An HTML 5–Based Graphical User Interface and a Transformation for Tennis Data in XML

Daniel Weidner and Dietmar Seipel

Abstract: XML is widely recognized as a prevalent and extensively employed data format for representing semi–structured, complex and nested data. In earlier work, we had used transformation grammars based on the field notation of our logic programming tool FNQUERY for querying, transforming, and updating XML documents. However, some aspects could not be addressed in this transformation.

In the present paper, we present an updated transformation approach data based on the graph notation grammars of FNQUERY to generate identifiers and establish a nesting logic of points and hits based on timestamps. We can overcome the limitations of previous field notation grammars, while ensuring the consistent and precise usage of coordinates.

In addition, we have built a web–based user interface for the tennis tool using HTML 5 and Prolog's definite clause grammars (DCGs). We will elucidate the motivations behind this transition and outline our approach for defining DCGs in the context of the new tool.

Keywords: Prolog; XML Processing; Field Notation Grammars; Graph Notation Grammars; HTML 5; Definite Clause Grammars; Web Application

1 Introduction

Markup languages, such as XML and HTML, are essential for structuring and formatting content in various contexts. The universal data format XML allows for the storage and exchange of structured information with its flexible syntax and customizable tags. It is commonly used for data storage, configuration files, and web services. HTML is specifically designed for web pages. It focuses on the presentation and layout of content within browsers, using predefined tags to defining structure and formatting for elements. Both XML and HTML serve distinct purposes, with XML emphasizing data representation and interoperability, while HTML concentrates on visually rendering web pages for user–friendly browsing.[1]

In this work, we work with both languages to present an updated version of our previous work on the XML transformation of tennis data, and to create a new graphical user interface for the tennis tool in HTML 5. Starting with a convolutional neural network (CNN), data is generated and stored in an XML file. However the data is unclean and need pre–processing, before we can use them for visualization and further data analysis, like it has already been done in [We03].

[1] This paragraph was originally drafted by ChatGPT.

The *domain–specific, declarative tool* FnQuery/Pl4Xml was developed for *querying, transforming and updating* Xml data, cf. [Di02]. It contains two Prolog document object models: the field notation (FN) and the graph notation (GN), which represent Xml data as terms and facts in Prolog, respectively, and languages for querying, transforming and updating Xml data. In the prior research, we exclusively used field notation grammars, which proved effective for an initial approach, but had limitations in achieving certain effects. In contrast, this paper specializes in graph notation and details their advantages. I.e. we build new grammars with this notation to generate identifiers and create a nesting logic of points and hits based on timestamps. Additionally, we ensured the continuous and accurate usage of coordinates throughout the process.

Furthermore, we have implemented a new *graphical user interface* of the tennis tool based on Html 5 using Definite Clause Grammars (DCGs). We will elucidate the motivations behind this change and provide an explanation of our approach in defining DCGs to achieve the desired functionality.

The rest of this paper is structured as fallows: Section 2 summarizes our previous work on transforming tennis data in Xml with field notation grammars, and Section 3 presents the context of field and graph notation grammars. Section 4 demonstrates the usage of both FN and GN to represent the tennis data. In Section 5, we show the new web–based, graphical user interface for the tennis tool, build with definite clause grammars (DCGs), created from a first Html 5, CSS and JavaScript approach. Finally, Section 6 concludes with a summary.

2 Current Status of the Tennis Tool

Our current version of the tennis tool is packed in two Docker containers: the *Convolutional Neural Network (CNN)* Tennis Recognition and a dockerized version of Prolog–based *logic programming* tool Declare [Ba19; We21]. These containers will interact through a shared volume, with Declare accessing the Xml file generated by the CNN [Ba19; Pr17].

Declare [Se94; Se97] – originally called DisLog – is a *declarative logic programming* tool implemented in Prolog, which has been used in many previous publications since then. The non–declarative features of Prolog have been used for the internal implementation of Declare, but the external features of Declare are declarative. Declare uses Field and Graph Notation Grammars to transform the unstructured Xml into a clean and structured XML format. The resulting clean file will then be visualized in a web browser using a Prolog web application. Additionally, we intend to incorporate data mining capabilities within the browser, similar to our previous work in [We19].

To provide access to the original videos, we have two approaches. Firstly, tennis experts can use their local videos (e.g., in mp4 format) by having Declare read and list them. Secondly, we aim to enable other users to share videos through a cloud–based solution. The architectural design of this system is depicted in Figure 1.

The presented study builds upon our prior research conducted in [We22]. This paper serves as a continuation of our previous work with significant updates; we will provide a concise summary of the essential aspects covered in [We22] in the following section. In that paper, we discussed the process of transforming tennis data stored in XML format using field notation grammars (FNGs). We addressed the need for cleaning and structuring XML files, generated by a convolutional neural network (CNN), for analysis. This is because the automatically generated XML files are found to be unsuitable.

The coordinates are recorded from a camera behind the court, rather than from a bird's–eye perspective. Additionally, not all frames of the video are interesting for the further process. Therefore, only important frames should be filtered. To address these issues, we presented a process of XML transformation based on field notation grammars. We described how we transformed the unclean data obtained from the CNN into structured, clean data that can be used for further processing such as data mining. The transformation involves using FNGs and solving linear equations to project `x`– and `y`–coordinates into the plane.

In the specific context of tennis data, we described the architecture of the CNN-generated XML file. The file that represents a tennis match is structured in sets, games, points, and hits. It contains information about player positions, ball trajectories, and court coordinates. We explained how the XML file is structured based on the CNN's analysis of the video.

The transformation process involves renaming rally tags to point tags, as each rally represents a single point. Frames with the event value `contact` are identified as containing hits, and the start and end of rallies are determined.

3 XML Objects

The *Extensible Markup Language* XML is a well–known standard data format for representing and exchanging semi–structured data and knowledge. Prolog can handle complex tree–structured objects nicely, and has an easy to use meta–programming property. Therefore Prolog is very helpful for processing XML [Ab00].

In [Di02; Se07; Se15; Se18], the declarative logic programming tool FNQUERY/PL4XML was developed for *querying*, *transforming* and *updating* XML data. Two alternative representations for XML have been introduced, the field notation and the graph notation, and the same transformation languages are applicable for transforming and updating both representations.

In our prior research [We22], we had focused on field notation; the current paper shifts the emphasis towards an in–depth exploration of graph notation. The field notation was useful for filtering and renaming XML elements. However, for the complete tennis data transformation it is important to establish a link between different XML elements. In this section, we will first summarize the most important information of the Field Notation and then introduce the Graph Notation. We will discuss the differences to the Field Notation and the advantages of the Graph Notation.

3.1 XML Objects in Field Notation

Field Notation (FN) is a generic Document Object Model (DOM) for representing intricate objects in XML. FN encapsulates objects as Prolog triples `T:As:C` comprising a tag name `T`, attribute/value pairs `As`, and sub-elements `C`, named content. This hierarchical representation allows for the seamless navigation in and the manipulation of XML data.

To transform XML data in field notation, PL4XML introduces Field Notation Grammars (FNGs). FNGs employ rules defined by the binary, infix–predicate `--->/2`, with the left hand side representing the input FN–triple or XML file, and the right hand side representing the transformed output. By sequentially applying FNG rules, XML data can be efficiently modified, enriched, or restructured to suit specific requirements.

3.2 XML Objects in Graph Notation

Conversely, by storing XML objects as Prolog facts through an object/relational mapping, FNQUERY is capable to implement the backward axes of XQuery [Ch03; Wa07]. We are able to define the graph notation as a relational representation of XML, based on the two relations `ref/3` and `val/2`.

The graph notation database can store many XML elements at the same time. In this approach, unique identifiers are used to refer to elements I and their descendant elements J. The notation `father(`J`)` represents the parent of J within I, `tag(`J`)` denotes the tag associated with J and `text(`J`)` refers to the textual content of a text element J. With this we have

- I is mapped to a fact `reference(`I^*`,tag(`I`),`I`,`N`)`, where I^* is an identifier that is not the id of any other element in the database.
- A non–text element J is mapped to a Prolog fact `reference(father(`J`),tag(`J`),`J`,`N`)`.
- For every attribute/value–pair $A : V$ of J we get a fact `attribute(`J`, `A`, `V`)`.
- A text element J is mapped to a fact `value(father(`J`),`J`,`N`)`.

The order of the sub–elements K of a non–text element J is reflected by the numbers N in the facts `reference(`J`,_,`K`,`N`)` and `value(`J`,`$text(K)$`,`N`)`, which must all be different. Consequently, the XML elements stored within the Graph Notation database can be reconstructed.

Interaction between Graph and Field Notation

For the transition to the Graph Notation, it is possible to switch by assigning the value `gn` of the Declare variable `fn_mode`. An FN–triple in field notation can be stored into the GN database using the predicate `fn_to_gn/2`. This will return an identifier, that refers to the stored FN–triple. It is also possible to reconstruct the stored FN–triple using the inverse predicate `gn_to_fn/2`. The items in the graph notation database, referenced by `Id` can be loaded and saved from and to a file `F`, respectively, using the calls `dread(xml, F, [Id])` and `dwrite(xml, F, Id)`. For this the `fn_mode` has to be `gn`.

Advantages of the Graph Notation Database

The main advantage of Graph Notation are the additional axes in location steps, which we could not and cannot use in Field Notation. If `T` is any Prolog term then it is possible to select several elements of the XML file. For example with `preceding::T`, we can select an element with the tag `T`, that is preceding the considered FN–triple in the document order, except the ancestors. By making this switch, specifically in the context of tennis data, we gain the ability to select the current court element when a frame is deemed interesting. This implies that we need to transform the coordinates using the preceding court coordinates.

Comparison of Field and Graph Notation

Considering the advantages in *expressibility* offered by the Graph Notation over the Field Notation, one might question the necessity of using Field Notation at all. This question can be addressed by considering two key points. Firstly, the usage of Graph Notation Databases requires the storage of the facts in a deductive database, which can lead to significant *storage* requirements, particularly when dealing with large XML files. Secondly, as expected, the transformation process proved to be *faster in Field Notation* mode, since the navigation to XML sub–elements is faster in a term than in the Prolog database. For this, we conducted transformations on several XML files, using simple grammars that functioned equivalently for both Field Notation and Graph Notation. In Listing 2 we dropped all operations using backward axes, such as, e.g., `preceding`; we maintained consistent court coordinates and avoided iterations and nesting. This allowed us to compare the transformations by initiating the same conversion process using both FN and GN modes. We categorized the videos into three types: Short videos (highlight reels), medium–length videos (extended highlights), and long videos (full matches). The results of these comparisons are presented in Table 1. Often, the operations were about 15 times faster on FN than on GN.

Length of Video (Type)	FN Time [sec.]	GN Time [sec.]
1–5 min (Short Highlight Videos)	0.02 – 0.1	1 – 2
10–20 min (Extended Highlight Videos)	1 – 2	10 – 20
90–180 min (Full Match)	5 – 10	60 – 150

Table 1: Rune Time Comparison of FN and GN Mode for Different Video Lengths

4 Transforming Tennis Data Represented in XML

The XML transformations of our previous work [We22] had been using the field notation for XML data. In this paper, we combine them with the graph notation of XML to obtain additional transformations. In this section, we show four updates, which we did in our work.

1. To work with the current court coordinates, that are recognized, we want to add an identification number. We therefore iterate court ids, which are not stored in the first place. This can not be done with FNG's because siblings need to be calculated.
2. We use these id's to get the current court coordinates for calculating the ball trajectories into the plane.
3. We want to calculate whether a game is finished or not, to get an automated nesting of games (and in the future sets). For this, we check the times between the last hit of a rally and the serve of the next rally. If there is a time delay of more than 15 seconds, then there is a new game. In the future, we want to add scores, to calculate if a game (and set) has ended, because times are imprecise, especially if the video is cut.
4. Hits, Points, Games and Sets are initially identified by the frame number. As for the court iteration, we iterate all sets, games, points and hits.

4.1 Adding Court Identifiers

The CNN recognizes court coordinates. This recognition is done at the beginning and every frame it is checked whether the coordinates differ, i.e. there was a camera movement. If this is the case, then the CNN stores the new coordinates. We want to have a good access to the court tags; especially, we want to know between which frames a court recognition appeared. But since the CNN only stores the coordinates, we define a graph notation grammar to generate court id's, without changing the CNN storage implementation. This iteration grammar can be seen in Listing 1

```
Court ---> Court :-
   fn_item_parse(Court, court:_:_),
   findall( ID,
      ( Court2 := Court/preceding::court,
        Ida := Court2@id,
        atom_to_number(Ida, ID) ),
      Tuples ),
   ( Tuples = [] -> Id = '1'
   ; max(Tuples, ID_minus_one), Id is ID_minus_one+1 ),
   Court := Court*[@id:Id].
```

Listing 1: Create Court ID's

Here, for a court tag we search all preceding tags with also the name court. If this list is empty, we found the first court tag, so it gets the id 1. Otherwise, we take the maximal Id that we have found and add 1. Note, that `Court*[@id:Id]` adds the attribute value pair `id:Id` to the elements of `court`, with `Id` being set above.

4.2 Project Coordinates into the Plane Using Court Coordinates

Next, we aim to utilize the court tags and the stored coordinates to project ball coordinates onto the plane for each relevant frame. Relevant frames are those with value `rally_start/_end` and `contact` for the attribute `event`. In a preceding step, field notation grammars were applied to frames for renaming them as hits. The grammar which executes the coordinate transformation can be found in Listing 2.

Here, line 2 normalizes the fact and the lines 3 and 4 get time and event, if relevant. The lines 5 and 6 get x– and y–coordinates for the ball and the x–coordinates of the players. By combining the latter information with the players' handedness and the y–coordinate of the ball, it is possible to determine whether a hit is classified as either a forehand or a backhand by the top (y–coordinate of ball > 0) or bottom player (y–coordinate of ball < 0), respectively. This can be seen in the lines 27-31. We search for the maximal court id appearing in the XML–file before the current hit in the lines 8-12. We double–check this id with the lines 13-17. Then, the mathematical projection is done from line 17 to 26. Finally, the computed information is stored and returned as New Hit, see the lines 32 and 33.

```
 1  Hit ---> New_Hit :-
 2      fn_item_parse(Hit, hit:_:_),
 3      Time := Hit@time, Event := Hit@event,
 4      member(Event, [rally_start,rally_end,contact]),
 5      XPosBall  := Hit@xPosBall,  YPosBall  := Hit@yPosBall,
 6      XPosTopPl := Hit@xPosTopPl, XPosBotPl := Hit@xPosBotPl,
 7      Source_Point = point(XPosBall,YPosBall),
 8      hit_to_court(Hit, Court),
 9      source_to_target_via_court(Source_Point, Court, Target_Point),
10      Target_point = point(XPos,YPos),
11      atom_to_number(YPos, YPos_Number),
12      ( YPos_Number < 0 -> XPosPl = XPosBotPl
13      ; XPosPl = XPosTopPl ),
14      Handedness = right,
15      determine_hand(XPosBall, XPosPl, Handedness, Hand),
16      As = [id:'1', hand:Hand, type:ground, time:Time, x:XPos, y:YPos],
17      fn_to_gn(hit:As:[], New_Hit).
18
19  hit_to_court(Hit, Court) :-
20      findall( Courtid,
21         ( Courtida := Hit/preceding::court@id,
22           atom_to_number(Courtida, Courtid) ),
23         Courts ),
24      max(Courts, MaxCourt),
    Match := Hit/ancestor::match,
    Court := Match/descendent::court,
    RealMaxCourt := Court@id,
    atom_to_number(RealMaxCourt, MaxCourt).

source_to_target_via_court(Source_Point, Court, Target_Point) :-
    XLT := Court@xLT, YLT := Court@yLT,
    XLB := Court@xLB, YLB := Court@yLB,
    XRT := Court@xRT, YRT := Court@yRT,
    XRB := Court@xRB, YRB := Court@yRB,
    LT = point(XLT, YLT), LB = point(XLB, YLB),
    RT = point(XRT, YRT), RB = point(XRB, YRB),
    vanish_point(LT, LB, RT, RB, Vanish),
    view_point(LB, RB, Vanish, View),
    project_point(Source_Point, LB, RB, Vanish, View, Target_Point).
```

Listing 2: Transformation of Ball Coordinates

The explanation of the mathematical projection using `vanish_point/5`, `view_point/4` and `project_point/6` can be found in [We22].

4.3 Generating the Game Nesting

Next we want to nest the points by games and later sets. So far, the rallies appear sequentially one after the other. But, we want to group them by the corresponding game and set. For this we use the times stored in all frames. Since there is a break between two games, we notice a time gap between them. We use this larger time difference to close one tag with name `game` and open a new one. With this first approach, we generate a semi–plausible nesting of points inside games, see Listing 3. In the future, our intention is to use logic programming to compute a score. Using this technique, we aim to ascertain whether a game or a set has been completed.

```
...
   <game id='1' ...>
      <point id='1' ...>
         <hit id='1' .../> ... <hit id='1' .../> </point>
      <point id='2'> ... </point>
      ...
   </game>
   <game id='2'>
   ...
```

Listing 3: Nesting of Points inside Games

4.4 Adding Set, Game and Point Id's and Final XML–File

Like we did in Section 4.1, we add Id's for all Sets, Games and Points. The first set of the final XML–file then looks like in Listing 4.

```
% XML and Match Prefix
<set id="1" video_file="samprasagassiset1.avi">
 <game id="1" service="A" score_A="0" score_B="0">
  <point id="1" top="B" error="0">
   <hit id="1" hand="fore" type="ground" time="00:00:42" x="0.17" y="-9.07"/>
   <hit id="2" hand="back" type="ground" time="00:00:44" x="-0.49" y="5.89"/>
   ...
  </point>
  ...
 </game>
 ...
</set
```

Listing 4: Final XML–File

5 Updating the Graphical User Interface of the Tennis tool

This section will elaborate on the reasons for the transition from XPCE to HTML. Furthermore, we will provide an explanation of our implementation approach using Definite Clause Grammars (DCGs) for incorporating HTML functionality.

5.1 Using Docker, HTML, and DCGs

The tennis tool, which is integrated into the toolkit Declare, has been using the GUI library XPCE for visualizing the transformed tennis data. However, the graphical application plug–in developed in Prolog using XPCE poses challenges for external accessibility to the tennis tool. Furthermore, XPCE is incompatible with certain Linux versions that solely support Wayland instead of X11, which limits the functionality of the tennis tool. Consequently, there is a possibility of the tennis tool not functioning in certain Linux distributions.

Hence, our initial approach was to develop a website using HTML5, CSS, and JavaScript, considering that Declare already includes HTML objects that incorporate Prolog commands executed through XPCE. However, the challenge with this approach arises from our aim to eliminate XPCE, as the same HTML objects are incompatible with web browsers, since the Prolog commands cannot be executed.

Definite Clause Grammars (DCGs)

Nonetheless, we were able to leverage the basic framework consisting of HTML5 CSS and JavaScript following [Og23]. By adopting Definite Clause Grammars (DCGs), we successfully developed a web application that fulfilled our requirements. ChatGPT describes DCGs as follows: Definite Clause Grammars (DCGs) are a formalism used in the field of computational linguistics and natural language processing for describing the syntax of natural languages[Ab00; Co78; Sh86]. They are a type of logic programming, specifically Prolog, that allows the specification of context–free grammars in a concise and readable manner.

DCGs are defined by a set of rules, known as definite clauses, which define the structure of sentences in a language. Each definite clause consists of a head and a body, separated by the symbol `:-` (read as *if* or *if and only if*). The head represents a non–terminal symbol or a syntactic category, while the body specifies a sequence of terminals (words) and non–terminals (other syntactic categories) that can follow the head. The following example of a simple DCG rule describes a basic English sentence:

```
sentence --> noun_phrase, verb_phrase.
```

In this rule, sentence is the head representing a complete sentence, while noun_phrase and verb_phrase are non–terminals representing a noun phrase and a verb phrase, respectively. The comma (`,`) indicates that the noun phrase must be followed by a verb phrase in order to form a valid sentence. DCGs also allow for additional features, such as constraints and semantic annotations, to enhance the expressiveness and precision of the grammar.

One of the key advantages of DCGs is their ability to generate parse trees or parse sentences using bottom–up parsing techniques. By employing techniques like leftmost derivation and backtracking, DCGs can provide detailed syntactic analyses of sentences based on the defined grammar rules. Overall, Definite Clause Grammars offer a powerful and flexible approach for modeling the syntax of natural languages, and they are widely used in various natural language processing tasks, including parsing, language generation, and machine translation [Me03; No19].[2]

In our current approach, we employ this formalism to initially express HTML code as DCGs and subsequently utilize URI, variables, and additional Prolog code to enhance the web application's executability, making it on par with the original tennis tool.

[2] This paragraph was originally drafted by ChatGPT and has been extended with references by us.

5.2 DCGs for the Tennis Tool Web Application

In the following subsection, we will illustrate our process of defining the DCGs for the web pages of the tennis tool, beginning with the representation of HTML elements. For this consider the following HTML part:

```
<tag>
   <subtag1 x="a">This is a subtag</subtag>
   <subtag2>
      <subsubtag>This is a subsubtag</subsubtag>
   </subtag2>
</tag>
```

To define the structure of the DCGs, we establish the `tag` as the head of the DCG. The body of the DCG consists of the unary predicate `html`. Within `html`, we utilize a list to represent the content of the `tag`. Each element in the list follows a specific structure: a binary function symbol representing the subtag's name, with the first argument being a list of attribute/value pairs for the subtag. The second argument can either be a string if the content of the HTML element is solely that string or a reference to another DCG (see `subtag2`) if the content is multi–part. Then, the preceding HTML part as DCG's looks as follows:

```
tag -->
   html([
      subtag1([x="a"], ("This is a subtag")),
      subtag2([], \subtag2) ])

subtag2 -->
   html([
      subsubtag([], ("This is a subsubtag")) ])
```

Instead of defining another DCG, one can also use a list of subelements. However, this nesting quickly becomes confusing. For a naive parser, it is easier to distinguish only between the two cases from above. Furthermore, one can also implement Prolog code within the DCG body.

By integrating variables into our DCGs, we gain the ability to define them in a more concise manner. This advancement gives us greater control over individual HTML elements using the commands of the Tennis Tool. In certain instances, it proves advantageous to employ URIs for managing additional HTML subpages, reducing the need for unnecessary web controls with JavaScript. Depending on the specific application, incorporating JavaScript may still be beneficial, although for the tennis tool this decision was made on a per–command basis. The URI control is facilitated through the use of `http_handler`. Further details regarding this implementation can be found in [Og23; Wi09].

Figure 2 shows the previous XPCE–based Tennis Tool (upper part) and the new Prolog web application (lower part). The lower part shows the transformed Tennis Tool, which has been migrated to a web–based application implemented in Prolog with DCG's. This transition allows for enhanced accessibility and functionality, offering a modernized and user–friendly interface for users to interact with the Tennis Tool.

6 Conclusion and Future Work

In this work, we have updated the process transforming tennis data in XML, which had originally been introduced in [We22]. Combining field and graph notation brings us closer to a completely automated transformation of the XML data derived from the video of a tennis match.

Logic programming and field and graph notation grammars were very useful for transforming this unclean XML data. On the one hand, we have used FNG's for filtering and renaming certain XML elements, on the other hand we have applied GNG's, whenever more complex executions were necessary.

To complete this transformation process, all that is missing now is the exact hitting type and a score analysis for a better nesting, which we want to implement in the next step using logic programming.

In addition, we have upgraded the XPCE based tennis tool to a Prolog web application using DCGs. We also want to integrate the analysis with data mining that we have done in previous work [We03; We19] using manually entered data,

References

[Ab00] Abiteboul, Serge and Buneman, Peter and Suciu, Dan: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, 2000.

[Ba19] Baumgart, M.: Erkennung von Spielstand, Schlagposition und Spielertrajektorien beim Tennis, Master Thesis, University of Würzburg, 2019.

[Ch03] Chamberlin, Don and Florescu, Daniela and Robie, Jonathan and Simeon, Jerome and Stefanescu, Mugur: XQuery: A Query Language for XML. In: SIGMOD Conference. Vol. 682, p. 50, 2003.

[Co78] Colmerauer, Alain: Metamorphosis grammars. Natural language communication with computers/, pp. 133–188, 1978.

[Di02] Dietmar Seipel: Processing XML–Documents in Prolog. In: Workshop on Logic Programming (WLP 2002). 2002.

[Me03] Mellish, Christopher S. and Clocksin, William F.: Programming in Prolog: Using the ISO Standard, 2003.

[No19] Nogatz, Falco and Seipel, Dietmar and Abreu, Salvador: Definite Clause Grammars with Parse Trees: Extension for Prolog. In: 8th Symposium on Languages, Applications and Technologies (SLATE 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[Og23] Ogborn, Anne: Tutorial – Creating Web Applications in SWI-Prolog, `https://github.com/Anniepoo/swiplwebtut/blob/master/web.adoc`, 2012 (Accessed on June 27, 2023).

[Pr17] Proestler, M.: Erkennung und Nachverfolgung von Balltrajektorien bei 2D-Fernsehaufnahmen im Tennis, Master Thesis, University of Würzburg, 2017.

[Se07] Seipel, Dietmar: PL4XML– An Swi–Prolog Library for XML Data Management (Manual), Citeseer, 2007.

[Se15] Seipel, Dietmar: Deductive Databases, Lecture Notes of a Course at the University of Würzburg, Lecture Script, since 2015.

[Se18] Seipel, Dietmar and Nogatz, Falco and Abreu, Salvador: Domain–Specific Languages in Prolog for Declarative Expert Knowledge in Rules and Ontologies. Computer Languages, Systems & Structures/, 2018, ISSN: 1477–8424, URL: `https://doi.org/10.1016/j.cl.2017.06.006`.

[Se94] Seipel, Dietmar and Thöne, Helmut: DisLog – A System for Reasoning in Disjunctive Deductive Databases. In: Proceedings of the 5th International Workshop on the Deductive Approach to Information Systems and Databases (DAISD 1994). Universitat Politècnica de Catalunya., pp. 325–343, 1994.

[Se97] Seipel, Dietmar: DisLog – A Disjunctive Deductive Database Prototype. In: Proc. Twelfth Workshop on Logic Programming (WLP'97). Pp. 136–143, 1997.

[Sh86] Shieber, Stuart M and Karttunen, Lauri and Kay, Martin and Pereira, Fernando C.N.: A Compilation of Papers on Unification-based Grammar Formalisms, Pts. I and II. Center for the Study of Language and Information, 1986.

[Wa07] Walmsley, Priscilla: XQuery. O'Reilly Media, Inc., 2007.

[We03] Wehner, J.: Verwaltung und Analyse von Zeitreihen zu Videosequenzen, Diploma Thesis, University of Würzburg, 2003.

[We19] Weidner, Daniel and Atzmueller, Martin and Seipel, Dietmar: Finding Maximal Non–Redundant Association Rules in Tennis Data. In: Declarative Programming and Knowledge Management. Springer, pp. 59–78, 2019.

[We21] Weidner, Daniel and Waleska, Marcel and Seipel, Dietmar: Interfacing the Declarative Toolkit Declare Using Python and Docker./, 2021.

[We22] Weidner, Daniel and Seipel, Dietmar: XML–Processing Using Field Notation Grammars Applied to Tennis Data./, 2022.

[Wi09] Wielemaker, Jan and others: Logic Programming for Knowledge–Intensive Interactive Applications. 2009.

A Appendix

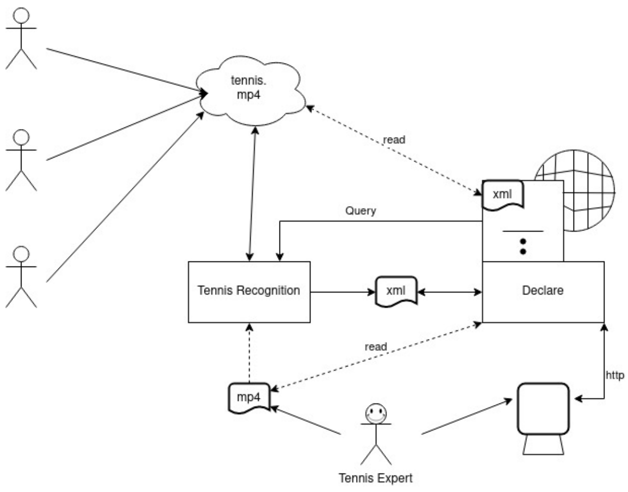
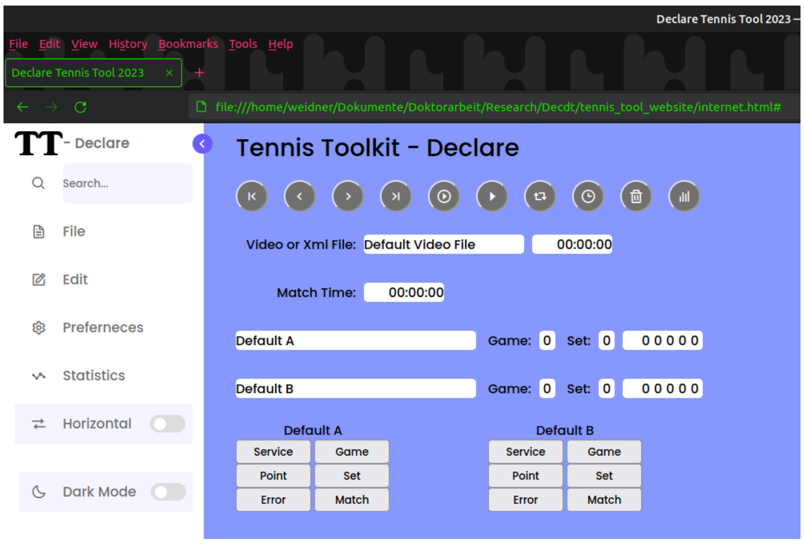


Figure 1: Architecture of the Web Tennis Tool of Declare



file:///home/weidner/Dokumente/Doktorarbeit/Research/Decdt/tennis_tool_website/internet.html#

Video or Xml File: Default Video File 00:00:00

Match Time: 00:00:00

Default A Game: 0 Set: 0 0 0 0 0 0

Default B Game: 0 Set: 0 0 0 0 0 0

Figure 2: The HTML–GUI of the Tennis Tool