet giving it a name.

Open an existing expression for editing

To open an already existing expression, the user clicks the "Open" button on the toolbar. This will display a list of all the existing expressions in the database. The user then selects the expression he/she wants to open from the dropdown list. The expression will now appear on the workbench, ready to be edited.

Deleting an expression

To delete an expression, the user clicks the "Delete" button on the toolbar and will then be asked to confirm that the expression should be permanently removed.

4.3 Implementation

4.3.1 Technologies used

Several different technologies were used in the development of the prototype. They are listed and described below.

Grails, Groovy, JavaScript

The back-end was implemented using Grails, an open-source, Java-based web application framework. Grails enabled easy integration with the OpenMRS API, and facilitated rapid development of the prototype back-end. Grails is itself built on top of other technologies such as the Groovy Scripting language for Java. The flexibility offered by Groovy made it possible to add functionality to built-in Java-classes and imported library classes.

JavaScript was used on the front-end to request, receive and manage the information from the back-end. Requests were sent from the UI whenever the user action required it.

The Java Reflection API

A crucial feature of the Java programming language utilized in this project was the ability to read metadata of the different language constructs (classes, data types, methods, etc.) through the Java Reflection API which allows the program to introspect on itself at runtime. For example, by using the introspection features, it is possible to write code that can tell the user what methods and class it is running inside. A description of how reflection is utilized in this project is presented later in this chapter.

Dynamic features of Groovy

Groovy provides support for dynamically adding behaviour and properties to classes at runtime. To exemplify this, consider extending the built-in Java-class java.util.Date with a "getYearsSince()" method that returns the number of years since the date the object represents. In traditional Java, this can only be achieved by subclassing the java.util.Date class. In Groovy, however, this method can be added to the Date class at runtime with the following code:

```
java.util.Date.metaClass.getYearsSince = {
    return new Date().getYear() - delegate.getYear()
}

date = new Date(80, 1, 2)

println date.getYearsSince()

>> 28
```

This feature was especially helpful as it enabled the possibility to add custom functionality to instances of library classes with little effort. To illustrate the advantage of this feature, consider the Patient class in the OpenMRS API. The Patient class has its own compile-time methods, like getAge, getTribe, etc. The data model also has an Observation-class (named Obs), with a reference to the patient, made available through the method with signature `public Patient getPatient()`. There is however, no reference from the Patient class to observations linked with patients. The problem is: how do we implement a method, available on all patient-objects that give us the list of observations registered for them? The intuitive

approach to this could be by subclassing the Patient class with, say an ExtendedPatient class and implement a new method returning a list of observations. Something like:

```groovy
class ExtendedPatient extends Patient {
      public Collection<Obs> getObservations() {
            // Fetch and return observations for this patient
      }
}
```

Now, this would ensure that all "extended patients" (instances of the ExtendedPatient class) in the system has a method that returns a list of observations for the patient. But consider also the observations in the list that this method returns. These are instances of the Obs class and still got the function getPatient which still returns a Patient object, not an ExtendedPatient. Thus, the achievement of writing the ExtendedPatient class is really limited unless one subclasses and overwrites methods throughout the entire data model. With Groovy, the getObservation-method can be added runtime with the following piece of code:

```groovy
Patient.metaClass.getObservations = {
      // Fetch and return observations for this patient
}
```

Groovy will now ensure that all instances of the Patient class or subclasses of it have a getObservation method.

Groovy does not, however support statically typed return-type of dynamically added methods, something that disables introspection features on them. To cope with this, the Groovy class TypeExtensions was written to enable borrowing of methods from a statically typed Java-class while keeping track of return values of the borrowed methods (see Appendix F for the whole class).

```groovy
class TypeExtensions {
    private static HashMap<Class, Class> extenders = new HashMap<Class, Class>();

    /**
     * Adding methods of extender class to the extendee class
     * All instances of the extendee class will have methods of the extender class
     */
    public static extend(Class extendee, Class extender) {
        extenders.put(extendee, extender)
        for (Method m in extender.getDeclaredMethods()) {
            def method = m.name
            extendee.metaClass."$method" << {->
                extender."$method"(delegate)
```

```
            }
        }
    }
    /**
     * Returns a list of methods for extended class
     */
    public static List<Method> getAllMethods(Type clazz) {
        if (!clazz) return null
        if (extenders.containsKey(clazz))
            return Arrays.asList(clazz.getMethods())
                    .plus(Arrays.asList(extenders.get(clazz).getMethods()))
        else
            return Arrays.asList(clazz.getMethods())
    }
(...)
}
```

The `extend` method in the above Groovy class takes two classes as parameters. The first class is the one to be extended (the extendee) with the methods belonging to the class in the second parameter (the extender). When called, the extend-method first adds the extender class to a hash map (using the extendee as key) for later use. Then, the methods of the extender class are looped through, adding each method to the extendee class using its metaClass. The new method of the extendee is a code block calling the original method of the extender with itself as parameter. That's why all methods of extension classes must have the first parameter being of the class it intends to extend. For example, the method returning observations for a patient described earlier would be required to take a Patient object as parameter:

```
public static Collection<Obs> getObservations(Patient delegate) {
    // Fetch and return observations for the delegate patient
}
```

JSON as exchange format

JavaScript Object Notation (Crockford, 2006) is a lightweight data exchange format that is easily parsed by JavaScript. It was therefore very suitable as exchange format from the back-end to the front-end. The front-end sends request by regular get and post requests, while the back-end returns a JSON structured response to the request. Grails comes with a converter that easily transforms a Groovy object to a JSON object:

```
def employeesAsJSON = Employee.list() as JSON
println employeesAsJSON

>> [{"id":1,"class":"Employee","address":2,"age":28,"birthDate":new
```

37

```
Date(332231400000),"department":1,"firstName":"Jim","jobTitle":"Student","lastName"
:"Ojam"},]
```

Class introspection and generics

Type introspection capability is required from the underlying data model in order to enable reading of return type values for the different classes and properties in it.

Java is a reflective programming language, which means it offers the possibility to introspect on its own language constructs and obtain meta-information about classes, methods, return values and parameters. For instance, it is possible to determine what kind of class an object is an instance of:

```
static void printClassName(Object o) {
      System.out.println("The object is an instance of: "+o.getClass())
}
```

Invoking this method with a date object as parameter will produce:

```
printClassFor(new Date())
>> The object is an instance of: class java.util.Date
```

It is also possible to list all the methods available for an object by reading declared methods of the class the object is an instance of:

```
printMethodsFor(new Date())
>> public int java.util.Date.hashCode()
public int java.util.Date.compareTo(java.lang.Object)
public int java.util.Date.compareTo(java.util.Date)
public java.lang.Object java.util.Date.clone()
public boolean java.util.Date.equals(java.lang.Object)

public java.lang.String java.util.Date.toString()
(...)
```

This type of metadata is read from the data model by the back-end, and returned for use by the front-end, for example when determining what comparison operators to show when the user selects the age property of the patient (which actually refers to the getAge() method of the Patient class, as explained in the following section).

JavaBean naming pattern

The JavaBeans component architecture specifies a naming convention for classes and field accessors. A common convention is to expose private fields of a class through getters and setters, allowing the outside world to retrieve and manipulate the field. If we consider the field `private Date birthdate` in the class `Person`, the corresponding getters and setters for this field would be:

```
public String getBirthdate() {
    return this.birthDate;
}
public void setBirthdate(String bd) {
    this.birthDate = bd;
}
```

According to the JavaBeans specification, `birthdate` would be said to be a property of the `Person` class.

In Groovy, properties following the JavaBean naming convention are accessible using dot notation. For example, given an instance of the Person-class, the person's birthdate would be written as:

```
person.birthdate
```

This would implicitly invoke the getBirthdate -method (i.e. not by accessing the private "birthdate" field directly, which would violate the access restriction on the declared birthdate field).

The birthdate field is of the java.util.Date type which itself has its own getters and setters, like getYear and setYear. Correspondingly, the year of the person's birth date can be expressed as:

```
person.getBirthdate().getYear()
```

This is, no matter how readable for an experienced programmer, a less natural way of referring to a person's year of birth. In Groovy, the same thing would be written:

```
person.birthdate.year
```

In a more complex and comprehensive data model, like the OpenMRS, one can express fairly

readable references to properties along an object path. For instance, the birthdate of the person who created a form used in an encounter associated with an observation would be represented as:

```
observation.encounter.form.creator.birthdate
```

So, in short, the JavaBeans naming convention provides a more natural way of representing properties along paths in an object hierarchy.

This notation is used consistently in the front-end to provide a more readable representation of a condition, for example:

```
observation.encounter.form.creator.age is less than 18
```

Java generics

The introduction of reflection and generics in Java 1.5 opened for the possibility to "know" about the data type of items contained in collections. Prior to Java 1.5, element accessors of collection classes would only return instances of the built-in Object class. This class is the superclass of all objects in Java, so any object returned from a list would be an instance of the this. Thus, no further specification of the type objects contained in a collection was available prior to introduction of the reflection API.

For example, the generics features allows the siblings field in the Person class underneath to be declared as a list of other person-objects in the following way:

```java
private List<Person> siblings;
```

The corresponding get-method would look something like:

```java
public List<Person> getSiblings() {
        return siblings
}
```

Introspecting the getSiblings method would now determine that it returns a list of person-objects:

```java
Method method = Person.class.getMethod("getSiblings", null);
System.out.println(method.getGenericReturnType());
```

```
>> java.util.List<Person>
```

The information about the type of items in collections is used by the front-end to let the user set conditions on the items, in order to single out a desired item to use in the expression. For example using the where pseudo-operator:

```
patient.addresses.where(preferred is true).country equals "Namibia"
```

Here, the front-end will recognize the property addresses as a collection, and ask the back-end for what type of items it contains. The back-end will use introspection to read the enclosed type of this collection and answer that it contains objects of the Address class. The front-end will then go on by letting the user set conditions on the properties of the Address class (in this case, the country attribute refers to the address in the list where the preferred property is set to true).

Other Javascript libraries used

Several JavaScript libraries were used; The Prototype JavaScript framework was used to utilize object-oriented features in Javascript (which by default are rather limited), ease the creation of extended HTML components, and simplify DOM manipulation and Ajax requests. The script.aculo.us user interface library was used for visual features, while JQuery was used for the date selector widget and embedded dialog windows. Because different browsers have different XML APIs, the Sarissa library was used to ensure cross-browser compatible XML serialization.

4.3.2 Prototype implementation

Technical architecture
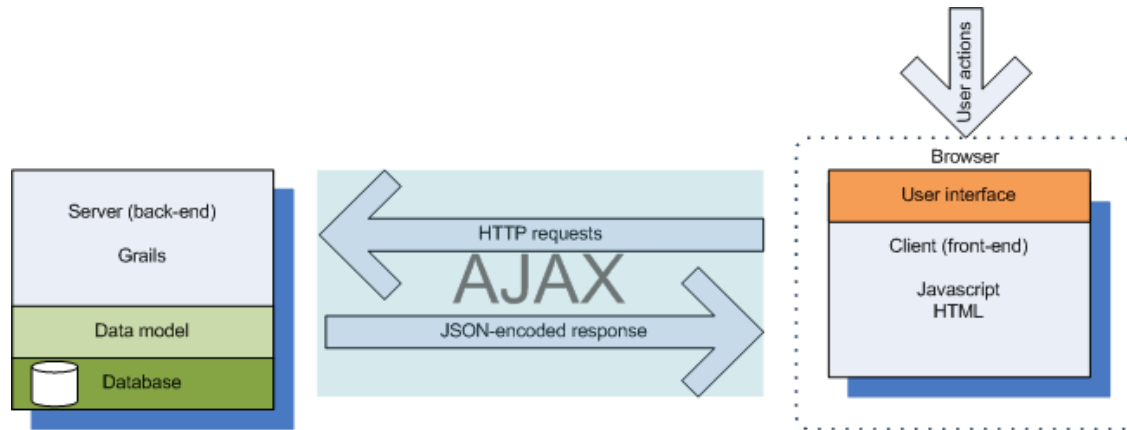
Figure 11: Architecture of the prototype

The prototype is made up of two separate, independent parts. On the server-side lays a Grails web application, which is responsible for interacting with the data-model by reading metadata about its classes and properties, and retrieving relevant data from the database. The communication between the two parts is done through Ajax requests. The front-end requests data from the back-end whenever needed by posting HTTP requests and receiving the appropriate data as a JSON-encoded response.

The back-end application can be deployed in several Java Servlet containers, like Tomcat, Geronimo, GlassFish, JBoss etc[8]. For the evaluation session, the prototype was deployed in Jetty[9].

The client side application is tested throughout the development stage using Firefox 3.0. It is briefly tested in Opera and Google Chrome without any clear incompatibility issues. However, the time constraint given this thesis did not justify thorough cross-browser compatibility checks.

---

[8] For a whole list of supported containers, please refer to http://docs.codehaus.org/display/GRAILS/Deployment

[9] http://www.mortbay.org/jetty/

42

Front-end

Expression structure

The front-end lets the user manipulate the different structural parts of an expression. To formally describe the syntax of a programming language, the Extended Bakus-Naur Form (EBNF) is widely adopted and used. The structure of an expression, as can be formulated and manipulated using the prototype can be described using EBNF grammar as:

```
expression        = condition [{(AND | OR) condition}].

condition         = single_condition | expression_group.

expression_group  = [ "not " ] "(" expression ")".

single_condition  = property {["." property]} comparison_operator value.

property          = propertyname ["["condition"]"].
```

This grammar can be visualized using the EBNF Visualizer application[10]. See Appendix I for a visual representation of the expression syntax.

The register

The register is a data structure used by the front-end to keep track of data model-specific features. The front-end totally de-coupled from the back-end and knows nothing about the underlying data structure or specific programming language constructs, this information is kept in the register. The prototype only knows that it must consult the register to obtain this information. The register supports three distinct language constructs: data types, comparison operators and value lookups.

Data types

Data types can easily be registered using the `Register.Datatype.add(options)` function, taking a list of options (JSON-style) as parameter.

A data type must always have an id. This should be a simplification of an actual data type in the programming language. For instance "java.lang.String" would typically just be given

---

[10] http://dotnet.jku.at/applications/Visualizer/

"string" as id. Data types can also have a list of aliases that would be which data types that should be generalized to it. Because the aliases are used to determine what comparison operators that are valid for each of them, the data types can be a quite broad simplification of those supported by the programming language. For example, instead of registering all numeric data types found in Java as separate, independent data types, they could be generalized into a "numeric" data type in the register:

```
Register.Datatype.add({
    id: 'numeric',
    aliases: [
        'java.lang.Integer',
        'java.lang.Double',
        'java.lang.Long',
        'java.lang.Float',
        'byte',
        'short',
        'int',
        'long',
        'float',
        'double'
    ]
})
```

The prototype will then treat all of the data types listed in aliases to be numeric, suggesting the same list of comparison operators to all.

Data model specific data types (i.e. classes) can also be registered using the same function. For instance, enabling support for instances of a customized Person class can be achieved by adding it to the register:

```
Register.Datatype.add({
    id: 'my.package.Person'
})
```

Comparison operators

Different data types have different comparison operators. These operators can be added to the register and will appear as options when the user encounters an associated data type. The data type association is specified when registering the comparison operator in the register. Consider the comparison operator "equals" (the "==" operator in groovy). It will be valid for a range of data types, including strings, numeric types, dates, etc. The equals operator can therefore be registered with the id of the associated operators:

44

```
Register.ComparisonOperator.add({
    id: 'equals',
    symbol: '==',
    display_name: 'is',
    datatypes: ['numeric', 'string', 'boolean', 'date']
})
```

Note that a comparison operator can also have a display name. This is what will be visible to the user in the GUI.

Logical operators

The only logical operators currently included are AND and OR. Support for other operators (i.e. exclusive or, XOR) can be enabled by adding them to the register:

```
Register.LogicalOperator.add({
    id: 'xor',
    symbol: '^',
    display_name: 'xor'
})
```

Lookups

The last, but not least important feature of the register is lookups. Lookups can be added to suggest different selection choices for the user. For instance, the list of available properties for a class is retrieved consulting the lookup register. A lookup can be either a hard-coded list of values, or an Ajax request to retrieve objects from the backend. For example, the lookup for boolean data types would be:

```
Register.Lookup.add({
    id: 'boolean',
    datatypes: ['boolean'],
    getItems: function() {
        return [true, false]
    }
})
```

A lookup that would retrieve a list of users from the backend would be:

```
Register.Lookup.add({
    type: 'request',
    id: 'persons',
    datatypes: ['my.package.Person'],

    request: function(args) {
```

```
        var url = 'backend/get_persons?' +
                  'match='+encodeURI(args.match)+
                  '&datatype='+encodeURI(args.datatype)

        new Ajax.Request(url, {
            evalJSON: true,
            onComplete: function(response) {
                var json = response.responseJSON;
                this.respond(json)
            }.bind(this)
        })
    }
})
```

Lookups registered with the option "type: request" will need to define a responder callback function, which is invoked when the Ajax request is done loading the response.

It is only in the lookup segment of the register that the prototype keeps the information about the back-end. In essence, the front-end and the back-end are totally de-coupled - neither knowing anything about the other. The front-end only knows it must get the information from *somewhere*, and therefore it consults the lookup register to find out from where. It is only the implementation of a lookup that keeps a reference to the back-end. The back-end, on the other side knows that it serves a front-end, and therefore it must conform to the standards required by it. In the prototype, the front-end and the back-end are on the same server, therefore the URLs defined in the lookups are relative.

The classreader lookup

The only lookup required by the front-end is the classreader lookup. The implementation of this must return a list of PropertyNodes corresponding to the given arguments its request-function is called with. It does so by consulting the back-end. The current implementation of this lookup is reproduced in Appendix E.

Expression nodes
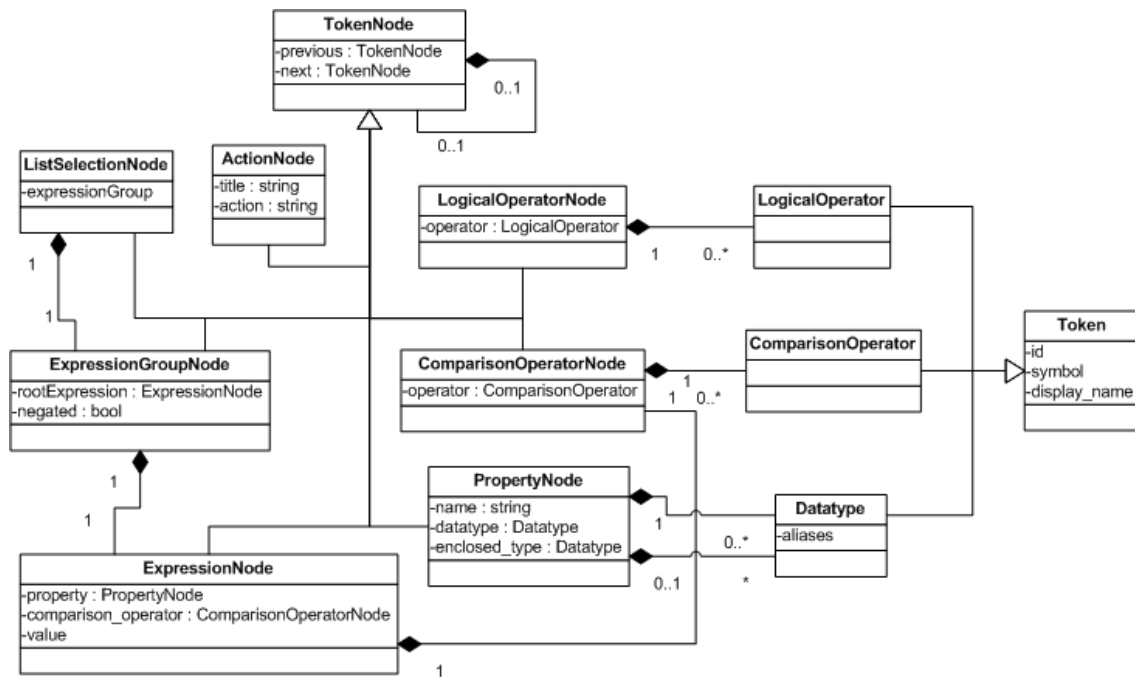
Figure 12: The different language constructs represented as TokenNodes

The different parts of an expression as described above, is represented as subclasses of the TokenNode JavaScript class in the file ExpressionNodes.js.

The TokenNode super class is implemented as a linked list, allowing each node to have other nodes linked either before or after. For example, a PropertyNode can be followed by another property node, and a condition can be followed by a logical operator, which in turn is followed by another condition. A description of the subclasses of the TokenNode follows.

ExpressionGroupNode

This holds a reference to an ExpressionNode instance, and a boolean indicating whether the group is negated or not.

ExpressionNode

This class has three fields: property, comparison_operator and value. The property field is an instance of PropertyNode, the comparison_operator is any registered comparison operator while the value can be literally anything.

PropertyNode

Nodes of this type contains a reference to the property name, the data type returned by this property and, in cases where the property returns a collection of an enclosed data type, objects of this class will have a reference to the enclosed data type.

ListSelectionNode

A node of this type contains an expression to select items from a list. ListSelectionNodes will always succeed a PropertyNode with a collection data type, and the condition will be based upon the enclosed type of the preceeding PropertyNode. So if the preceding PropertyNode has a collection of addresses, the ListSelectionNode will have a condition based on the properties of the address class.

LogicalOperatorNode

The LogicalOperatorNode contains an instance of a LogicalOperator and is always preceded and followed by either an ExpressionNode or an ExpressionGroupNode

ActionNode

An ActionNode is used to invoke certain actions supported by the front-end. The ActionNode is implemented in order to let the back-end decide when to invoke different front-end supported actions.

UI components

The Combobox control

The Combobox control is the most re-used UI component throughout the system. It is a flexible, rich, dynamic user interface control that allows for user typing in addition to selecting items from a list of possible choices. Several of the UI components described below acts as proxies for a Combobox instance by redistributing commands to, and listening for changes in it. The Combobox contains an instance of a ComboboxList which consists of ComboboxItem instances. A ComboboxItem can have a tooltip text that will be displayed when the user points the mouse over it (Figure 13).
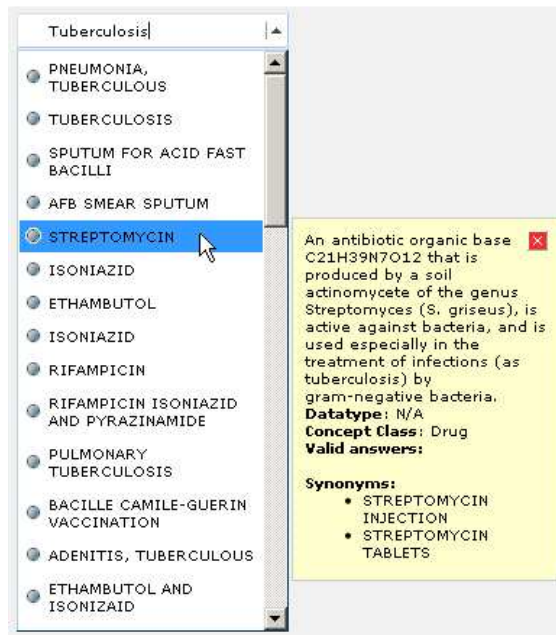
Figure 13: The Combobox UI Control showing tooltip hints for items matching "Tuberculosis"

Each of the expression nodes mentioned in the previous section have their own dedicated UI component (Figure 14). For example, the LogicalOperatorNode has a LogicalOperatorControl which is responsible for receiving user actions and manipulating the LogicalOperatorNode accordingly. All UI components have a getModel-method, which returns an instance of the associated token node. The different UI components are described below. As the TokenNode superclass, the Control superclass is implemented as a linked list, having references to next and previous controls.
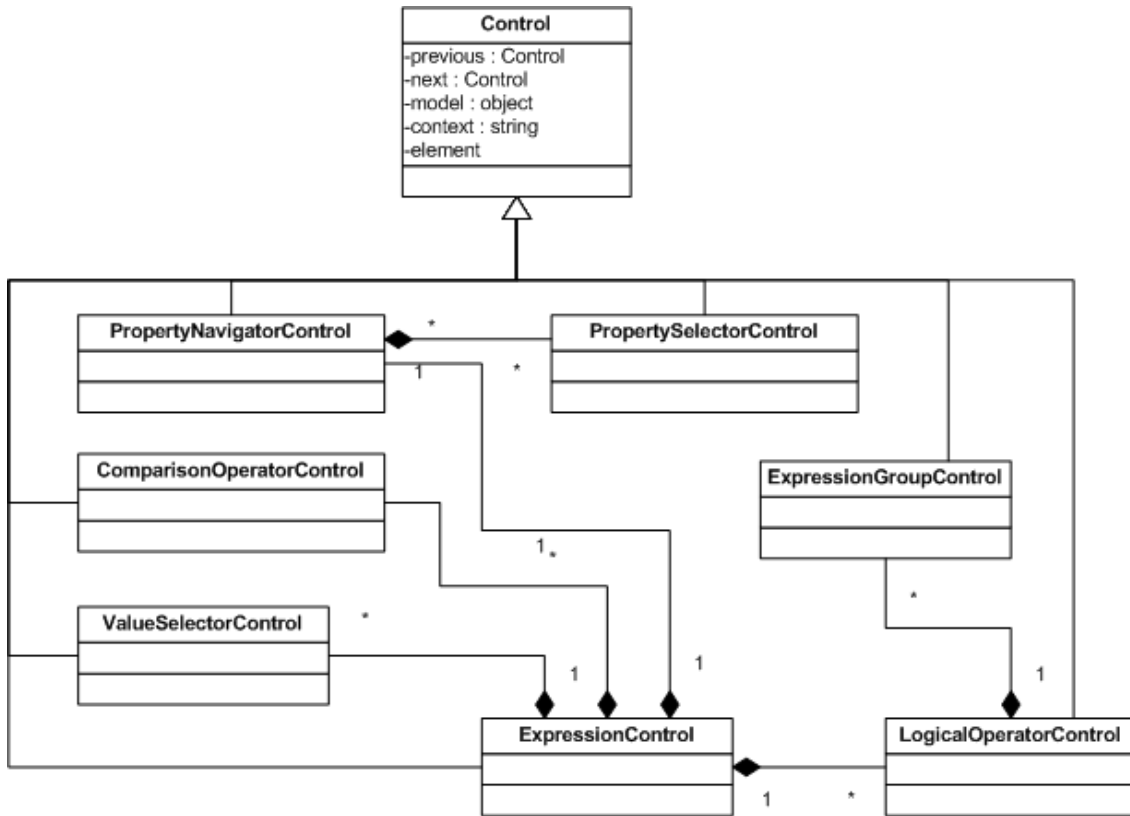
Figure 14: Classdiagram of UI components

ExpressionGroupControl

This creates an UI control that deal with an expression group. It is responsible for listening to user actions and adding new expressions at different points in the hierarchy. An instance of this control keeps a reference to the first of the inner ExpressionControl instances, which in turn, has a reference to the next, and so on.

ExpressionControl

This control is responsible for a single expression (condition). It contains a PropertyNavigatorControl, a ComparisonOperatorControl and a ValueSelectorControl. It is responsible for responding to events in these controls, and propagating events upwards in the control hierarchy.

PropertyNavigatorControl

The PropertyNavigatorControl lets the user navigate and select properties of the current data model. It creates a control for every property along the property path, and fires adequate

50

events based on user actions.

PropertySelectorControl

This control is responsible for administering the ComboBox for each of the different properties. Whenever the user changes the value of a PropertySelectorControl, an event is fired, informing the parent controls about the details of the change in this property.

ComparisonOperatorControl

The ComparisonOperatorControl is populated with the valid comparison operators for the last property in the current property path. If for instance the property path is:

```
employee.address.zipcode
```

The ComparisonOperatorControl would contain comparison operators associated with numeric values. If the last property is of a complex type, i.e. of the Address type, a ComparisonOperatorControl will also contain the properties of this type. Selecting one of these properties will cause the control to fire an event, telling the PropertyNavigatorControl to add the selected property to the end of the current property path.

ValueSelectorControl

This control displays an input-field for the user to type the value to be compared. If a lookup is registered for the current data type or property path, the ValueSelectorControl will retrieve the list items from the lookup and display them as a list of possible alternatives. The ValueSelectorControl can also use a widget for selecting special value-types, for example a date.

XML serializer

To convert the expression from the internal representation of TokenNodes to an XML representation of the expression, the XMLSerializer class was written.

In order to serialize an expression, a new instance of the XMLSerializer would be created with the root-expression as constructor parameter:

```
var serializer = new XMLExpressionSerializer(new ExpressionGroupNode(), datamodel)
```

Then, the serialize-function of the XMLSerializer instance will return an XML document object, which in turn can be serialized to a string using the Sarissa XML Serializer.

```
var serializer = new XMLExpressionSerializer(app.getModel(), datamodel)
```

The XMLExpressionSerializer constructor takes the root-expression as parameter and has the two functions serialize() and deserialize() which returns an XML document object and an xml string, respectively. See Appendix J for an example of an XML serialized expression.

Back-end

An essential part of the back-end implementation is the return-type introspection on property get-methods. Introspection features are used when the front-end asks the back-end for a list of properties belonging to a given class. Introspection features in the back-end are implemented in the Groovy-class TypeExtensions. More specifically, the static method getEndClass in the TypeExtensions class will return the return type of the method corresponding to the last property in a property path (a string of dot-separated properties, i.e. "employee.address.zip"). This is an example invocation of the getEndClass-method:

```
// What type does the getName-method of the Patient-class return?
System.out.println(TypeExtensions.getEndClass(Patient.class, "tribe"));
 >> class org.openmrs.Tribe

// The getAddresses-method of the Patient class returns a java.util.Set of
// org.openmrs.PersonAddress instances
System.out.println(TypeExtensions.getEndClass(Patient.class, "addresses"));
>> java.util.Set<org.openmrs.PersonAddress>

// What type would be returned if calling
// emp.getDepartment().getAddress().getZip()
// on the Employee-object emp
System.out.println(TypeExtensions.getEndClass(Employee.class, "department.
location"));
>> class no.uib.bna049.example.Address

// Projecting across a list
System.out.println(TypeExtensions.getEndClass(Employee.class,
"projects.find({it.name.contains('database')}).department.location.zip"));
>> int
```

The getAllMethods method (see Appendix G) returns a list of all methods declared for a given class:

```
System.out.println(TypeExtensions.getAllMethods(Address.class));
>>[(...) public void no.uib.bna049.example.Address.setCountry(java.lang.String),
public java.lang.String no.uib.bna049.example.Address.getStreet(), public void
no.uib.bna049.example.Address.setStreet(java.lang.String), public java.lang.String
no.uib.bna049.example.Address.getCity(), public void
no.uib.bna049.example.Address.setCity(java.lang.String), public int
no.uib.bna049.example.Address.getZip(), public void
no.uib.bna049.example.Address.setZip(int), public native int
java.lang.Object.hashCode(), public final native java.lang.Class
java.lang.Object.getClass(), (...)]
```

Combined, these two methods can give us a list of all methods at the end of a property path:

```
Type endclass = TypeExtensions.getEndClass(Employee.class, "department.location");
List<Method> methods = TypeExtensions.getAllMethods(endclass);
System.out.println(methods);
>>[(...) public void no.uib.bna049.example.Address.setCountry(java.lang.String),
public java.lang.String no.uib.bna049.example.Address.getStreet(), public void
no.uib.bna049.example.Address.setStreet(java.lang.String), public java.lang.String
no.uib.bna049.example.Address.getCity(), public void
no.uib.bna049.example.Address.setCity(java.lang.String), public int
no.uib.bna049.example.Address.getZip(), public void
no.uib.bna049.example.Address.setZip(int), public native int
java.lang.Object.hashCode(), public final native java.lang.Class
java.lang.Object.getClass(), (...)]
```

Please refer to Appendix H for the full implementation of the getEndClass method.

The BackendController class

The back-end is written as a Grails controller. A Grails controller-class exposes its defined methods as an URL. To illustrate, consider the controller-method "sayhello" in the Groovy controller-class HelloWorldController:

```
class HelloWorldController {
    def sayhello = {
        render "Hello World!"
    }
}
```

This method is available at the relative server url: `helloWorld/sayhello` and will display "Hello World!" in the browser window when accessed.

The front-end communicates with different controller-methods according to the lookups registered in the front-end. The only one critical and required controller-method is the one delivering the list of properties to the front-end. This method takes two parameters – one is the property path and the other is the starting-point class (or context). The controller-method is then required to return a JSON object with a list of read-methods of the end class and their

53

return type.

Delivering other types of data to the front-end

Several lookups can be defined for the front-end, triggered by a certain path, data type, property name or combination of these. For instance, the gender of a person is often represented by a "gender" property in a Person-class. The format on the value of this property may vary from "M, F" to "Male, Female", "1/0" etc... This format may be unknown to the user, and not even standardized in the data model. Therefore it might be useful to query the database for a list of all the possible values of the gender property. To achieve this, the lookup is added to the register with a matching property name and data type and an implementation of the request-function.

The request-function calls an URL, invoking a back-end controller that looks in the database for a list of distinct values for the gender field, and then returns it.

On the client-side (JavaScript):

```javascript
Register.Lookup.add({
    id: 'gender',
    property: /.*gender/,
    datatypes: ['string'],
    request: function() {
        new Ajax.Request('backend/json_genders', {
            evalJSON: true,
            onComplete: function(response) {
                var json = response.responseJSON
                this.respond(json)
            }.bind(this)
        })
    }
});
```

On the server-side (Groovy controller):

```groovy
def json_genders = {
    response.setHeader('Content-type', 'application/json;charset=UTF-8')
    render Person.executeQuery("select distinct p.gender from Person p")
}
```

4.3.3 Integration with the OpenMRS API

Integrating the OpenMRS API with Grails was straight-forward. The OpenMRS API jar file

is available for download from openmrs.org, and has built-in object-relational mapping through embedded Spring and Hibernate configuration files.

After the OpenMRS API jar file was downloaded, it was copied to the `lib` subfolder of the Grails project folder.

The content of the Spring configuration file `applicationContext-service.xml` was then copied from the OpenMRS JAR into the `grails-app/conf/spring/resources.xml` file. The database schema was then created and populated with demo data using sql files available for download from openmrs.org. Finally, entering the database login details in the `grails-app/conf/DataSource.groovy` file was all the configuration work needed to do in order to integrate OpenMRS in the Grails application.

5 EVALUATION

This chapter is divided in two parts. The first part begins by presenting the characteristics and purpose of evaluation, followed by a description of the evaluation techniques and design utilized in this thesis. The second part presents the findings, and a general discussion of their impact.

5.1 What is an evaluation and why do we do it?

The purpose of an evaluation is to assess the effects and effectiveness of something, typically some innovation, intervention, policy, practice or service (Robson, 2002, p. 202). Evaluation can be classified as either being formative or summative. This classification is often used in educational sciences where the formative evaluation is used by an instructional designer to improve a curriculum or educational programme, while the summative evaluation is at a later stage used to assess the outcome of the change in curriculum. The distinction is well illustrated in Robert Stakes notable quote "When the cook tastes the soup, that's formative; when the guests taste the soup, that's summative" (quoted in Scriven, 1991, p. 169). Formative evaluation is intended to help in the *development* of the programme, innovation or whatever is the focus of the evaluation, while a summative evaluation "concentrates on assessing the effects and effectiveness of the programme" (Robson, 2002, p. 206).

5.2 Usability evaluation

Different methods exist for evaluating the usability of a computer system. In a study by Tullis and Stetson several different methods for assessing website usability were compared (Tullis & Stetson, 2004). This study suggests that the questionnaire yielding most "correct" answers at the lowest sample size was the Systems Usability Scale (SUS). At a sample size of only 8, SUS gives an accuracy rate at about 75%, while the others stay as low as 40-45%. The SUS is a freely available, widely tested and easy to apply questionnaire used to assess the degree of usability offered by a computer system.

% of "Correct" Conclusions
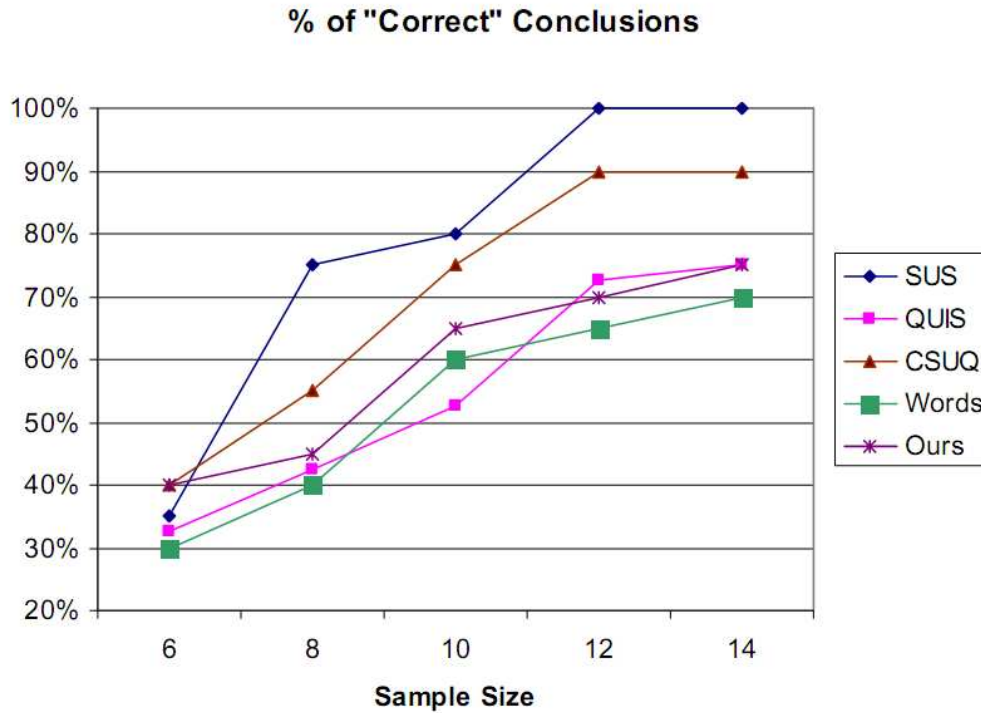
Figure 15: The systems usability scale compared to other available usability measurement scales (Tullis & Stetson, 2004)

SUS is developed as a global assessment scale that can be used to compare usability across different contexts. Brooke argues that there is no objective way of measuring usability. For example, the ISO 9241-11 standard suggests that usability should be evaluated in terms of effectiveness, but "effectiveness are very obviously determined by the types of task that are carried out with the system" (Brooke, 1996). Therefore, he claims that comparing two different systems is a matter of "comparing apples and oranges", and that the only possible way to do comparable usability assessments across different systems is by using a subjective assessment scale (Ibid.). What is measured is how a group of users perceive and judge the system in use rather than how the system performs at any given task or property.

5.3 Evaluation design

The evaluation of the prototype was performed in two phases. Phase one used a formative approach with a desire to find areas of improvement and unveil possible errors. The feedback received from this phase was used to guide further improvement of the system. The second phase had a summative nature, as the goal was to measure the outcome of the first phase.

57

Although a subordinate goal of this phase was to find areas of improvement, its major aim was to answer the question "how well does it work?" (in culinary terms, "how good does the soup taste?"). The formative evaluation that followed the development was already described in chapter 4, so only the summative evaluation will be presented in this chapter.

Apart from the evaluation conducted at time of development, the final prototype was evaluated by four users with both domain knowledge and technical understanding. These evaluations focused on how the users appreciate the proposed prototype, both in terms of usability (i.e. ease of use), and its applicability (i.e. its ability to solve the actual problem to which it is proposed as a solution).

Revisiting the description of SUS above, a remark on the type of system is needed. Tullis & Stetson (2004) evaluated different methods for assessing *website* usability. The Eligibility Criteria Builder however has more characteristics of an application and to a less extent website in the traditional sense. Accordingly, the term "website" may not be an accurate description of it (a more precise term will be "web-application"). However, all but one of the original methods in this study was originally developed for websites. They served as a means to evaluate computer systems in general. They were later adapted by the authors to the context of websites (Tullis & Stetson, 2004). The SUS offers a ready-to-use questionnaire form in which the user gives an answer according to their degree of agreement to a list of statements.

5.3.1 Questionnaire

Each evaluator was asked to respond to a questionnaire that accompanied the evaluation e-mail (see Appendix A). This questionnaire consisted of two parts. Part I was the SUS questionnaire and part II a semi-structured questionnaire with questions about previous experience with similar systems, and general judgements and comments about the system.

For part I, the SUS scale, the questions asked were:

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system

7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome/awkward to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

The questions asked in the second part were:

1. Experience with other systems

    a) I know of similar systems (Yes/No)
    b) I have used similar system(s) before (Yes/No) *If answer is No, go to d)*
    c) I have used these systems before:
    d) I will summarize the major difference between the system(s) I have previously used and this as:

2. Technology

    a) I would like to use a system like this in my own practice (Yes/No)
    b) In what ways, if any, could this system improve your current practice?
    c) What potential advantages, if any, could the use of a system like this have?
    d) What potential disadvantages, if any, could the use of a system like this have?
    e) How can this system be improved in terms of functionality and user friendliness?
    f) Were you able to formulate any tasks from your own practice? If not, what was the problem?
    g) General comments, thoughts, etc.

5.3.2 Evaluation website

For convenience, an evaluation website was set up and made accessible for the evaluators (Figure 16). The site had a login screen, where each evaluator had to type in a pre-assigned participant id. When authenticated, they got the choice between solving the practice tasks, doing the actual test-tasks as specified in the evaluation guidelines or restarting the session.
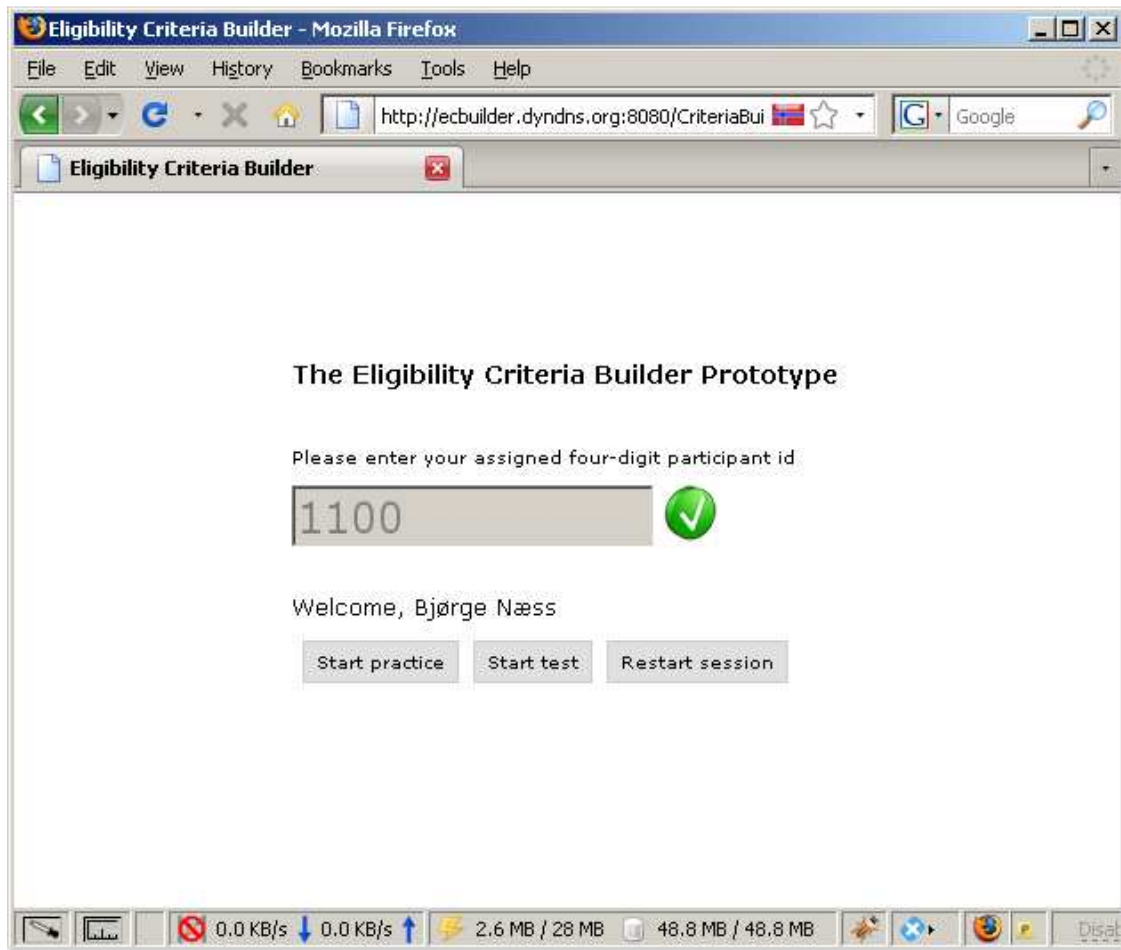
Figure 16: The Evaluation site, welcome screen after authentication.

When clicking either the "Start practice" or "Start test" buttons, a new window opens with the prototype. To make it easier for the evaluators, the task description were integrated in the prototype and would appear when they chose the corresponding task from two added dropdown fields (Figure 17).

Figure 17: The evaluator selects which task number he/she is about to solve

5.3.3 Evaluators

Four evaluators were recruited for the evaluation of the prototype. Ideally, it would have been at least twice as many, but it turned out to be quite a challenge recruiting the right people. Several relevant mailing lists were inquired, but with rather modest response. Fortunately, we managed to get four skilled evaluators with both domain knowledge and technical insight. Because the evaluators were situated in different parts of the world (India, Pakistan, Norway), communication were done using e-mail.

After agreeing to participate, the evaluators were sent an e-mail with instructions on how to access the evaluation site, perform the evaluation tasks and fill in the attached questionnaire (See Appendix A for the whole e-mail).

5.3.4 Practice tasks

To make sure the evaluators become familiar with the system and how it works, two test-

tasks were prepared beforehand. These tasks were based on a simple data-model consisting of three classes: Employee, Department and Address (see Figure 18). This data-model was far less complex compared to the OpenMRS model and it was intended to give the evaluators an understanding of how the prototype works through learning-by-doing.
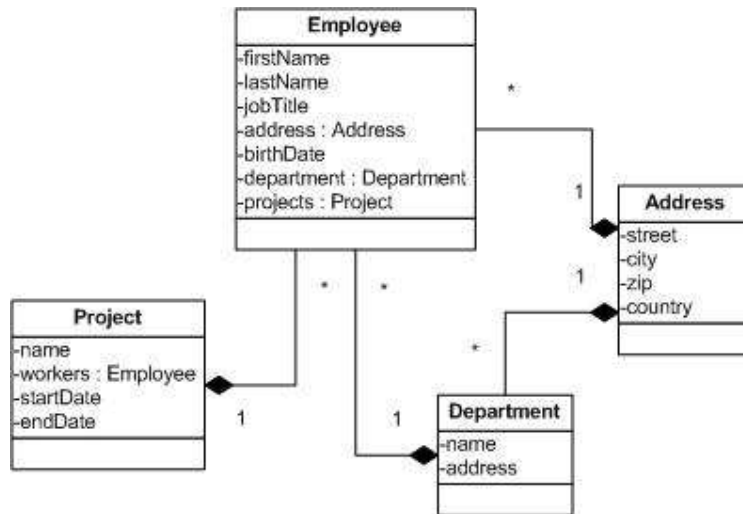
Figure 18: The data model used in training

Practice task 1

Inclusion criteria:

    1.1. Working at the Department of Social Anthropology
    1.2. Age is above 24 years
    1.3. Address is somewhere in Norway
    1.4. Job title does not contain "managing".

Exclusion criteria:

    1.5. 50 years or younger
    1.6. Working at department of Information and Media Sciences
    1.7. Living outside Norway

Practice task 2

Inclusion criteria:

    2.1 Born before fifteenth of May 1976

2.2 Working at a department that is located either in Tanzania or in Uganda, but not in the big cities of Dar es Salaam or Kampala.

2.3 Working at a Department whose name contains Faculty

Exclusion criteria:

2.4 Current address is outside Tanzania or Uganda

2.5 Working on a project ended within the last year with "water" in the project name

5.3.5 Evaluation tasks

For the evaluation session, the evaluators were given three main tasks and two additional tasks. The three main tasks had varying degree of complexity ranging from simple to complex. For the additional tasks the user was asked to edit a previously saved task, and to think of any criteria that he/she remembered from his/her own practice and try to solve it using the system. To give the evaluation a feature of authenticity, all tasks were authentic eligibility criteria taken from real-life protocols[11] retrieved from clinicaltrials.gov. The main assignment was to build an expression for each of the criterion in the evaluation tasks described below.

The evaluation tasks are deliberately chosen on the basis of their level of complexity. Choosing only simple tasks would most likely have produced results that would make it easy to conclude that the system is well-suited for the given purpose and easy to use. It was a goal, however, to identify the upper limits of the system by exposing it to what can be called an *acid test*[12]. The evaluation targeted on identifying the systems capabilities - what tasks are possible to formulate with the system, and what tasks are too complex to formulate it (i.e. to find out "where the shoe pinches").

Evaluation task 1

Inclusion Criteria:

---

[11] See Appendix C for a reference to the study prototocols the criteria is taken from

[12] Originally a notion of determining whether a given metal is gold

1.1. Males or females aged greater than or equal to 1 to less than 4 years

1.2. Known residents of the village of Bancoumana, Mali or its surrounding area

1.3. Identified as having a malaria infection by blood film examination

Exclusion Criteria:

1.4. Convulsions or history of convulsions

1.5. Known hypersensitivity or allergy to artemisinin derivatives or mefloquine or mefloquine chemically related compounds (for example quinine and quinidine)

1.6. Presence of any known serious chronic disease (e.g. AIDS, sickle cell disease, malignancy)

Evaluation task 2

Inclusion Criteria:

2.1 Born before fifteenth of May 1976

2.2 Age > 1 year

2.3 Axillary temperature $\geq$ 37.5°C and/or history of fever in the previous 48 hours without any other evident cause

2.4 Unmixed infection with P. falciparum of between 250 and 100,000 asexual parasites/mm3 as determined by microscopic exam of the thick or thin smear

2.5 An informed consent obtained from the patient or his/her guardian (in case of patients $\leq$ 18 years old) and assent for children (8-18 years old)

Exclusion Criteria:

2.6 Other severe chronic diseases (e.g., cardiologic, renal, or hepatic diseases; HIV/AIDS; severe malnutrition)

2.7 History of allergy to mefloquine, artesunate, quinine, tetracycline, or clindamycin

2.8 Pregnancy (based on urine test), since this group of patients receives other drugs for malaria treatment in accordance with Peruvian national guidelines.

Evaluation task 3

Inclusion Criteria:

3.1. Aged 1 to 5 years

3.2. HIV infection (previously confirmed by 2 ELISAs for children > 18 months; DNA PCR for those < 18 months)

3.3. Informed consent from the parent/caretaker

3.4. Ability to return for follow-up (lives within a radius of 15 km from hospital and unlikely to change residence during the course of the study)

Exclusion Criteria:

3.5. Children already enrolled in other studies

3.6. Children with severe abnormalities which are likely to impair oral intake (for example, severe cerebral palsy)

3.7. Severely ill children requiring urgent admission and resuscitation

Additional tasks

Load the first test task's Inclusion criteria for editing and change it so that only females are included in this study. Save the expression again when you are done.

Can you think out some tasks that you experienced in your practice and express it using the system? If you do, you can save it as "My task #". If you encounter difficulties or were unable to accomplish it, please write a short note about it in the attached questionnaire.

5.3.6 Data collection

Questionnaire response and a log of the solutions to the tasks by the evaluators constitute the data that was collected in this thesis. In addition, as one of the evaluators was located in Bergen, this evaluator was observed while performing the evaluation tasks. The notes taken during this observation session are discussed at the end of this chapter. It would also have been interesting to verify the performance and validity of the criteria formulated by the evaluators by running them against a real-world clinical trial database and comparing the returned patients with the ones that was actually judged (by clinicians) as eligible for the trial. This was however not possible as no such database was available. Nevertheless, in order to be able to analyze the answered solutions, a collection of example (reference) solutions was made beforehand. Comparing these solutions to the solutions given by each evaluator provided information about the extent to which the evaluators used the system the way they were expected to.

5.4 Findings

In this section, the collected data are presented and analyzed. First, response to the questionnaire is presented, followed by a summary and analysis of the tasks as they were solved by the evaluators. Finally an analysis of the field notes taken in the observation session is further elaborated.

5.4.1 Response to questionnaires

Part I: SUS

In the SUS part of the questionnaire, each question was given a score between 1 and 5 where 1 was "strongly disagree" (SD) and 5 were "strongly agree" (SA). A score of 2 is further classified as "disagree", a score of 4 is "agree" and a score of 3 is "neutral". The responses given by the test-users are presented in Table 1. The numbers to the right of each question in represent the number of evaluators that answered by putting the mark in this column. See Appendix B for the raw response data.

Table 1. *How the Participants Responded to the Systems Usability Scale*[13]

Part I: SUS	SD				SA
	1	2	3	4	5
1 I think that I would like to use this system frequently				1	2
2 I found the system unnecessarily complex		2		2	
3 I thought the system was easy to use		2		1	1
4 I think that I would need the support of a technical person to be able to use this system		4			
5 I found the various functions in this system were well integrated				2	2
6 I thought there was too much inconsistency in this system	2	1		1	
7 I would imagine that most people would learn to use this system very quickly		1	2	1	
8 I found the system very cumbersome/awkward to use		3		1	
9 I felt very confident using the system			2	2	
10 I needed to learn a lot of things before I could get going with this system			2	2	

As seen in Table 1., all the evaluators that answered question one (all except one, who left this question blank) agreed to the claim that they would like to use the system frequently. Two answered that they strongly agreed, while the last evaluator gave this claim a score of 4 (agree).

Two of the evaluators found the system too complex to use, while the other two disagreed to

---

[13] Please note that one evaluator did not answer question 1

this claim. The reason for this disagreement may be due to differences in knowledge about the OpenMRS data model.

One evaluator strongly agreed to the claim that the system was easy to use, one agreed and two others disagreed to this. All disagreed to the claim that they would need help from a technical person to use the system, but at the same time two of these evaluators also disagreed to the claim that the system was easy to use. Hence, their appreciation of the system may be summarized as "it is easy to understand how it works, but it is not so easy to use it".

All evaluators either agree or strongly agree to the statement that the various functions of the system were well integrated. While one of the evaluators agreed to the statement that there was too much inconsistency in the system, the three others either disagreed or strongly disagreed to this claim.

To the claim that most people would learn to use the system very quickly, the views differed. One agreed to this, another one disagreed while the two others answered neutral to this. In this question, the evaluators were actually asked about how quick they think *other* users will learn how to use the system. It is clear, however, that with the fairly short explanation of how the system works and the usage example attached in the evaluation e-mail, it is evident that all of the evaluators managed to learn the basic usage (by solving at least some of the tasks properly). Whether most other people would learn to use the tool as quickly as the evaluators, is, however, remains an open question.

Three evaluators strongly disagreed that they found the system very cumbersome/awkward to use, while the last one agreed to this claim. Two evaluators agreed that they felt confident using the system, while the other two gave neutral answers to this claim. Also, two evaluators agreed that they needed to learn a lot of things before using the system, while the other two answered neutral. The different answers to these questions may be attributed either to different levels knowledge of, and experience with the OpenMRS data model, or to different levels of programming experience by the evaluators.

Part II: Experience with other systems & technology

To the questions about experience with similar systems, only one of the participants answered that they had knowledge of a similar system, and named the OpenMRS cohort builder as one

of which he had previous experience. When asked to compare the previously used system with the Eligibility Criteria Builder, this evaluator stated that:

"This system can search for very, very specific information; at a far greater depth than Cohort Builder." [Evaluator 2]

All but one of the participants answered that they would like to use a system like this in their own practice. One evaluator left this question unanswered. When asked whether and how the Eligibility Criteria Builder could improve their current practice, the views differed. One of the participants viewed the prototype as a handy query tool that could speed up the process:

It could allow clinicians in our organization to quickly find patients matching very specific criteria. This of course could expedite the entire process of analysis. [Evaluator 2]

Another participant envisioned other uses for a tool like this, not only limited to selecting eligible participants to a clinical trial.

It would be useful if the system could import a user specified dictionary so that it could adapted for developing Inclusion & Exclusion criteria for any type of selection procedure, not just in medicine. [Evaluator 3]

A viewpoint by another evaluator is that just having computerized eligibility criteria represents a great improvement alone:

"By forcing the eligibility criteria to be defined so that they can also be computerized." [Evaluator 1]

To the question about the potential advantages, if any, of the use of a system like the Eligibility Criteria Builder, the first evaluator judges the tool as something that can ensure adherence to the protocol:

"It ensures that the protocol is adhered to" [Evaluator 1]

The second evaluator re-emphasized the statement from the previous answer, and upheld the view of the prototype as a query tool:

"Again, allowing one to find highly specific information quickly. " [Evaluator 2]

The third evaluator sees it as a tool to develop selection criteria for all types of surveys that

require a population sample:

"To quickly develop selection criteria for all sorts of sample surveys" [Evaluator 3]

The general purpose of the prototype is also clearly understood by the last evaluator:

This system could help in ensuring that data collected conforms to predetermined conditions thereby reducing errors. [Evaluator 4]

When asked about the potential disadvantages the use of a system like the Eligibility Criteria Builder could have, the second evaluator indicated that one of the earlier mentioned strengths of the system could also be a weakness because of the domain knowledge it requires from the user:

"Because it can be so specific, only people with specialized knowledge in that field could operate the system." [Evaluator 2]

Another evaluator suggests that the user should be able to add domain-specific dictionaries to the system, through the user interface.

Domain-specific dictionaries should be possible to add to the software by the user. [Evaluator 3]

This is clearly an interesting point, but also relies heavily on the vocabulary structure implemented by the underlying data model. An interface should ensure that whatever domain-specific terms that will be added by the user at the criteria definition stage will be the same terms later used at the data entry stage. It is also worth noting that the infrastructure for knowledge modelling may already be available in the existing software.

Another response to this question by the same evaluator could be classified as a missing feature rather than a potential disadvantage of using a system like this:

This system does not point out errors in the expressions. [Evaluator 3]

It is nevertheless an important comment, and this missing feature should receive attention in any future version.

Another potential disadvantage of using a system like this is mentioned by another evaluator:

"It may lose some of the flexibility and medical based decisions." [Evaluator 1]

This is a problem if a tool like the Eligibility Criteria Builder would be used uncritically to automatically enrol patients into a clinical trial. The tool should however not be used this way. It should rather suggest to the user a list of potentially eligible patients recommended for participation. Each patient should ultimately be approved based on a qualified judgement by a clinician.

The fourth evaluator answers this question from a usability perspective pointing to an inherently restrictiveness and inflexibility in the system (without any further elaboration).

It might be too restrictive and inflexible for the end user. [Evaluator 4]

Each evaluator was also asked about how the system could be improved in terms of functionality and user friendliness. To this question, one evaluator answered:

It would help if the system permitted me to edit the Code for the expressions directly as that is sometimes required for better control. [Evaluator 3]

This is to be considered an advanced feature for expert users with programming skills. Because the user interface was designed to enable non-programmers to define the selection criteria, the possibility of "programming" the criteria code was not prioritized. If there would be any future development of the prototype, this should however be considered as an interesting feature as it will enable extremely powerful expression formulation. This evaluator also adds another interesting remark:

Errors in expressions should be highlighted so that they can be easily corrected. [Evaluator 3]

The user interface of the Eligibility Criteria Builder is designed to prevent (or at least limit) syntactical errors from occurring as it forces the different parts of the expression to be selected from a set of valid choices. Nevertheless, the system still allows for errors to be typed, so this evaluators point is important. It would be a critical requirement provided the code edit feature described above was to be implemented. Errors in the formulated criteria can potentially have an enormous impact on both the safety of the involved patients and the outcome of the study.

Another improvement was suggested by another evaluator:

When setting conditions (ex: where [concept is CDC CATEGORY C]) another window opens, but when setting conditions in there, yet another window can open, this time right on top of the first, making it hard to distinguish between the two. This should be resolved. [Evaluator 2]

This is a very interesting remark that also has an easy technical solution. As an effect of this issue, rather than get the impression of a new window opening, the user might just as well perceive it as the fields in the current window gets blanked out. A workaround for this issue would be to adjust the position of newly opened window in the bottom-right direction. Thus, the background window would appear more clearly as layered behind the newly opened one.

This evaluator has another suggestion for improvement:

Also, I would like to be able to search for terms within concept descriptions (ex: selecting a concept that contains the word "chronic" in the description). [Evaluator 2]

This feature is actually supported. For example, when searching for "chronic", the system returns the concept "Asthma", which has "chronic" among the words in the description but not in the title. This is a feature that is supported by the backend implementation, as it is the one that receives the search term from the user interface, processes it and decides what elements to return. The fact that the evaluator did not realize this may however indicate that this feature should be clearer to the user.

Here is another suggested improvement:

In this setup, the major limiting factor is the fact that it uses the OpenMRS API, which to people not familiar with it, is complex. But a person knowing OpenMRS would very quickly be able to complete these steps. [Evaluator 1]

This comment is not directly concerning the prototype itself, but the data model used to test it, and the familiarity of it by the users. The OpenMRS data model is complex, but on the other hand, it is also realistic as a candidate data model for use with the system. The prototype should be able to handle complex data models, without sacrificing usability. It would, however be reasonable to expect clinicians responsible for the clinical trial to have at least some familiarity with the data model.

71

It was also suggested that the system should be clearer on the semantics of the different fields (properties) that was made available for user selection.

Provide more meaning and consistence to the fields in the system. [Evaluator 4]

For example, there is no way of knowing what the property "accessionNumber" of the observation class means, unless the user has detailed knowledge about the underlying data model. This point is returned to in the future work section (6.2) in chapter six.

None of the evaluators were able to formulate tasks from their own practice (even though one of the evaluator answered "Yes" to this question, no custom tasks were saved for this user). One of the participants ran into connectivity problems while testing and the two others did not have any available criteria to test.

In the final open-ended question, the evaluators were given the opportunity to write down their general comments and thoughts about the system. This is how they responded:

If integrated into OpenMRS (not just as a module), then it could easily replace the Cohort Builder. I liked how you can search from general demographics down to the most specific observations. [Evaluator 2]

This participant suggests that the prototype could be used instead of the Cohort Builder in the OpenMRS client system. This suggestion was passed on to one of the lead-developers of OpenMRS, which responded positively. Due to time constraints, this was however not explored this time. Further exploration of this should certainly be a part of any future work and should involve cooperation with the OpenMRS developers.

A nice tool that could be generalized for developing selection criteria for a wide variety of applications [Evaluator 3]

Again, this participant views this tool as a more general purpose tool for creating logic expressions that can be used in other types of applications.

This is a very interesting project, and I look forward to seeing the outcomes of it. [Evaluator 1]

The evaluator that did not have any previous experience with- or detailed knowledge of- the OpenMRS data model, emphasized the need for a more though trough user interface.

More thoughts need to be given to develop a more user-friendly view of the data.
 [Evaluator 4]

5.4.2 Analysis of the solutions by the evaluators

In this part, I will compare the user-solved tasks with my reference solutions. The goal of this analysis is not to compare the user tasks with an alleged "correct solution" (as there can be many), but rather to learn how the users actually use the prototype by identifying the level of similarity or dissimilarity between the expressions formulated by the test users and the expressions that was formulated on beforehand.

The tasks solved by the evaluators had varying degree of similarity with the reference solutions. Some tasks were almost identical, while others had little or no resemblance at all. Please refer to Appendix D for a complete list of the evaluators' solutions to the different tasks. The level of similarity with the reference solutions is classified in five categories:

o   High (high degree of similarity with the reference solution)
o   Medium (part of the answer is correct)
o   Low (little resemblance with the reference solution)
o   Error (invalid due to a bug with the system)
o   Missing (left blank)

Table 2. *Overview of how Evaluators Solved the Different Tasks*

		Evaluator 1	Evaluator 2	Evaluator 3	Evaluator 4
Task 1	Inclusion	High	High	High	High
	Exclusion	Error	Error	High	Error
Task 2	Inclusion	High	Medium	Missing	Low
	Exclusion	Error	Error	Missing	Error
Task 3	Inclusion	Error	Error	Missing	Low
	Exclusion	Missing	Low	Missing	Low

As shown in Table 2., six of in total 24 tasks were solved with a high degree of similarity to the reference solutions. All of the evaluators succeeded in solving the first inclusion task. For the first exclusion task, the prototype failed in saving the task properly due to an error. As an

unfortunate consequence, the solution became corrupted as one part of the expression repeatedly constituted the whole expression (see Appendix D - evaluation task 1, exclusion criteria 2 by evaluator 2 and 4). This error in the prototype was not discovered before the user tasks were loaded for analysis long after the evaluation session was ended. Evaluator 3 saved each criterion as a separate task, and thereby avoided this error.

Out of the 24 tasks, five were intentionally left completely unanswered. One evaluator alone accounts for four of the missing five tasks. It is reasonable to believe that the corrupted tasks were to at least some extent solved properly. In total, nine of the tasks were left blank or had a low degree of resemblance with the reference task. This can be explained as due to several factors:

- o The complexity of the data model
- o Poor usability provided by the prototype
- o The complexity of the tasks

The OpenMRS data model is complex. It is a result of 30 years of experience with the Regenstrief Medical Record system (Mamlin, et al., 2006). Even though the prototype connects with and reads meta-information about the OpenMRS API (which hides much of the complexity of the underlying relational data model), the OpenMRS API as the user interacts with is still complex and unintuitive to a non-programmer. This was especially brought to light under the observation of evaluator 4 (see 5.4.3) which in particular found the API difficult to navigate in.

The complexity of the data model can also affect the usability as experienced by the evaluators. As stated by Evaluator 4, the high number of properties to each class (many of which did not make any sense, e.g. the "attributeMap" attribute), was somehow confusing and made navigation more difficult.

Due to the reasons given in section 5.3.5, some of the tasks were chosen intentionally because of their high degree of complexity and this in spite of the awareness that they would most likely be impossible to formulate using the prototype. Facing these seemingly invincible tasks may have been a motivation killer.

5.4.3 Observation session

This section presents an elaboration of field notes taken while observing one of the evaluators performing the given evaluation tasks.

At first, the evaluator had trouble knowing what properties to select from the dropdowns. He was not too familiar with the OpenMRS data model, and did not know where the information he wanted to use in the expression was located. With some help, he quickly learned that most of the useful information (except more primitive attributes like year, age, etc) was located at "observations → [where concept is ...]→valueAs(...)". This was clearly perceived as non-intuitive to him. So his first impression was that this tool was very confusing and it was difficult to know where to look for the exact properties. He also suggested that there were too many options, leading to even more confusion. For example, if one wants to set an expression by selecting the [observation where concept is PREGNANCY STATUS] of which the answer is represented in the valueAsBoolean property, all the other properties of the concept data type appeared as choices. In this case, valueAsBoolean should appear as the only valid option for the user. This is however a result of that the tool is designed to be generic. A probable "solution" to this problem would be to add a mechanism to easily add internal data model constraints of which the interface could make use.

The evaluator also suggested that the criteria chosen for evaluation tasks were highly complex, that real world eligibility criteria were usually less complex. Also, he remarked that OpenMRS was maybe not be the best suitable data model as it would be more realistic if the prototype were connected to a data model of a specific study and tested by having the evaluators formulating the exclusion inclusion criteria for that specific study – implicating that the chosen criteria was not "compatible" with the chosen data model. He also said that the evaluation task 3.4 "Ability to return for follow-up" is impossible represent in the system. This would typically be a judgement by the clinical worker, and recorded in the database using a case report form (CRF). This information was, however represented as a concept in the database used in evaluation, but evidently perceived as inaccessible to the user.

5.5 Final words on evaluation

Retrospectively, it is easy to see that the evaluation would have had a much more realistic character if tested against a database designed specifically for a clinical trial, including both

basic patient data demographics, journal history, electronic CRFs etc. It is worth mentioning that a clinical trial may also take advantage of a generic database/data model like OpenMRS, so it may not be unthinkable that this type of comprehensive data model integrated with the Eligibility Criteria Builder. However, most of the problems experienced by the observed evaluator can be attributed to the complexity of the OpenMRS data model, not the prototype itself.

As it turned out to be more difficult than first anticipated to find qualified evaluators, the research performed in this thesis is far from complete. In order to give a qualified judgement of the prototype, its applicability and usability, a greater number of evaluators should have been used for the evaluation. In particular, the Systems Usability Scale (SUS) performs at its best using a sample size of eight evaluators. At a sample size of four, it actually performs slightly worse than its contestants (Figure 15). Thus, for the usability evaluation, it is impossible to give any clear conclusions. A conclusion regarding the open-ended questions about the technology (pt. 2 in the second part of the questionnaire) is neither easy to draw on the basis of such a small sample group. A few general points can however be extracted from the questionnaire and observation session combined:

- o None of the evaluators had knowledge of similar systems[14]
- o All but one of the evaluators would like to use this system in their own practice[15]
- o The prototype has room for improvements on the usability side
- o The high complexity level of the OpenMRS data model used in the evaluation possibly made it more difficult to formulate the tasks as expressions than necessary
- o The eligibility criteria chosen for the evaluation was possibly more complex than they usually are in real-world clinical trials

So, what can be said on the basis of feedback from four users? It is possible to use the data material presented above and the experience gained through the development of this prototype to give directions to further development and research, which is presented in the next chapter.

---

[14] One of the evaluator mentions OpenMRS Cohort builder as a similar system, but as argued in section 2.2 and 2.6, both the implementation and purpose of this is different than of the Eligibility Criteria Builder

[15] One of the evaluators left this question blank

6   CONCLUSIONS AND FUTURE WORK

The research question for this thesis was "*How can a data model independent software tool be developed to support users to formally define criteria for patient eligibility in a clinical trial*". In order to answer this question, a generic, data model independent prototype (Eligibility Criteria Builder) was developed. For evaluation purposes, it was connected to the comprehensive OpenMRS data model. The prototype was evaluated with four different users with both technical and domain knowledge. For the evaluation, eligibility criteria from real-world clinical trial protocol were used. These criteria varied in complexity, from simple (i.e. "age less than 18") to complex (i.e. "HIV infection (previously confirmed by 2 ELISAs for children > 18 months; DNA PCR for those < 18 months)"). The evaluation indicated that the prototype may have been well suited to formulate less complex tasks, while the more complex criteria posed a greater challenge to the users. This can, however, to some extent be attributed to the choice of using the complex OpenMRS data model in the evaluation phase.

The project was in general viewed as an interesting and important project by the evaluators and almost all of them stated that they would like to use the system in their own work. There were different opinions about the ease of use and complexity of the system, indicating that further improvements are needed. Comparing the proposed solutions by the evaluators suggests that familiarity with the OpenMRS data model influenced the successful use of the system. This further indicates that the complexity of the data model has an impact on the usability of the system.

Both the literature study and feedback from the domain experts and the evaluators suggest that the need for a system like the Eligibility Criteria Builder is clear and present and an automated way of determining patient eligibility for a clinical trial can improve both efficiency and safety of clinical trials. The time spent identifying eligible patients in today's paper based regime is significant, and thus has a great potential for improvement. The safety of patients is not automatically ensured by using a computerized system alone, but can be so as a consequence to the fact that it promotes the definition of more formal and unambiguous eligibility criteria.

6.1 Reflection

Retrospectively, it is clear that a simpler data model should be used during evaluation. The complexity of the OpenMRS data model posed a great challenge to the evaluators, and most likely also influenced their appraisal of the usability of the system. If a simpler, yet more intuitive data model was used as the basis for expression formulation, this situation may have looked different. The data model used during evaluation was the OpenMRS API, which is a simplification of the underlying relational database schema. However, the purpose of the API is to provide the programmers a more efficient way of accessing data after they are extracted from the database. Thus, the API is designed for efficacy purposes and not as a means to describe the data model in terms of concepts and their relations. For this purpose, an ontology specification of a clinical trial may have represented a better subject data model for the Eligibility Criteria Builder. As the prototype is developed as a generic tool, there are, however, no restrictions in the prototype that prevents it from being connected to ontology with little effort.

Due to the errors in some of the tasks saved by evaluators, it is also evident that more time should have been spent on error detection and bug fixing. Unfortunately, this error was not noticeable to the user during the evaluation session and was therefore not discovered until much later, when the solved tasks were extracted from the database for analysis.

The number of evaluators should ideally have been twice as large. As the Systems Usability Scale outperforms its peers at a sample size of eight, at least this many users should have participated in the usability evaluation. With a minimum of eight test users, it would have been possible to draw conclusions based on the response to the SUS questionnaire with greater confidence. The evaluators were primarily technologist with a high level of clinical trial domain knowledge. In order to get a better picture on how clinicians with no or little technical competence would judge the system, a representative group that better matches the target user for the prototype should be recruited.

It is also important to reflect on possible unforeseen consequences of using formally defined eligibility criteria to automatically select eligible patients for a clinical trial. Could there be consequences that may even lead to reduced safety for patients? What if an important criterion is missing or left out unintentionally and not discovered? This can certainly be a problem even without a computerized system, but if the eligibility determination is fully

78

automated (i.e. no clinician involvement), such flaws may have less likelihood of being discovered. Therefore, an emphasis on the importance of human involvement is needed. The Eligibility Criteria Builder should never be used to completely automate the enrolment process, but rather as a means to increase the efficiency and reduce the resources needed to determine eligibility of each individual potential participant in a clinical trial.

6.2 Future work

There is a long way to go before the proposed prototype could be made available for production use.

The more general and technical improvements should receive more focus in future development. In particular, the specific issues and areas of improvement disclosed during evaluation should be prioritized (e.g. the dialog window issue as reported by one of the evaluators). A point given by one of the evaluators highlights the importance of better error checking facilities and feedback from the system if inconsistencies occur. Also, improved constraints between properties, data types and valid values (as pointed out by another evaluator) should receive attention in further development. This could be achieved by implementing an extended data model description format.

As it was suggested to view the Eligibility Criteria Builder as a possible future replacement of the OpenMRS Cohort builder, exploring this possibility should also receive more attention in the future.

As one of the challenges with this system was the complexity of the data model, a more intuitive data model should be used as the subject data model to which the prototype is interacting in such an evaluation. In particular, using an ontology as the underlying data model would be a very interesting approach to explore in the future.

Because of lack of access to a database with real patients that actually were recruited to a real-world clinical trial, it was not possible to assess the validity of the answers given by the evaluators. Thus, another important future aspect would be to examine the quality of the expressions produced by the users using the system. To what extent are the right patients identified with help of the computerized criteria created by using the system? Finding the answer to this would require a thorough study in real-world clinical trials that compares the

group of patients selected using computerized eligibility criteria versus the group of patients actually enrolled by clinicians in the trial. Only then will it be possible to uncover the real utility and value of the Eligibility Criteria Builder.

7 BIBLIOGRAPHY

Adikari, S., McDonald, C., & Collings, P. (2006). *A design science approach to an HCI research project*.

Andriessen, D. (2006). *Combining design-based research and action research to test management solutions*.

Avison, D., & Fitzgerald, G. (2003). Information systems development: methodologies, techniques and tools.

Brooke, J. (1996). SUS-A quick and dirty usability scale. *Usability Evaluation in Industry*, 189-194.

Choi, J., Bakken, S., Lussier, Y. A., & MendonÇA, E. A. (2006). Improving the Human Readability of Arden Syntax Medical Logic Modules Using a Concept-oriented Terminology and Object-oriented Programming Expressions. *CIN: Computers, Informatics, Nursing, 24*(4), 220.

Chow, S., & Liu, J. (2004). *Design and analysis of clinical trials: concepts and methodologies*: Wiley-Interscience.

Crockford, D. (2006). The application/json Media Type for JavaScript Object Notation (JSON). *Request for Comments, 4627*.

Dix, A., Finlay, J., & Abowd, G. D. (2004). *Human-Computer Interaction*: Prentice Hall.

Fink, E., Kokku, P. K., Nikiforou, S., Hall, L. O., Goldgof, D. B., & Krischer, J. P. (2004). Selection of patients for clinical trials: an interactive web-based system. *Artificial Intelligence In Medicine, 31*(3), 241-254.

Fridsma, D. B., Evans, J., Hastak, S., & Mead, C. N. (2007). The BRIDG Project: A Technical Report. *J Am Med Inform Assoc*, M2556.

Friedman, L. M., Furberg, C. D., & DeMets, D. L. (1996). *Fundamentals of Clinical Trials*: CV Mosby.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *Management Information Systems Quarterly, 28*(1), 75-106.

Jenders, R. A., Corman, R., & Dasgupta, B. (2003). *Making the standard more standard: a data and query model for knowledge representation in the Arden syntax*.

Järvinen, P. (2005). *Action Research as an approach in design science.* Paper presented at the European Academy of Management Conference, Munich, Germany.

Lewin, K. (1946). Action research and minority problems. *Journal of Social Issues, 2*(4), 34-46.

Mamlin, B., Biondich, P., Wolfe, B., Fraser, H., Jazayeri, D., Allen, C., et al. (2006). *Cooking Up An Open Source EMR For Developing Countries: OpenMRS–A Recipe For*

Successful Collaboration.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems, 15*, 251-266.

May, I. C. H. (1996). *ICH Harmonised Tripartite Guideline: Guideline for Good Clinical Practice (GCP) E6*.

Ohno-Machado, L., Wang, S., Mar, P., & Boxwala, A. (1999). *Decision support for clinical trial eligibility determination in breast cancer*.

Oliveira, A. G., & Salgado, N. C. (2006). Design aspects of a distributed clinical trials information system. *Clinical Trials, 3*(4), 385.

OpenMRS Wiki (2008). Retrieved 22th of october, 2008, from http://openmrs.org/wiki/Dictionary_101

Orrill, C. H., Hannafin, M. J., & Glazer, E. M. (2004). Disciplined inquiry and the study of emerging technology. *Handbook of research for educational communications and technology: A project of the Association for Educational Communications and Technology*, 335-354.

Piantadosi, S. (1997). *Clinical Trials: A Methodologic Approach*: Wiley.

Rapoport, R. N. (1970). Three Dilemmas in Action Research: With Special Reference to the Tavistock Experience. *Human Relations, 23*(6), 499.

Richardson, C. (2008). ORM in Dynamic Languages. *Queue, 6*(3), 28-37.

Robson, C. (2002). *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*: Blackwell Publishers.

Scriven, M. (1991). *Evaluation Thesaurus*: Sage.

Sommerville, I. (2007). *Software Engineering* (8th ed.): Addison-Wesley.

Susman, G. I., & Evered, R. D. (1978). An Assessment of the Scientific Merits of Action Research. *Administrative Science Quarterly, 23*(4), 582-603.

Tu, S., Kemper, C., Lane, N., Carlson, R., & Musen, M. (1993). A methodology for determining patients eligibility for clinical trials. *Methods of Information in Medicine, 32*, 317-317.

Tullis, T. S., & Stetson, J. N. (2004). *A Comparison of Questionnaires for Assessing Website Usability*.

Wang, F., & Hannafin, M. J. (2005). Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development, 53*(4), 5-23.

Whippen, D., Deering, M. J., & Ambinder, E. P. (2007). Advancing High-Quality Cancer Care. *Journal of Oncology Practice, 3*(4).

Willoughby, C., Fridsma, D., Chatterjee, L., Speakman, J., Evans, J., & Kush, R. (2007). A Standard Computable Clinical Trial Protocol: The Role of the BRIDG Model. *Drug Information Journal, 41*, 383-392.

APPENDIX A. THE EVALUATION E-MAIL AND INSTRUCTIONS

E-mail template

```
Dear <NAME>,

Thank you for taking interest in evaluating the The Eligibility
Criteria Builder Prototype.

Please follow the instructions in the attached Evaluation
Guidelines PDF document.
When done, please return a filled-in version of the Questionnaire
document.

If you have any problems opening the files or web-address, or
encounter any problems
during testing, please let me know immediately.

Your participant id is: <PARTICIPANT-ID>

Best regards,
Bjørge Næss

Master student at Department of Information Science and Media
Studies
University of Bergen (UoB)
www.infomedia.uib.no
```

Evaluation guidelines.pdf

1 EVALUATION GUIDELINES

Thank you for taking interest in evaluating the The Eligibility Criteria Builder Prototype.

The evaluation is expected to take between 1 ½ to 2 hours to complete.

2 PREREQUISITES

In order to test the prototype, you will need a working internet connection and Mozilla Firefox version 2.0 or higher installed on your computer.

3 GETTING STARTED

Open Mozilla Firefox and go to the address http://ecbuilder.dyndns.org

Enter your four-digit participant id that you received in the e-mail along with this document. Now, you will see three different buttons: "Start practice", "Start test" and "Restart session"

To get to know the prototype and how it works, click "Start practice" and continue with the practice tasks listed below. You can consult the documentation at any time by clicking on the "Documentation" button on the toolbar. Section 3 in the Documentation explains how to create your first criteria expression.

3.1 Practice tasks

Create a new expression for each of the following criteria.

NOTE: Create a separate expression for each task's inclusion and exclusion criteria, and don't forget to save each expression before you continue.

1. Practice task #1

Inclusion criteria:

  1.1. Working at the Department of Social Anthropology
  1.2. Age is above 24 years
  1.3. Address is somewhere in Norway
  1.4. Job title does not contain "managing".

Exclusion criteria:

  1.5. 50 years or younger
  1.6. Working at department of Information and Media Sciences

      1.7.   Living outside Norway

2. Practice task #2

Inclusion criteria:

      3.1.   Born before fifteenth of May 1976
      3.2.   Working at a department that is located either in Tanzania or in Uganda, but not in the big cities of Dar es Salaam or Kampala.
      3.3.   Working at a Department whose name contains Faculty

Exclusion criteria:

      3.4.   Current address is outside Tanzania or Uganda
      3.5.   Working on a project ended within the last year with "water" in the project name

When you are done, remember to save the last expression before you close the window and go on with the actual test by clicking on the "Start test" button.

3.2   Test tasks

These are authentic eligibility criteria taken from real clinical trial protocols. The first task contains criteria from different protocols, while task two and three are from 1 and 1, respectively.

3.2.1   Read this first

We are now moving away from Employees, departments and projects, and over to the Patient context. You will now work on the OpenMRS data structure and some basic information about its different parts follows ("OpenMRS Wiki," 2008):

- **Encounter**: A single, specific interaction between the patient and the provider. An encounter can be any interaction, including doctor visits, home visits, counselor appointments, etc. Encounters are typically represented as a form, consisting of hundreds of observations.
- **Observations**: Anything actively measured or observed during an encounter. As an example, patients' weights, heights, blood pressures, and BMIs are

observations, as well as qualitative facts including the number of years a patient smoked, the activities in which the patient experiences shortness of breath, and finding on an X-ray. Although typically an observable question, demographics are an exception, and are recorded as separate concepts.

- **Concepts** are the individual data point collected form a population of patients. Concepts include both questions and answers; for example, the question of blood type is a possible concept, but the responses, "A", "B," & "O" would be considered concepts as well. The bottom line is, if you have a medical concept of any sort, and it's needed within your records system, it needs to be defined within the dictionary.

The patients' observations are where to look for medical information about a patient. Every patient has a list of observations, and each observation has a concept and a value. According to the type of concept, the value contains the recorded information. If the concept is a question, the value contains the answer to that question. If the concept is a measurement of height, the value contains the height as a numeric value. Some concepts has a list of possible values associated with it. For example, the concept CLINIC TRAVEL TIME, which refers to a question on an encounter form asking "How long did it take you to travel to clinic today?" has four other concepts as valid answers: LESS THAN 30 MINUTES, ONE TO TWO HOURS, 30 TO 60 MINUTES or MORE THAN TWO HOURS.

The concept PREGNANCY STATUS, referring to the question "Is the patient pregnant?" is of data type Boolean and therefore it has only two valid answers: True or False.

Depending on the type of observation concept, the value is located in valueCoded (for concept answers), valueNumeric (for observations requiring a numeric value), valueAsBoolean (for most observations that determines a diagnosis or yes/no questions), valueDrug (if the concept is a drug) and valueDatetime (where a certain date is recorded) for the observations.

For the CLINIC TRAVEL TIME question, requiring the answer to be ONE TO TWO HOURS will be formulated as:

observations [where concept is CLINIC TRAVEL TIME] valueCoded is ONE TO TWO HOURS

In the prototype, this expression would look like:

Because patients have multiple observations, [where (…)] is used to single out the exact observation we want to use in the expression.

If the patient must be pregnant, the expression will look like:

observations [where concept is PREGNANCY STATUS] valueAsBoolean is true

In the prototype, this would be:

The test tasks are listed below. Please formulate each task's inclusion and exclusion criteria separate from each other, and remember to select the correct task name from the list.

Note: If you find any of the criteria too difficult to formulate, you can always save it and go on with the next, but if you do so, please leave a comment in the Textual description field.

1. Test task #1

Inclusion Criteria:

1.1. Males or females aged greater than or equal to 1 to less than 4 years1
1.2. Known residents of the village of Bancoumana, Mali or its surrounding area1
1.3. Identified as having a malaria infection by blood film examination1

Exclusion Criteria:

    1.4. Convulsions or history of convulsions1

    1.5. Known hypersensitivity or allergy to artemisinin derivatives or mefloquine or mefloquine chemically related compounds (for example quinine and quinidine)1

    1.6. Presence of any known serious chronic disease (e.g. AIDS, sickle cell disease, malignancy)1

2. Test task #2

Inclusion Criteria:

    2.1. Age > 1 year

    2.2. Axillary temperature ≥ 37.5ºC and/or history of fever in the previous 48 hours without any other evident cause

    2.3. Unmixed infection with P. falciparum of between 250 and 100,000 asexual parasites/mm3 as determined by microscopic exam of the thick or thin smear

    2.4. An informed consent obtained from the patient or his/her guardian (in case of patients ≤ 18 years old) and assent for children (8-18 years old)

Exclusion Criteria:

    2.5. Other severe chronic diseases (e.g., cardiologic, renal, or hepatic diseases; HIV/AIDS; severe malnutrition)

    2.6. History of allergy to mefloquine, artesunate, quinine, tetracycline, or clindamycin

    2.7. Pregnancy (based on urine test), since this group of patients receives other drugs for malaria treatment in accordance with Peruvian national guidelines.

3. Test task #3

Inclusion Criteria:

    3.1. Aged 1 to 5 years

3.2. HIV infection (previously confirmed by 2 ELISAs for children > 18 months; DNA PCR for those < 18 months)

3.3. Informed consent from the parent/caretaker

3.4. Ability to return for follow-up (lives within a radius of 15 km from hospital and unlikely to change residence during the course of the study)

Exclusion Criteria:

3.5. Children already enrolled in other studies

3.6. Children with severe abnormalities which are likely to impair oral intake (for example, severe cerebral palsy)

3.7. Severely ill children requiring urgent admission and resuscitation

4. Additional tasks

Load the first test task's Inclusion criteria for editing and change it so that only females are included in this study. Save the expression again when you are done.

Can you think out some tasks that you experienced in your practice and express it using the system? If you do, you can save it as "My task #". If you encounter difficulties or were unable to accomplish it, please write a short note about it in the attached questionnaire.

When done, please open the attached questionnaire and answer the questions.

5. References

OpenMRS Wiki. (2008). Retrieved 22th of october, 2008, from http://openmrs.org/wiki/Dictionary_101

Project background.pdf

Project background

Testing the effects of a new treatment involves exposure of the treatment to a group of patients. These patients are a subset of a greater group of participants, where the subset is selected based on certain eligibility criteria. These criteria are traditionally expressed in natural language in its own section in a protocol. The protocol is a document written before the study is carried out. It describes the planned study in detail and acts as a guideline to all parties involved in the study. The eligibility criteria section can contain both inclusion and exclusion criteria. The inclusion criteria are a list of conditions each and every patient must meet in order to be enrolled in the study (i.e. the patient is diagnosed with the disease the treatment is supposed to treat). Similarly, the exclusion criteria list conditions that would disqualify the patient for enrolment in the study (i.e. a known allergic reaction to the medicine in question).

When a clinical trial is carried out, the eligibility criteria are used to enroll new patients in the study. This is done by cross-checking medical records for every patient against each criterion. Some large-scale clinical trials need up-to 3000 eligible patients (and a considerable bigger group of potential participants). So, in consequence the patient selection is a highly extensive and time-consuming task.

Traditionally, the inclusion and exclusion criteria are written in natural language in the protocol document. This opens for ambiguities and different interpretations among clinical researchers (for example, when reading the Eligibility section in a protocol document, it is only implicit to the reader that a patient must met ALL of the inclusion criteria and NONE of the exclusion criteria in order to be enrolled).

In the OMEVAC project, the goal is to move from a paper-based way of conducting clinical trials to exploit the possibilities given by computers and electronic capture devices. One of the achievements will be patient databases and electronic medical records. Within this prospective, automatic patient selection based on a computerized eligibility criteria can reduce the resources needed to enroll patients, and possible reduce the risk of misinterpretation of the inclusion and exclusion criteria, which in turn can be a potential safety hazard for patients involved in the study.

APPENDIX B. EVALUATOR RESPONSES

Evaluator 1

Part I

	Strongly disagree				Strongly agree

1. I think that I would like to use this system frequently

			X	
1	2	3	4	5

2. I found the system unnecessarily complex

	X			
1	2	3	4	5

3. I thought the system was easy to use

				X
1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system

	X			
1	2	3	4	5

5. I found the various functions in this system were well integrated

				X
1	2	3	4	5

6. I thought there was too much inconsistency in this system

X				
1	2	3	4	5

7. I would imagine that most people would learn to use this system very quickly

		X		
1	2	3	4	5

8. I found the system very cumbersome/awkward to use

			X	
1	2	3	4	5

9. I felt very confident using the system

		X		
1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system

			X	
1	2	3	4	5

Part II

1. Experience with other systems

a) I know of similar systems

	X
Yes	No

b) I have used similar system(s) before

	X
Yes	No

If answer is No, go to d)

c) I have used these systems before:

d) I will summarize the major difference between the system(s) I have previously used and this as:

2. Technology

a) I would like to use a system

X	

93

like this in my own practice

Yes	No

b) In what ways, if any, could this system improve your current practice?

By forcing the eligibility criterias to be defined so that they can also be computerized. Some of them were to my knowledge very difficult to implement (or impossible).

c) What potential *advantages*, if any, could the use of a system like this have?

It ensures that the protocol is adhered to

d) What potential *disadvantages*, if any, could the use of a system like this have?

It may loose some of the flexibility and medical based decisions.

e) How can this system be improved in terms of functionality and user friendliness?

In this setup, the major limiting factor is the fact that it uses the OpenMRS API, which to people not familiar with it, is complex. But a person knowing OpenMRS would very quickly be able to complete these steps.

f) Were you able to formulate any tasks from your own practice? If not, what was the problem?

Did not have any that I had at hand when doing this test.

g) General comments, thoughts, etc.

This is a very interesting project, and I look forward to seeing the outcomes of it.

Evaluator 2

Part I

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently					X
	1	2	3	4	5

	Strongly disagree				Strongly agree
2. I found the system unnecessarily complex		X			
	1	2	3	4	5

3. I thought the system was easy to use				X	
	1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system		X			
	1	2	3	4	5

5. I found the various functions in this system were well integrated					X
	1	2	3	4	5

6. I thought there was too much inconsistency in this system		X			
	1	2	3	4	5

7. I would imagine that most people would learn to use this system very quickly			X		
	1	2	3	4	5

8. I found the system very cumbersome/awkward to use		X			
	1	2	3	4	5

9. I felt very confident using the system				X	
	1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system

		X		
1	2	3	4	5

Part II

1. Experience with other systems

   a) I know of similar systems

	X
Yes	No

   b) I have used similar system(s) before

	X
Yes	No

   If answer is No, go to d)

   c) I have used these systems before:

OpenMRS Cohort Builder

   d) I will summarize the major difference between the system(s) I have previously used and this as:

This system can search for very, very specific information; at a far greater depth than Cohort Builder.

2. Technology

    a) I would like to use a system like this in my own practice

X	
Yes	No

    b) In what ways, if any, could this system improve your current practice?

> It could allow clinicians in our organization to quickly find patients matching very specific criteria. This of course could expedite the entire process of analysis.

    c) What potential *advantages*, if any, could the use of a system like this have?

> Again, allowing one to find highly specific information quickly.

    d) What potential *disadvantages*, if any, could the use of a system like this have?

> Because it can be so specific, only people with specialized knowledge in that field could operate the system.

    e) How can this system be improved in terms of functionality and user friendliness?

> When setting conditions (ex: where [concept is CDC CATEGORY C]) another window opens, but when setting conditions in there, yet another window can open, this time right on top of the first, making it hard to distinguish between the two. This should be resolved. Also, I would like to be able to search for terms within concept descriptions (ex: selecting a concept that contains the word "chronic" in the description).

    f) Were you able to formulate any tasks from your own practice? If not, what was the problem?

> No. I ran into connectivity problems.

    g) General comments, thoughts, etc.

> If integrated into OpenMRS (not just as a module), then it could easily replace the Cohort Builder. I liked how you can search from general demographics down to the most specific observations.

Evaluator 3

Part I

	Strongly disagree				Strongly agree

1. I think that I would like to use this system frequently

				X
1	2	3	4	5

2. I found the system unnecessarily complex

			X	
1	2	3	4	5

3. I thought the system was easy to use

	X			
1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system

	X			
1	2	3	4	5

5. I found the various functions in this system were well integrated

			X	
1	2	3	4	5

6. I thought there was too much inconsistency in this system

X				
1	2	3	4	5

7. I would imagine that most people would learn to use this system very quickly

			X	
1	2	3	4	5

8. I found the system very

	X			

98

cumbersome/awkward to use

1	2	3	4	5

9. I felt very confident using the system

			X	
1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system

			X	
1	2	3	4	5

Part II

1. Experience with other systems

   a) I know of similar systems

	X
Yes	No

   b) I have used similar system(s) before

	X
Yes	No

   If answer is No, go to d)

   c) I have used these systems before:

   d) I will summarize the major difference between the system(s) I have previously used and this as:

I haven't used such a tool before.

2. Technology

   a) I would like to use a system like this in my own practice

X	
Yes	No

b) In what ways, if any, could this system improve your current practice?

It would be useful if the system could import a user specified dictionary so that it could adapted for developing Inclusion & Exclusion criteria for any type of selection procedure, not just in medicine.

c) What potential *advantages*, if any, could the use of a system like this have?

To quickly develop selection criteria for all sorts of sample surveys.

d) What potential *disadvantages*, if any, could the use of a system like this have?

Domain-specific dictionaries should be possible to add to the software by the user.

This system does not point out errors in the expressions.

e) How can this system be improved in terms of functionality and user friendliness?

It would help if the system permitted me to edit the Code for the expressions directly as that is sometimes required for better control.

Errors in expressions should be highlighted so that they can be easily corrected.

f) Were you able to formulate any tasks from your own practice? If not, what was the problem?

Not applicable

g) General comments, thoughts, etc.

A nice tool that could be generalized for developing selection criteria for a wide variety of applications

Evaluator 4

Part I

		Strongly disagree				Strongly agree

1. I think that I would like to use this system frequently

	1	2	3	4	5

2. I found the system unnecessarily complex

			x	
1	2	3	4	5

3. I thought the system was easy to use

	x			
1	2	3	4	5

4. I think that I would need the support of a technical person to be able to use this system

	x			
1	2	3	4	5

5. I found the various functions in this system were well integrated

			x	
1	2	3	4	5

6. I thought there was too much inconsistency in this system

			x	
1	2	3	4	5

7. I would imagine that most people would learn to use this system very quickly

	x			
1	2	3	4	5

8. I found the system very cumbersome/awkward to use

	x			
1	2	3	4	5

9. I felt very confident using the

		x		
1	2	3	4	5

system

10. I needed to learn a lot of things before I could get going with this system

		x		
1	2	3	4	5

Part II

1. Experience with other systems

   a) I know of similar systems

	x
Yes	No

   b) I have used similar system(s) before

	x
Yes	No

   If answer is No, go to d)

   c) I have used these systems before:

   No

   d) I will summarize the major difference between the system(s) I have previously used and this as:

   NA

2. Technology

   a) I would like to use a system like this in my own practice

Yes	No

   b) In what ways, if any, could this system improve your current practice?

   NA

c) What potential *advantages*, if any, could the use of a system like this have?

> This system could help in ensuring that data collected conforms to predetermined conditions thereby reducing errors.

d) What potential *disadvantages*, if any, could the use of a system like this have?

> It might be too restrictive and inflexible for the end user.

e) How can this system be improved in terms of functionality and user friendliness?

> Provide more meaning and consistence to the fields in the system.

f) Were you able to formulate any tasks from your own practice? If not, what was the problem?

> Yes

g) General comments, thoughts, etc.

> More thoughts need to be given to develop a more user-friendly view of the data.

APPENDIX C. EVALUATION TASKS AND CLINICALTRIALS.GOV IDENTIFIERS

Evaluation task 1 is compounded by single criteria from different protocols and is followed by the clinicaltrials.gov identifier is in parenthesis. All criteria in tasks 2 and 3 is taken from protocols with identifier NCT00164216 and NCT00122941, respectively. The full protocol document is found by searching for the identifier at http://clinicaltrials.gov/ct2/search.

Evaluation task 1

Inclusion Criteria:

1.1. Males or females aged greater than or equal to 1 to less than 4 years (*NCT00740090*)
1.2. Known residents of the village of Bancoumana, Mali or its surrounding area (*NCT00740090*)
1.3. Identified as having a malaria infection by blood film examination (*NCT00167739*)

Exclusion Criteria:

1.4. Convulsions or history of convulsions (*NCT00167739*)
1.5. Known hypersensitivity or allergy to artemisinin derivatives or mefloquine or mefloquine chemically related compounds (for example quinine and quinidine) (*NCT00243737*)
1.6. Presence of any known serious chronic disease (e.g. AIDS, sickle cell disease, malignancy) (*NCT00327964*)

Evaluation task 2

All taken from NCT00164216

Inclusion Criteria:

2.1. Age > 1 year
2.2. Axillary temperature $\geq$ 37.5ºC and/or history of fever in the previous 48 hours without any other evident cause
2.3. Unmixed infection with P. falciparum of between 250 and 100,000 asexual parasites/mm3 as determined by microscopic exam of the thick or thin smear
2.4. An informed consent obtained from the patient or his/her guardian (in case of patients $\leq$ 18 years old) and assent for children (8-18 years old)

Exclusion Criteria:

2.5. Other severe chronic diseases (e.g., cardiologic, renal, or hepatic diseases; HIV/AIDS; severe malnutrition)

2.6. History of allergy to mefloquine, artesunate, quinine, tetracycline, or clindamycin

2.7. Pregnancy (based on urine test), since this group of patients receives other drugs for malaria treatment in accordance with Peruvian national guidelines.

Evaluation task 3

All taken from NCT00122941

Inclusion Criteria:

3.1. Aged 1 to 5 years

3.2. HIV infection (previously confirmed by 2 ELISAs for children > 18 months; DNA PCR for those < 18 months)

3.3. Informed consent from the parent/caretaker

3.4. Ability to return for follow-up (lives within a radius of 15 km from hospital and unlikely to change residence during the course of the study)

Exclusion Criteria:

3.5. Children already enrolled in other studies

3.6. Children with severe abnormalities which are likely to impair oral intake (for example, severe cerebral palsy)

3.7. Severely ill children requiring urgent admission and resuscitation

APPENDIX D. REFERENCE SOLUTIONS AND EVALUATOR SOLUTIONS

Evaluation task 1

Inclusion criteria

Reference solution

```
( age is greater than or equal to 1 and age is less than 4 and addresses→[ where (
preferred is true )]→cityVillage is Bancoumana and addresses→[ where ( preferred
is true )]→country is MALI (ML) and observations→[ where ( concept is MALARIAL
SMEAR )]→valueCoded is POSITIVE )
```

Evaluator 1

```
 ( ( age is greater than or equal to 1 and age is less than 4 ) and ( addresses→[
where ( cityVillage is Bancaoumana and country is MALI (ML) )] ) and (
observations→[ where ( concept is MALARIAL SMEAR )]→valueCoded is POSITIVE ) )
```

Evaluator 2

```
 ( ( gender is M or gender is F ) and ( age is greater than or equal to 1 and age
is less than 4 ) and ( personAddress→country is Bancoumana or
personAddress→country is MALI (ML) ) and observations→[ where ( concept is
MALARIAL SMEAR )]→valueCoded is POSITIVE )
```

Evaluator 3

- (observations→[where (concept is PATIENT AGE and (valueNumeric is greater
  than or equal to 1 and valueNumeric is less than 4))])
- (observations→[where (concept is MALARIAL SMEAR)])
- (observations→[where (location→region is Bancoumana or location→region is
  Mali or location→region is Neighbouring area)])

Evaluator 4

```
( age is greater than or equal to 1 and age is less than 4 addresses→[ where (
cityVillage is Bancouman )]→country is MALI (ML) and observations→[ where (
concept is MALARIA, MILD or concept is MALARIAL SMEAR )]→valueCoded is POSITIVE )
```

Exclusion criteria

Reference solution

```
( observations→[ where ( concept is CONVULSION )]→valueAsBoolean is true or
observations→[ where ( concept is ARTEMISININ HYPERSENSITIVITY OR ALLERGY
)]→valueAsBoolean is true or observations→[ where ( concept is AIDS
)]→valueAsBoolean is true or observations→[ where ( concept is SICLE CELL DISEASE
)]→valueAsBoolean is true )
```

Evaluator 1

```
( observations→[ where ( concept is AIDS or concept is SICLE CELL DISEASE
)]→valueAsBoolean is true or ( observations→[ where ( concept is AIDS or concept
is SICLE CELL DISEASE )]→valueAsBoolean is true ) or observations→[ where (
concept is AIDS or concept is SICLE CELL DISEASE )]→valueAsBoolean is true )
```

Evaluator 2

```
( observations→[ where ( concept is AIDS )]→valueAsBoolean is true and (
observations→[ where ( concept is AIDS )]→valueAsBoolean is true or
observations→[ where ( concept is AIDS )]→valueAsBoolean is true ) and
observations→[ where ( concept is AIDS )]→valueAsBoolean is true )
```

Evaluator 3

```
•  ( observations→[ where ( concept is OTHER AND UNSPECIFIED CONVULSIONS )] )
•  ( observations→[ where ( concept is ARTEMISININ HYPERSENSITIVITY OR ALLERGY or
   concept is ALLERGY TO MEFLOQUINE RELATED DRUGS )] )
•  ( observations→[ where ( concept is CHRONIC DISEASE and valueCoded is AIDS or
   valueCoded is SICKLE CELL or valueCoded is MALIGNANCY )] )
```

Evaluator 4

```
( observations→[ where ( concept is AIDS )]→valueAsBoolean is true and
observations→[ where ( concept is AIDS )]→valueAsBoolean is true and
observations→[ where ( concept is AIDS )]→valueAsBoolean is false )
```

Evaluation task 2

Inclusion criteria

Reference solution

```
( birthdate→yearsSince is greater than 1 and encounters→[ where ( obs→[ where (
concept is TEMPERATURE (C) )]→valueNumeric is greater than 37,5 )]→obs→empty is
false and observations→[ where ( concept is INFORMED CONSENT GIVEN
)]→valueAsBoolean is true and observations→[ where ( concept is INFORMED CONSENT
GIVEN )]→valueAsBoolean is true )
```

Evaluator 1

```
( age is greater than 1 and ( encounters→[ where ( encounterDatetime→hoursSince is
less than or equal to 48 )]→allObs→[ where ( concept is TEMPERATURE (C)
)]→valueNumeric is greater than 37.5 ) and ( observations→[ where ( concept is
INFORMED CONSENT GIVEN )]→valueAsBoolean ) and ( observations→[ where ( concept is
INFORMED CONSENT GIVEN )]→valueAsBoolean is true ) )
```

Evaluator 2

```
( age is greater than 1 and ( observations is true and observations is greater than
37.5 ) or ( observations is true and observations is greater than 37.5 and
```

```
observations is true and observations Is before Sun Nov 30 2008 19:00:00 GMT+0100
and observations Is after Fri Nov 28 2008 19:00:00 GMT+0100 ) or ( observations Is
before Sun Nov 30 2008 19:00:00 GMT+0100 and observations Is after Fri Nov 28 2008
19:00:00 GMT+0100 ) and observations )
```

Evaluator 3

```
Missing
```

Evaluator 4

```
( age is greater than 1 and observations is greater than 37.5 and observations is
true and observations )
```

Exclusion criteria

Reference solution

```
( observations→[ where ( concept is HIV POS )]→valueAsBoolean is true or
observations→[ where ( concept is RENAL DISEASE )]→valueAsBoolean is true or
observations→[ where ( concept is MALNUTRITION )]→valueAsBoolean is true or
observations→[ where ( concept is ALLERGY TO MEFLOQUINE RELATED DRUGS
)]→valueAsBoolean is true or observations→[ where ( concept is PREGNANCY
)]→valueAsBoolean is true )
```

Evaluator 1

```
( age is greater than or equal to 1 and age is less than or equal to 5 and (
observations→[ where ( concept is INFORMED CONSENT GIVEN )]→valueAsBoolean is true
or observations→[ where ( concept is INFORMED CONSENT GIVEN )]→valueAsBoolean is
DETECTED ) and observations→[ where ( concept is INFORMED CONSENT GIVEN
)]→valueAsBoolean is true and )
```

Evaluator 2

```
( observations→[ where ( concept is URINE PREGNANCY TEST )]→valueCoded and
observations→[ where ( concept is URINE PREGNANCY TEST )]→valueCoded is true and
observations→[ where ( concept is URINE PREGNANCY TEST )]→valueCoded is POSITIVE )
```

Evaluator 3

```
Missing
```

Evaluator 4

```
( observations→[ where ( concept is PREGNANCY )]→valueAsBoolean is true
observations→[ where ( concept is PREGNANCY )]→valueAsBoolean is true and
observations→[ where ( concept is PREGNANCY )]→valueAsBoolean is true )
```

Evaluation task 3

Inclusion criteria

Reference solution

```
( age is less than 5 and age is greater than or equal to 1 and observations→[
where ( concept is HIV POS )]→valueCoded is POSITIVE and observations→[ where (
concept is HOSPITAL DISTANCE )]→valueNumeric is less than 15 and ( (
observations→[ where ( concept is ELISA )]→valueCoded is POSITIVE and
birthdate→monthsSince is greater than 18 ) or ( birthdate→monthsSince is less than
18 and observations→[ where ( concept is HIV DNA POLYMERASE CHAIN REACTION
)]→valueCoded is DETECTED ) ) )
```

Evaluator 1

```
 ( observations→[ where ( concept is URINE PREGNANCY TEST )]→valueCoded is true or
observations→[ where ( concept is URINE PREGNANCY TEST )]→valueCoded is true or
observations→[ where ( concept is URINE PREGNANCY TEST )]→valueCoded is POSITIVE )
```

Evaluator 2

```
( ( age is greater than or equal to 1 and age is less than or equal to 5 ) and (
observations is POSITIVE and observations is POSITIVE and age is greater than 1.5 )
or ( observations is POSITIVE and age is less than 1.5 ) and observations and
observations )
```

Evaluator 3

```
Missing
```

Evaluator 4

```
( age is greater than 1 and age is less than 5 and observations→[ where ( concept
is HIV POS )]→valueAsBoolean is false )
```

Exclusion criteria

Reference solution

```
( currentStudies→size is 0 or observations→[ where ( concept is CELEBRAL PALSY
)]→valueAsBoolean is true )
```

Evaluator 1

```
Missing
```

Evaluator 2

```
( studies→[ where ( endDate Is after Sat Nov 29 2008 19:00:00 GMT+0100 )] and
observations→[ where ( concept is CDC CATEGORY C )]→valueAsBoolean is true and
observations→[ where ( concept is CDC CATEGORY C )]→valueAsBoolean is true )
```

Evaluator 3

```
Missing
```

Evaluator 4

```
( relationships )
```

APPENDIX E. THE CLASSREADER LOOKUP (JAVASCRIPT)

```
1    Register.Lookup.add({
2        id: 'classreader',
3        property: /.*/,
4        _cache: new Hash(),
5        getItems: function(args) {
6            var queryString = Object.toQueryString(args)
7            var url = 'backend/classreader?'+queryString
8
9            var list = this._cache.get(queryString)
10
11           if (list) return list
12
13           new Ajax.Request(url, {
14               asynchronous: false,
15               evalJSON: true,
16               onComplete: function(response) {
17
18
19                   var res = response.responseJSON;
20
21                   if (res == null || typeof res != 'object')
console.error('Could not retrieve choices from url: '+url)
22
23                   list = []
24
25                   // Create action nodes
26                   if (res.actions)
27                       for (var i = 0; i < res.actions.length; i++)
28                           list.push(ActionNode.create(res.actions[i]))
29
30                   // Create property nodes
31                   if (res.properties)
32                       for (var j = 0; j < res.properties.length; j++)
33                           list.push(PropertyNode.create(res.properties[j]))
34
35                   this._cache.set(queryString, list)
36
37               }.bind(this)
38           });
39
40           return list
41       }
42   })
```

APPENDIX F. THE TYPEEXTENSIONS GROOVY CLASS

```groovy
1      import java.lang.reflect.Method
2      import java.lang.reflect.ParameterizedType
3      import java.lang.reflect.Type
4
5      /**
6       * Created by IntelliJ IDEA.
7       * User: Bjørge
8       * Date: 17.okt.2008
9       * Time: 15:08:30
10      */
11
12     class TypeExtensions {
13         private static HashMap<Class, Class> extenders = new HashMap<Class,
Class>();
14
15         /**
16          * Adding methods of extender class to the extendee class
17          * All instances of the extendee class will have methods of the extender
class
18          */
19         public static extend(Class extendee, Class extender) {
20             extenders.put(extendee, extender)
21             for (Method m in extender.getDeclaredMethods()) {
22                 def method = m.name
23                 extendee.metaClass."$method" << {->
24                     extender."$method"(delegate)
25                 }
26             }
27         }
```

APPENDIX G. THE GETALLMETHODS METHOD

```
28          /**
29           * Returns a list of methods for extended class
30           */
31          public static List<Method> getAllMethods(Type clazz) {
32              if (!clazz) return null
33              if (extenders.containsKey(clazz))
34                  return Arrays.asList(clazz.getMethods())
35                      .plus(Arrays.asList(extenders.get(clazz).getMethods()))
36              else
37                  return Arrays.asList(clazz.getMethods())
38          }
39
40          public static Method findMethod(Type type, String property) {
41              Method foundMethod = (Method)getAllMethods(type).find({
42                  return (
43                      it.name == property ||
44                      it.name == toBeanReadMethod("get", property) ||
45                      it.name == toBeanReadMethod("is", property)
46                  )
47              })
48              //println "Found: $foundMethod"
49              return foundMethod
50          }
51          /**
52           * Looks up the extender class for parameter class
53           */
54          public static Class getExtenderForClass(Class clazz) {
55              return extenders.get(clazz)
56          }
57
58          /**
59           * Cleans the expression for collection operators and returns it
60           */
61          private static String prepareExpression(String expression) {
62              String newexp = "";
63              int level = 0;
64              for (char ch : expression.toCharArray()) {
65                  if (ch == '{') level++;
66                  else if (ch == '}') level--;
67                  else if (level == 0) newexp += ch;
68              }
69              return newexp;
70          }
71          /**
72           * Converts a bean-property to an equivalent method using prefix
73           * Prefix would typically be either 'get', 'set' or 'is'
74           */
75          private static String toBeanReadMethod(String prefix, String property) {
76              return
prefix+property.substring(0,1).toUpperCase()+property.substring(1)
77          }
78
```

APPENDIX H. THE GETENDCLASS METHOD

```
79        /**
80         * Reads a property path and returns the return value of the last
81         * corresponding get-method.
82         */
83        public static Type getEndClass(Type clazz, String expression) {
84
85            expression = prepareExpression(expression);
86            if (expression.equals("")) return clazz;
87            String[] exprs = expression.split("\\.");
88
89            for (String prop : exprs) {
90                try {
91
92                    Method foundMethod = null;
93
94                    // If the class is a parameterized type, get the inner type
95                    if (clazz instanceof ParameterizedType) {
96
97                        // This is the enclosing type
98                        Type rawType = ((ParameterizedType)clazz).getRawType()
99
100                        // This is the inner type
101                        clazz =
((ParameterizedType)clazz).getActualTypeArguments()[0];
102
103                        // First, look for the property in the inner type
104                        foundMethod = findMethod(clazz, prop)
105
106                        // If no matching method of enclosing type could be found,
107                        // try to look at the enclosing type
108                        if (!foundMethod) foundMethod = findMethod(rawType, prop)
109                    }
110                    else
111                        foundMethod = findMethod(clazz, prop)
112
113                    if (foundMethod) clazz = foundMethod.getGenericReturnType()
114
115                } catch (NullPointerException e) {
116                    println "Error: "+e
117                }
118            }
119            return clazz;
120        }
121 }
```

114

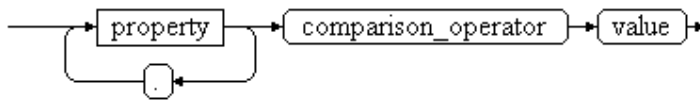# APPENDIX I. A VISUAL REPRESENTATION OF EXPRESSION SYNTAX

expression
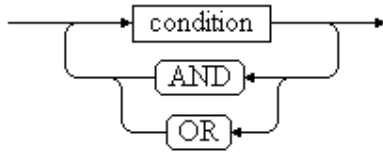
condition
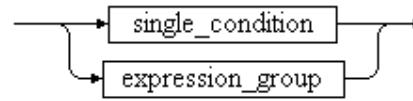
single_condition

expression_group
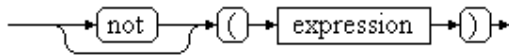
property
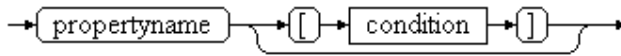
APPENDIX J. EXAMPLE OF AN XML SERIALIZED EXPRESSION

```xml
<expression modelId="org.openmrs">
    <group negated="false">
        <expressionline>
            <properties>
                <property name="age" datatype="numeric"/>
            </properties>
            <comparisonoperator id="less_than"/>
            <value displayValue="5">"5"</value>
        </expressionline>
        <logicaloperator id="and"/>
        <expressionline>
            <properties>
                <property name="age" datatype="numeric"/>
            </properties>
            <comparisonoperator id="greater_or_equal"/>
            <value displayValue="1">"1"</value>
        </expressionline>
        <logicaloperator id="and"/>
        <expressionline>
            <properties>
                <property name="observations" datatype="collection"
enclosed_datatype="org.openmrs.Obs"/>
                <property name="where" datatype="org.openmrs.Obs"
type="listselection">
                    <group negated="false">
                        <expressionline>
                            <properties>
                                <property name="concept"
datatype="org.openmrs.Concept"/>
                            </properties>
                            <comparisonoperator id="concept_equals"/>
                            <value displayValue="HIV POS">"HIV POS"</value>
                        </expressionline>
                    </group>
                </property>
                <property name="valueCoded" datatype="org.openmrs.Concept"/>
            </properties>
            <comparisonoperator id="concept_equals"/>
            <value displayValue="POSITIVE">"POSITIVE"</value>
        </expressionline>
        <logicaloperator id="and"/>
        <expressionline>
            <properties>
                <property name="observations" datatype="collection"
enclosed_datatype="org.openmrs.Obs"/>
                <property name="where" datatype="org.openmrs.Obs"
type="listselection">
                    <group negated="false">
                        <expressionline>
                            <properties>
                                <property name="concept"
datatype="org.openmrs.Concept"/>
                            </properties>
                            <comparisonoperator id="concept_equals"/>
```

```
                                <value displayValue="HOSPITAL DISTANCE">"HOSPITAL
DISTANCE"</value>
                            </expressionline>
                    </group>
                </property>
                <property name="valueNumeric" datatype="numeric"/>
            </properties>
            <comparisonoperator id="less_than"/>
            <value displayValue="15">"15"</value>
        </expressionline>
        <logicaloperator id="and"/>
        <group negated="false">
            <group negated="false">
                <expressionline>
                    <properties>
                        <property name="observations" datatype="collection"
enclosed_datatype="org.openmrs.Obs"/>
                        <property name="where" datatype="org.openmrs.Obs"
type="listselection">
                            <group negated="false">
                                <expressionline>
                                    <properties>
                                        <property name="concept"
datatype="org.openmrs.Concept"/>
                                    </properties>
                                    <comparisonoperator id="concept_equals"/>
                                    <value displayValue="ELISA">"ELISA"</value>
                                </expressionline>
                            </group>
                        </property>
                        <property name="valueCoded"
datatype="org.openmrs.Concept"/>
                    </properties>
                    <comparisonoperator id="concept_equals"/>
                    <value displayValue="POSITIVE">"POSITIVE"</value>
                </expressionline>
                <logicaloperator id="and"/>
                <expressionline>
                    <properties>
                        <property name="birthdate" datatype="date"/>
                        <property name="monthsSince" datatype="numeric"/>
                    </properties>
                    <comparisonoperator id="greater_than"/>
                    <value displayValue="18">"18"</value>
                </expressionline>
            </group>
            <logicaloperator id="or"/>
            <group negated="false">
                <expressionline>
                    <properties>
                        <property name="birthdate" datatype="date"/>
                        <property name="monthsSince" datatype="numeric"/>
                    </properties>
                    <comparisonoperator id="less_than"/>
                    <value displayValue="18">"18"</value>
                </expressionline>
                <logicaloperator id="and"/>
```

117

```xml
                <expressionline>
                    <properties>
                        <property name="observations" datatype="collection"
enclosed_datatype="org.openmrs.Obs"/>
                        <property name="where" datatype="org.openmrs.Obs"
type="listselection">
                            <group negated="false">
                                <expressionline>
                                    <properties>
                                        <property name="concept"
datatype="org.openmrs.Concept"/>
                                    </properties>
                                    <comparisonoperator id="concept_equals"/>
                                    <value displayValue="HIV DNA POLYMERASE CHAIN
REACTION">"HIV DNA POLYMERASE CHAIN
                                        REACTION"
                                    </value>
                                </expressionline>
                            </group>
                        </property>
                        <property name="valueCoded"
datatype="org.openmrs.Concept"/>
                    </properties>
                    <comparisonoperator id="concept_equals"/>
                    <value displayValue="DETECTED">"DETECTED"</value>
                </expressionline>
            </group>
        </group>
    </group>
</expression>
```