# Clinical Decision Support Systems

Steven Vanden Eynde
*Master in Computer Sciences: A.I.*

## 1. Introduction

In today's clinical practice, doctors and medical personnel are confronted with making a variety of decisions when treating a patient. The decisions they make are mainly based on common guidelines and experience in the field, obtained by treating previous patients. However, since any doctor is unable to remember every single patient case, diagnoses, treatments and possibly other useful information is recorded for future reference. This indeed can be a great help, but it requires a lot of time from the doctor if all this information has to be processed manually each time a new patient arrives. Furthermore, both the amount of information and its complexity have increased drastically with more new clinical tests and testing equipment and more medical knowledge. The development of clinical decision support systems (CDSSs) provides a solution for managing this information. Using medical guidelines in combination with the available information from previous patient cases, a CDSS can provide support for diagnosing new patients and preventing or treating a variety of diseases, hereby helping doctors and medical personnel to make quick and correct decisions.

Nowadays, a variety of intelligent computing techniques are already used in clinical decision support systems for medical planning, diagnosis and treatment [18]. These computing techniques mainly occur in the context of knowledge-based systems (KBS), which make use of techniques such as rule-based reasoning (RBR), case-based reasoning (CBR) or model-based reasoning (MBR), or in the context of more data-centered systems, which make use of artificial neural networks (ANN) or genetic algorithms (GA) [12]. A combination of two or more of these techniques can also be applied, often to overcome the disadvantages of individual techniques [8, 11, 10].

However, the CDSSs that are used today still have some issues. Most of the time they are specifically tailored to a specific problem and an existing healthcare system, so they are not good at handling knowledge from different sources. Most importantly, they are also not intuitive to develop and expand as more clinical knowledge becomes available. In this dissertation, we have therefore collaborated with *Agfa HealthCare* to develop a CDSS based on the logical systems IDP and EYE, since we strongly believe that these systems are able to overcome the main disadvantages of existing clinical decision support systems.

## 2. Objectives

In this dissertation, our goal is to contribute to the ongoing research on clinical decision support systems, and more specifically to the research on integrating logical systems in these CDSSs. By developing our own proof of concept CDSS based on the logical systems IDP and EYE, we will evaluate the usage of both systems in the context of a CDSS. Using the methods we followed in constructing our CDSS, we will also provide a detailed guide on how to develop a CDSS and how to integrate logical systems to perform several logical tasks to solve medical problems.

## 3. Logical systems

First we will shortly introduce the two used logical systems, namely IDP and EYE.

### 3.1. IDP

IDP is a system for model expansion under constant development by the *Knowledge Representation and Reasoning* (KRR) research group at the KU Leuven. The system as a whole is built from two underlying subsystems: a *grounder* named GIDL [16] and a *solver* named MINISAT(ID) [9]. Since these subsystems are two of the most performant systems in their respective jobs, the IDP system as a whole becomes one of the most performant systems for model expansion.

The modeling language that is used by IDP to perform its model expansion, is an extension of classical first-order logic (FO), which is referred to as FO($\cdot$) [15, 17]. FO($\cdot$) extends FO with, among others, types, arithmetic, aggregates, partial functions and inductive definitions. A logical vocabulary or theory written in

FO(·) always consists of several parts, called blocks. The blocks that are available for use in FO(·) are the `Declare`, `Given`, `Find`, `Satisfying` and `Data` block. The `Given` block contains that part of the represented vocabulary for which there is an interpretation available in the `Data` block, while the `Data` block can be considered as the input data of the system. The part of the represented vocabulary for which the interpretation is searched for during model expansion, is located in the `Find` block. The found interpretation for the vocabulary in this block is therefore considered as the output of the system. The final part of the vocabulary, represented in the `Declare` block, is an optional intermediate part that is only used to facilitate the expression of sentences in FO(·), so it plays no role in representing in- or output of the system. Finally, the `Satisfying` block represents the theory used by the system, containing logical rules and definitions.

### 3.2. EYE

EYE stands for *Euler YAP Engine* and is a logical inference system with support for proof construction, that is developed and maintained by *Agfa HealthCare* [5, 6]. EYE supports path detection and is mainly used to perform inference tasks in the context of the semantic web. The system is implemented in YAProlog, one of the most performant Prolog systems currently available. YAP thanks its high performance to a technique called *just in time indexing* or *demand driven indexing*, which allows flexible indexing of Prolog clauses [14]. Due to YAP's flexible indexing and high performance, the EYE system in its whole proves to be very scalable and performant in comparison with other similar reasoning engines [7].

The modeling language that is used by EYE to perform inference on, is called N3Logic. N3Logic is a form of coherent logic, but it has the same logical expressiveness as FO, since first-order logic can be linearly translated to coherent logic [3]. N3Logic makes use of N3 syntax, a compact and readable alternative for RDF [13], which also expands RDF with predicates [1]. With a minimal extension to the RDF data model, N3Logic attempts to create a language for the representation of both logic and data, ideally used in a semantic web context [2]. In N3Logic, knowledge is represented as triples, which consist of a subject, a predicate and an object. Subject, predicate and object are always represented as URI's (*Uniform Resource Identifiers*). This way, reasoning over logically interconnected data spread throughout the web becomes feasible, and existing knowledge can easily be integrated in new applications.

## 4. Development of a CDSS

Using the two previously presented logical systems, we have built a proof of concept clinical decision support system. This CDSS can be practically used for prediction and prevention of chronic diseases. The developed CDSS is structured as a client-server architecture, and is therefore available as a browser application, so it can be accessed from anywhere on the web. Since maintenance can be performed only on the server, this architecture makes sure that the system is easily maintained and every user has direct access to the latest version of the system. The developed CDSS and its accompanying source code are freely available at `https://dl.dropbox.com/u/3912107/MasterproefStevenVandenEynde.zip`.

### 4.1. Functionality overview

The architecture used in the development of the CDSS is shown in figure 1. Practically, the developed CDSS first offers the possibility to enter some properties of the current patient on the web page. This required data consists of the patient's gender, age, weight, height and smoking behaviour. Afterwards the user can ask the system to evaluate the current patient's wellness, based on his or her properties, by clicking a button in the user interface. The system then calculates risk factors for several chronic diseases, based on properties and diagnoses of previous patient cases. In this CDSS, risk factors for getting diabetes, hypertension, strokes and chronic heart diseases are calculated and shown in the user interface. Also, since most of these diseases are caused by the patient being overweight, the system suggests sports that can be practiced by the patient to lower his or her weight and become healthier. The suggested sports are also dependent on the patient's properties, and are displayed in the user interface as an additional interactive sport selection functionality. In addition to the suggested sports, there are some selectable sport characteristics displayed. When a certain sport characteristic is selected, all sports that answer to this characteristic become selectable. Eventually, after some of the suggested sports are selected, the user can ask the system to propose a sport schedule, based on the selected sports and the patient's properties, again by clicking a button. This sport schedule is then shown in the user interface, and indicates how many hours per week the patient should practice a suggested sport to lose a predefined amount of weight.

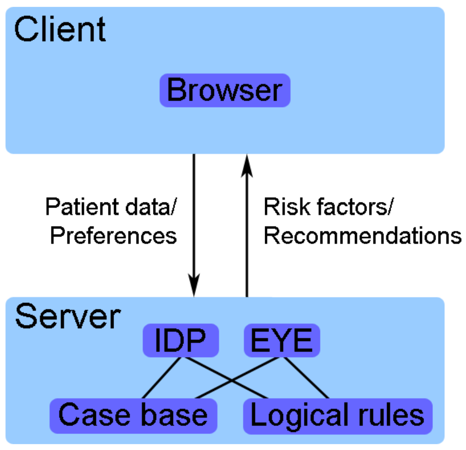

**Figure 1:** Client-server architecture of the developed CDSS

## 4.2. Technology overview

To make the developed web page interactive and dynamic, we have used the JavaScript libraries jQuery [4] and Knockout (`http://knockoutjs.com/`). jQuery is mainly used to easily communicate with the server via *Asynchronous JavaScript and XML* (AJAX), an asynchronous communication technique. During client-server communication, the user interface is also kept consistent by disabling certain buttons and showing loading icons through jQuery. While jQuery mainly offers low-level functions to make JavaScript operations easier, Knockout is more of a high-level library that implements the *Model-View-View Model* (MVVM) design pattern, which makes it possible to create dynamical web pages that are based on an underlying JavaScript data model. Thanks to Knockout, risk factors and suggested sports and sport characteristics are dynamically loaded into the web page, thereby making it unnecessary to make changes to the web page itself when different diseases, sports or sport characteristics are processed and displayed.

To make the server capable of managing several logical processes, we have used a Tomcat application server in combination with Java Servlet technology, which allows easy integration of the required logical systems through the use of Java as a programming language. The data coming from the client web page is therefore processed by the server using Java and passed to either IDP or EYE. The output of the logical systems is again processed using Java and sent back to the client application. The EYE logical system is easily integrated in the system using the available EYE Java API. This API acts as a Java wrapper that takes care of executing EYE in a separate process and collecting the output of the EYE process. However, for IDP, there is no such API available, which is why we developed an IDP Java class hierarchy to support simple execution of a separate IDP process, tailored to the process's needs. The top class in this class hierarchy takes care of the generic processes to start and end IDP, while subclasses are added to process the generated output differently by storing this output to local attributes. This way, the output of any IDP process can easily be gathered on the server by inspecting the attributes of the accompanying object.

## 4.3. Logical systems operation

The logical systems IDP and EYE are used in the developed CDSS to calculate risk factors, suggest sports and propose a sport schedule. These logical tasks all require different approaches and methodologies, but can nevertheless all be handled by one and the same logical system.

The calculation of risk factors is done by comparing the current patient with previous patient cases in a case base, thus making use of case-based reasoning. Practically, this is done by first selecting all the cases that are similar enough to the current patient. Within these similar cases, all cases that have a certain chronic disease are counted, and the ratio of this number to the total number of similar cases represents the risk factor of the specified chronic disease. In IDP, this is done by making extensive use of inductive definitions. All cases and their properties and diagnoses are hereby given as data in the IDP `Data` block, while all cases that are similar to the current patient are gathered in a relation that is specified by a definition in the `Satisfying` block. This relation is specified by the `Find` block, so the output of the IDP process will contain all cases that are similar to the current patient. In a similar way, relations and definitions are specified for gathering similar cases that are diagnosed with a certain chronic disease. When the cases that meet these different relations are all represented in the output of the IDP process, it is simply a matter of counting the amount of cases for the different relations to obtain the different risk factors. In EYE, there are two different methodologies to perform case-based reasoning. The first, which we call the findall-method, acts the same as the IDP process by gathering and counting all cases that are similar to the current patient and all cases within these similar cases that have certain chronic diseases. In EYE, this is done by logical rules instead of definitions, which takes a bit more time and space to fully specify. The second method is an EYE-specific method to perform case-based reasoning by using a *monotonic abduction-*

*deduction-induction cycle* (MONADIC), and is therefore called the MONADIC-method. This method generates a minimal model for each case that is present in the case base, comparing this case to the current patient, and then subdividing these models in different classes, depending on the fact that they are similar to the current patient or have a certain chronic disease or not. Afterwards, the amount of models in the different classes are automatically counted and the ratios of these amounts are returned. One of these ratios then represents the risk factor of the concerned chronic disease. While the MONADIC-method provides a structured and unambiguous way of performing case-based reasoning in EYE, it has the disadvantage that only one risk factor can be calculated per run, since the system can only subdivide models conform to one chronic disease at a time.

The second logical task in the CDSS, suggesting sports, requires rule-based reasoning, since the reasons why certain sports should be suggested or not are contained in logical rules which are based on the patient's weight, age and chronic disease risk and the sport intensity. Not only do the logical systems have to suggest sports, they also need to make sure the user interface keeps its consistency regarding which sports are selectable or not when certain sport characteristics are selected. This can be done using the same logical process for each of the functionalities, by including an operational part in the used logical vocabularies and theories. In IDP, inductive definitions are used to gather all sports that should be suggested, all sport characteristics that should be suggested, all sports that can be selected and all sports that are already selected. In EYE, exactly the same is done, but by making use of logical rules instead of definitions.

Proposing a sport schedule, finally, is also done using rule-based reasoning. To determine the proposed amount of hours per week to practice each selected sport, the weight of the patient, the projected weight loss and the intensity of the sport are taken into account. This is again done by an inductive definition in IDP, and some logical rules in EYE. Interesting to notice is that nearly all information about sports that is used in the logical task of suggesting sports, is reused in this logical task by both logical systems. The potential of data reuse is an important property of a system in a CDSS, because clinical data and rules are often widespread and should not be redefined with every new application.

## 5. Comparison between IDP and EYE

By constructing a CDSS based on IDP and EYE, it is already clear that both systems have their advantages and disadvantages. Here we will give a summarising overview of how IDP and EYE compare to each other with respect to their usability in a CDSS.

When a logical task requires the calculation or manipulation of numerical quantities or properties, IDP has a serious disadvantage, since it can only perform calculations on integers. Furthermore, the definition of an integer type in IDP also requires the definition of an accompanying domain. This domain has to be chosen carefully, since larger domains lead to larger execution times. Chosing the range of an integer domain is therefore a complicated task. In EYE, however, real numbers can be used in all sorts of numerical calculations without predefining domains.

A second disadvantage of IDP is that it has its own specific modeling language, which is not standardised. It is therefore difficult to migrate clinical knowledge that is available in other systems into IDP. Also, to run IDP, all logical knowledge should be contained in only one file that is formatted in FO($\cdot$) and should be stored locally. Since medical knowledge is often widespread, it will be necessary to gather all necessary knowledge and convert it to one file before running IDP. EYE, on the contrary, makes use of the modeling language N3Logic that is based on the standardised N3 syntax. Since knowledge formatted in N3 is widely spread on the semantic web, and EYE can access multiple files on the entire web as input, it can reason over all knowledge that is available through the semantic web. It is important, however, to take into consideration that not all knowledge on the semantic web is necessarily correct, so in the context of a CDSS, only medical knowledge from trusted sources should be used.

An advantage of EYE is that it supports different types of output, depending on the user's preferences. This output can even be defined as regular text, HTML or other formatting, which could be useful for the logical construction of web pages for example. IDP doesn't support this feature and has only one form of output, namely the interpretation of all predicates, functions and constants that are defined in the logical `Find` block.

The biggest advantage of IDP is that its modeling language is very intuitive to use. Inductive definitions are often the most natural way of representing knowledge, and when it is not, expressing knowledge in FO($\cdot$) is still straightforward. Due to the use of the standardised N3 syntax, N3Logic as a modeling language is more complex than FO($\cdot$). N3Logic makes use of logical rules instead of definitions, which makes knowledge representations bulky and harder to maintain compared to FO(·).

Lastly, to compare the performance of the two logical systems, we have conducted some performance tests. In these tests, both systems are asked to perform the same case-based reasoning task as is required within the CDSS, with a varying number of patient cases available. The tests were all repeated 10 times, and the test results are presented in tables 1 and 2. From these tests we can conclude that EYE is the most efficient when processing a small or average amount of data and logical rules thanks to a very small startup time. However, when processing large amounts of data and logical rules, IDP becomes the more efficient system due to its very efficient model expansion. Also notice that the runtime of both systems only rises linearly with the amount of cases, which means that both IDP and EYE are performant systems that are scalable to larger clinical decision support systems.

**Table 1:** Mean runtime and standard deviation (in ms) of IDP

| # cases | IDP |
|---:|---:|
| 10 | 1082(±210) |
| 100 | 1053(±197) |
| 1000 | 1039(±199) |
| 10000 | 1438(±203) |
| 100000 | 5540(±299) |

**Table 2:** Mean runtime and standard deviation (in ms) of EYE

| # cases | EYE (monadic) | EYE (findall) |
|---:|---:|---:|
| 10 | 324(±4) | 308(±8) |
| 100 | 419(±24) | 341(±4) |
| 1000 | 1254(±18) | 703(±6) |
| 10000 | 9730(±74) | 4282(±63) |
| 100000 | 126336(±415) | 55424(±2092) |

## 6. Conclusions and future work

In this dissertation we have developed a useful clinical decision support system based on the logical systems IDP and EYE. With the development of this CDSS, we showed that both IDP and EYE can practically be used to solve all kinds of medical problems by performing logical tasks in a CDSS, and we provided a detailed guide on how to integrate these logical systems in a CDSS. We have also uncovered and discussed both the advantages and disadvantages of the two logical systems. This dissertation is only a first step in the integration of IDP and EYE in clinical decision support systems, but we believe that both systems may prove to be a valuable addition to CDSSs.

To make the integration of logical systems in a CDSS even more valuable, further research on performance and other improvements of both IDP and EYE is definitely worthwhile. Also, the existing CDSS could be easily expanded to include even more clinical decision support tasks, which would make the system even more usable.

## References

[1] T. Berners-Lee & D. Connolly (2011). Notation3 (N3): A readable RDF syntax. `http://www.w3.org/TeamSubmission/n3/`.

[2] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf & J. Hendler (2008). N3logic: A logical framework for the world wide web. *Theory and Practice of Logic Programming*, 8(3):249–269.

[3] M. Bezem & T. Coquand (2005). Automating coherent logic. In G. Sutcliffe & A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of *Lecture Notes in Computer Science*, pp. 246–260. URL `http://dx.doi.org/10.1007/11591191_18`.

[4] B. Bibeault & Y. Katz (2010). *jQuery in Action, Second Edition*. 2nd edition.

[5] J. De Roo (2011). Euler YAP Engine, a birds EYE view. `http://www.agfa.com/w3c/Talks/2011/01swig/`.

[6] J. De Roo (2011). EYE Note. `http://eulersharp.sourceforge.net/2003/03swap/eye-note.txt`.

[7] J. De Roo (2012). Euler Proof Mechanism. `http://eulersharp.sourceforge.net/`.

[8] K. Kumar, Y. Singh & S. Sanyal (2009). Hybrid approach using case-based reasoning and rule-based reasoning for domain independent clinical decision support in ICU. *Expert Systems with Applications*, 36(1):65–71.

[9] M. Mariën, J. Wittocx, M. Denecker & M. Bruynooghe (2008). Sat(id): Satisfiability of propositional logic extended with inductive

definitions. In H. Kleine Büning & X. Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008*, volume 4996 of *Lecture Notes in Computer Science*, pp. 211–224.

[10] S. Montani, P. Magni, R. Bellazzi, C. Larizza, A. V. Roudsari & E. R. Carson (2003). Integrating model-based decision support in a multi-modal reasoning system for managing type 1 diabetic patients. *Artificial Intelligence in Medicine*, 29(1-2):131–151.

[11] E. Ocampoa, M. Maceirasa, S. Herrerab, C. Maurentea, D. Rodríguezc & M. A. Sicilia (2011). Comparing Bayesian inference and case-based reasoning as support techniques in the diagnosis of Acute Bacterial Meningitis. *Expert Systems with Applications: An International Journal*, 38(8):10343–10354.

[12] B. Pandey & R. B. Mishra (2009). Knowledge and intelligent computing system in medicine. *Computers in Biology and Medicine*, 39(3):215–230.

[13] RDF Working Group (2004). Resource Description Framework (RDF). `http://www.w3.org/RDF/`.

[14] V. Santos Costa, K. Sagonas & R. Lopes (2007). Demand-driven indexing of prolog clauses. In V. Dahl & I. Niemelä, editors, *Logic Programming*, volume 4670 of *Lecture Notes in Computer Science*, pp. 395–409. URL `http://user.it.uu.se/~kostis/Papers/iclp07.pdf`.

[15] J. Wittocx, M. Mariën & S. De Pooter (2010). *The IDP system*. `http://dtai.cs.kuleuven.be/krr/files/bib/manuals/IDP-manual.pdf`.

[16] J. Wittocx, M. Mariën & M. Denecker (2008). GidL: A grounder for FO+. In M. Thielscher & M. Pagnucco, editors, *Proceedings of the 12th International Workshop on Non-Monotonic Reasoning*, pp. 189–198.

[17] J. Wittocx, M. Mariën & M. Denecker (2008). The IDP system: A model expansion system for an extension of classical logic. In M. Denecker, editor, *Proceedings of the 2nd Workshop on Logic and Search*, pp. 153–165.

[18] A. Wright & D. F. Sittig (2008). A four-phase model of the evolution of clinical decision support architectures. *International Journal of Medical Informatics*, 77(10):641–649.