Encoding Schemes for a Discourse Support System for Legal Argument

Henry Prakken and **Gerard Vreeswijk**[1]

Abstract. This paper reports on the ongoing development of a discourse support system for legal argument named PROSUPPORT. A description is given of the system's encoding schemes with which the user can enter his or her analysis of the discourse. These schemes, which are implemented as web browser forms linked to a database, serve to capture support relations of propositions within arguments, and dialectical relations between arguments. In addition, they support the recording of relevant argumentative and procedural speech acts made with respect to these arguments, such as disputing or conceding a claim, and allocating the burden of proof. The main issue in developing these encoding schemes is how expressiveness of the schemes can be reconciled with ease of use, on a suitable theoretical basis.

1 Introduction

In several related areas of computer science there is a growing interest in software support for such discourse processes as discussion, negotiation, dispute resolution and collective decision making. Unlike with 'conventional' decision-support tools (such as knowledge-based systems), the task of such systems is not to produce or suggest solutions to a problem with the help of domain knowledge, but to help the participants in discursive interactions to structure their reasoning and discourse, so that they can make sense of the discourse and interact effectively.

One professional area where such systems are of great potential use is the law. Participants in legal procedures (including alternative procedures such as online dispute resolution) often face the complex task of managing the information they are confronted with and the communication and reasoning they are expected to engage in. Discourse support systems can provide important assistance for these tasks: they could facilitate the structured inputting of a variety of discursive data, such as which claims have been made, conceded or challenged, how the burden of proof was assigned, which grounds and evidence have been adduced and counterattacked, how these grounds and evidence can be assessed, and whether the parties have respected the rules of procedure. The system could then usefully display, combine and restructure this input, and compute the consequences of the user's evaluative decisions (e.g. who wins given a certain allocation of the burden of proof and assessment of evidence?). Such systems could also support the (semi-) automatic generation of case summaries or even verdicts. These functionalities can be put to use in a variety of contexts. Individual users can be supported in making their own analysis of the discourse, invisible for other participants. The joint participants can be supported in their communicative and disputational interactions. Or the supporting staff of a judge or other official can be supported in their task to preprocess an analysis of a case, and to pass on the results to the official. Finally, in online versions of dispute resolution discourse systems could be a principal means of interaction between the participants.

In the field of AI & Law there is a growing body of theoretical research on discourse support for legal argument and legal procedure (e.g. [3, 1, 4, 12]). However, substantial research on architectures for implementation and on user experiences is still sparse. We know of only two systems that have been implemented with practical use in mind, viz. Loui's Room 5 system [8] and Verheij's ArguMed tool [18], and one further system that is currently being implemented, viz. Lodder & Huygen's support tool for online dispute resolution [7].

In other application areas, such as meeting support and intelligent tutoring, more practical experience with discourse support systems has been gained (see e.g. [9, 16, 15, 2]). These experiences raise important issues for legal discourse support systems. One of the main lessons learned is that it is very easy to overestimate the users' ability and willingness to learn a new codification scheme [15, 2]. The PROSUPPORT project, on which this paper reports, intends to take this lesson at heart. Its aim is to develop a discourse support system for legal procedure that provides useful computational power to the user but that is also easy to use.

Naturally, these two goals tend to conflict. The desire to offer useful computational power to the user requires that the user's input is structured as much as possible, in a way that reflects the essential elements of legal discourse. The more these elements are made explicit by the user, the more the system can do with it. However, the desire to make these elements explicit requires complex representation schemes for the user's input, which leads to a tension with the lessons on usability learned in other areas. Put simply, the more expressive a language, the harder it is to learn and use. Resolving this tension in an optimal way is one of the main research themes of the PROSUPPORT project. In other words, the project aims to discover conditions under which "formality" in interactive systems of the studied kind is helpful instead of harmful (cf. [15]).

To elaborate on the desired expressiveness, the following features of legal reasoning are especially relevant. Firstly, legal reasoning is adversarial, which means that arguments pro and con a claim are exchanged and conflicts between arguments must be resolved. Secondly, legal reasoning contains several specialised reasoning forms, such as combining rules and precedents, attacking the application of a rule, using and attack witness or expert evidence, reasoning about causation, and so on. Finally, legal reasoning takes place in a procedural context, where the notions of presumptions and burden of proof are essential, and where not only arguments but also other speech acts

[1] Institute of Information and Computing Sciences, Utrecht University, PO Box 80089, 3508 TB Utrecht, The Netherlands, email: {henry,gv}@cs.uu.nl

are important (such as disputing or conceding a claim and allocating the burden of proof).

There is another tension to be resolved. Being a research project, the system should have a sound theoretical basis, which means that it should be based on plausible theories of the structure and rationality of argumentative discourse. Moreover, since we are dealing with software specification, this theoretical basis should preferably be formal. The latter is particularly important since discourse support systems might be expected to compute the 'current state' of a dispute, given the arguments, counterarguments and priority arguments stated thus far. This requires a precise theory of what is to be computed. Now a problem is that most of the available theories are quite complex and subtle, especially when they are formalised. Therefore, directly implementing these theories would again detract from the usability of the system. A user can simply not be expected to master subtle theoretical notions and distinctions, let alone to deal with formal syntax or mathematical notions. Accordingly, a second research challenge of the PROSUPPORT project is to resolve the tension between naturalness and theoretical well-foundedness of the encoding schemes offered to the user.

This paper reports on our current proposals to resolve these two tensions, focusing on the encoding schemes for the user's input. The system is meant for Dutch civil procedure, and will be illustrated with an application to an actual Dutch civil case. It is important to note that in our design the interfaces for entering the user's input and for displaying the system's output are independent. Once information is inputted into the system, it is stored in an internal dataformat, which supports different ways of restructuring and visualising the information. This paper will not discuss interfaces for the latter.

As for the input encodng schemes, we propose a simple generic encoding scheme for argumentative and procedural speech acts. As for arguments, the scheme captures support relations between propositions within arguments and dialectical relations between arguments, but for the rest it imposes a minimum of structure on the user's input. We will show that this encoding scheme can be straightforwardly implemented as web browser forms linked with a database. Furthermore, we will argue that the design can be theoretically based on logics for defeasible argumentation and formal dialogue games for dispute resolution. Finally, we will discuss some limitations and possible extensions of our encoding schemes, and compare our proposals with related research.

2 The application domain

In this section we briefly describe Dutch civil procedure as far as relevant for present purposes. (This description is taken from [12] and inspired by [6]).

A civil law suit is divided into a 'pleadings' phase, where the adversaries plea their case before the judge and provide evidence when assigned the burden of proof by the judge, and a 'decision phase', where the judge withdraws to decide the case. The pleadings phase is separated into a written and an (optional) oral part. In the written part the parties exchange at least two and usually four documents (in fact, the law is about to be changed to make this "usually two"). The first is plaintiff's *Statement of Claim*, which has to contain plaintiff's claim plus his grounds for the claim. These grounds may be purely factual: plaintiff may leave out the legal 'warrant' connecting grounds and claim, as may both parties in all their other arguments. Also, parties do not need to explicitly state common-sense knowledge, and if they state such knowledge, they don't need to prove it. However, the judge decides what is common-sense knowledge. Defendant replies with her *Defence*, which has to contain all of defendant's attacks against plaintiff's claim and grounds. These attacks may also concern issues of procedure, so that the procedural legality of a move can itself become the subject of dispute. The adversaries may then exchange further documents as long as allowed by the judge. Each party may also ask to provide oral pleading. During the pleadings phase, the adversaries may dispute, concede and retract claims, defer to the judge's decision about a claim, support claims with arguments, move counterarguments, and offer to provide evidence for their claims. The judge assigns the burden of proof to a party whenever appropriate, after which that party must provide evidence (usually documents, or witness or expert testimonies). After the pleadings phase has ended, the judge gives his/her verdict, bound by the following rules of evidence.

An important principle of Dutch civil procedure is that the judge is passive with respect to the factual basis of the dispute. For instance, the judge must accept undisputed claims of the adversaries, and s/he must evaluate the evidence and give the verdict on the basis of the facts adduced by the parties, with the exceptions of generally known facts and legal rules. Of course, this does not mean that the judge cannot take factual decisions at all; s/he must still assess whether the facts adduced by the adversaries sufficiently support their claims, which may in turn also be factual.

As for allocating the burden of proof, the general rule is that the parties bear the burden of proving their claims; however, the judge may decide otherwise on the basis of special statutory provisions or on grounds of reasonableness. Among other things, this means that the burden of proof can be distributed over the parties, and that making a claim does not automatically create a burden to prove it; cf. [6, 11].

Given these characteristics of the procedure, our system should allow the following input. As for the adversaries, it should be possible to express which claims the adversaries have made, and which arguments they have stated in support of their claims or by way of counterargument. Furthermore, the system should keep track of which claims have been disputed, conceded, retracted or left to the judge's decision. Finally, the system should capture discussions on the procedural correctness of the adversaries' input (including admissibility of evidence). As for the judge, the system should record his/her decisions about such procedural correctness and about the burden of proof, including the judge's grounds for these decisions (when given). The system should also record the judge's completions of the adversaries' arguments with legal or commonsense knowledge. Finally, the system should allow for the inputting of any other argument moved by the judge, especially his/her assessments of evidence and conflicting arguments.

It is important to note that the PROSUPPORT system is not primarily meant to support the dispute as it actually takes place. Rather, the system is meant to support rational reconstructions of the dispute made by an individual user, either during or after the dispute. For instance, it could be used in the pleadings phase by one of the adversaries in preparing a further procedural document, or in the decision phase by the judge (or his assistants), in preparing the final verdict. It could also be used as an analysis tool by law students in a course on legal argumentation.

3 An example case

Throughout this paper we will use the following example case, concerning a dispute concerning ownership of a large holiday tent. Plaintiff (Nieborg) and his wife were friends of Van de Velde, who owned

a large tent at a camp site. At some point van de Velde mentioned

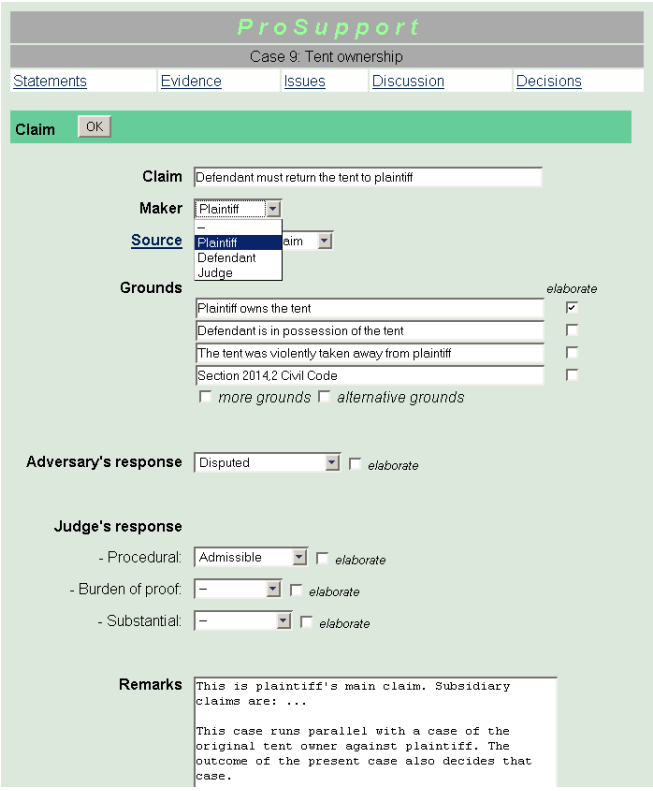

Figure 1. A claim form (expressing an argument).

was owner at the time of the violent events, or whether van de Velde had just given the tent on loan, so that van de Velde had remained the owner.

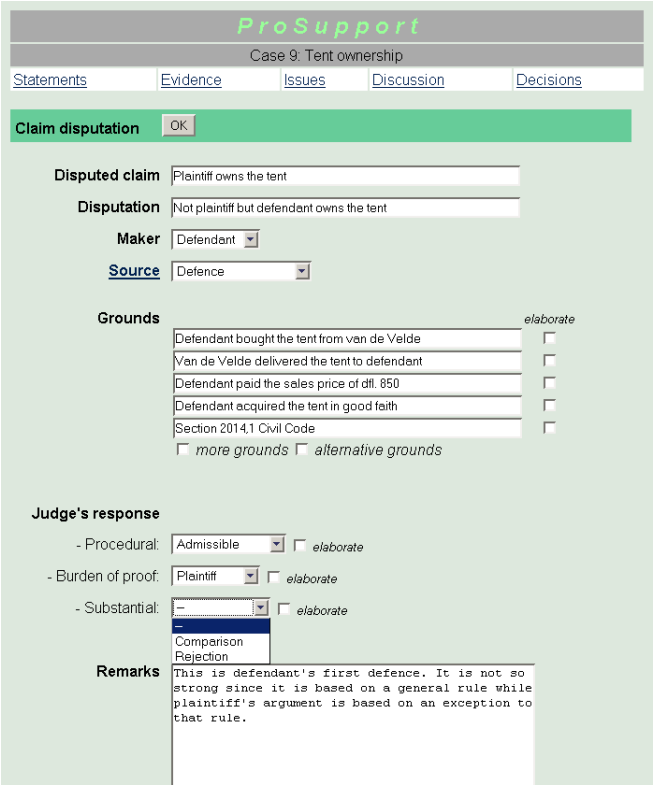

Figure 2. A claim disputation form (expressing a rebuttal).

that the tent was for sale for dfl. 850. Nieborg replied that he was interested but could not afford the price. Van de Velde still made his tent available to Nieborg, who in return helped van de Velde to paint his house, while Mrs. Nieborg for some period assisted Mrs. van de Velde with her domestic work. At some stage, Nieborg claimed that they had done enough work to pay the sales price for the tent, after which van de Velde became very angry and demanded the tent back since, so he argued, he had never sold the tent but only made it available to Nieborg for the period that he himself did not need it. He had done so since Nieborg had told him that he and his wife had never had have enough money to go on holiday. When Nieborg refused to return the tent, van de Velde, assisted by a group of people, threw Nieborg's son (who at that point was the only person present) out of the tent and took it away. A few months later, van de Velde sold the tent to defendant (van de Weg) and his wife. The sales price (dfl. 850) was paid with domestic work by Mrs. van de Weg in assistance of Mrs. van de Velde.

In court, Nieborg (plaintiff) claims return of the tent to him on the basis of his ownership. Van de Weg (defendant) disputes Nieborg's claim on the grounds that van de Velde had not sold the tent to Nieborg but only given it on loan, and that the work done by Nieborg and his wife was not done to pay the sales price but out of gratitude.

The relevant law is quite intricate and will not be explained here. The main issue on which the outcome of the case depended was whether van de Velde had sold the tent to Nieborg, so that Nieborg

Van de Weg was allocated the burden of proving that Nieborg had obtained the tent on loan. To meet his burden, he provided three witnesses, Van de Velde and two persons associated to van de Velde, Gjaltema and van der Sluis. Nieborg's main attack on van de Weg's evidence was that the witnesses were not credible: van de Velde had a personal interest in a win by van de Weg, and all three witnesses had declared something that Nieborg claimed was demonstrably false (we will not elaborate the latter point). However, the judge was convinced of their credibility, since their declarations supported each other and since Van de Weg had failed to find counterwitnesses. Nieborg therefore lost the case.

4 The discourse encoding schemes

We now turn to a description of the system's input encoding schemes, all based on the same generic scheme. In the present section we discuss their expressiveness and naturalness, while in the following section we describe them from a software-architecture point of view.

4.1 The schemes

In the present phase of the project, we have chosen for a simple format of arguments. Essentially, arguments are 'and trees' where the

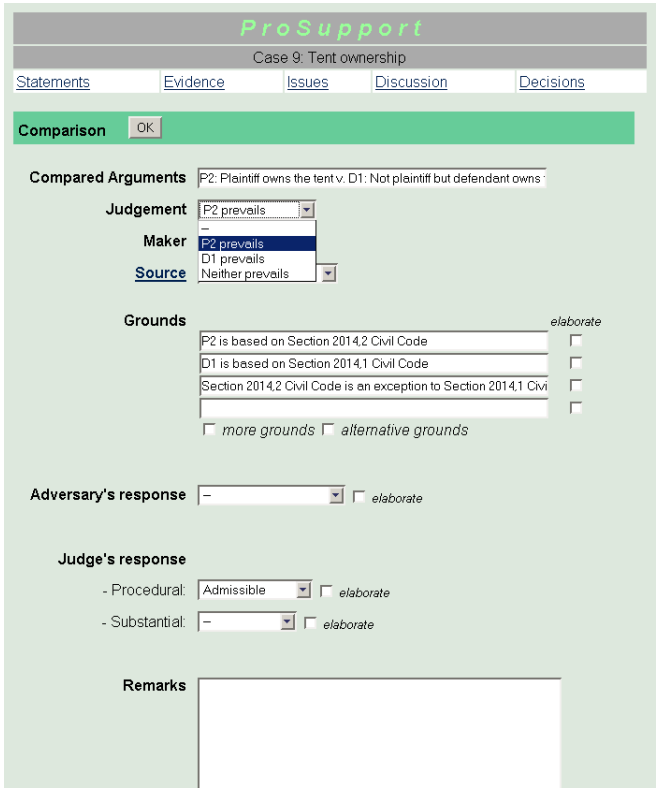

Figure 3. An argument comparison form (expressing a priority argument).

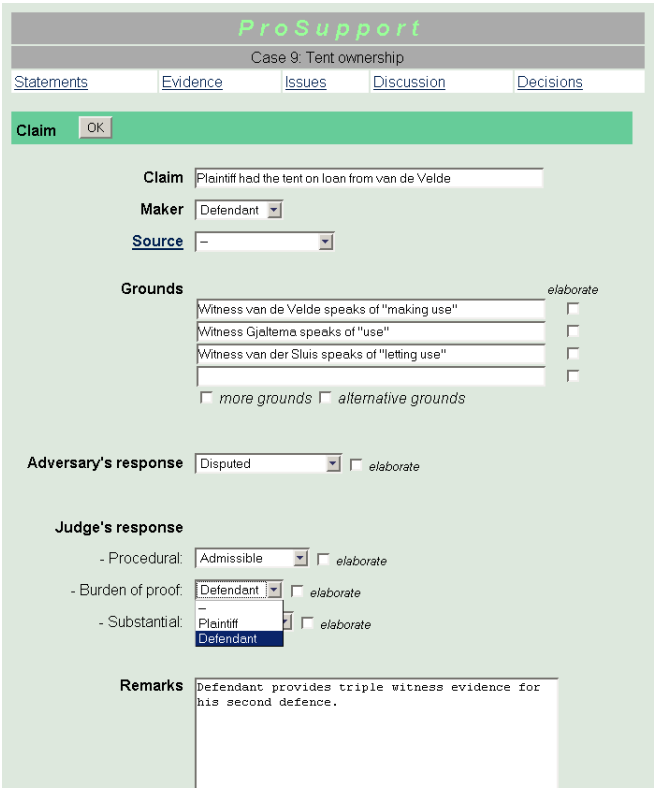

Figure 4. Another claim form (with an argument based on witness evidence).

nodes are propositional atoms and the links are inference rules. The tree's root is the conclusion and its leafs are the premises of the argument. This setup enables us to let the user input elementary arguments with a web form with a list of fields, as is illustrated by Figure 1[2], which displays a **Claim** form expressing an argument for plaintiff's main claim. The top field is the argument's conclusion and the fields under **Grounds** are its premises. If more than four grounds are needed, the user can tick the *more grounds* box and push the **OK** button. This scheme for arguments is recursive: elementary arguments can be extended by replacing one of its grounds with a subargument for that ground. This is achieved by ticking the *elaborate* box next to the ground to be elaborated and pushing the **OK** button, which returns another instance of the claim form, with the top field filled by the to-be-elaborated ground. This box can also be used if any other information about the ground is to be entered, such as that it was disputed, or that a certain burden of proof was attached to it.

To describe the further setup of the claim form, the top row hyperlinks are links to various overviews of the discourse generated by the system on the basis of previous input. Of these, as yet only the **Statements** and **Discussion** links have been implemented. The **Statements** link returns a table with all statements made so far by any of the participants, including useful 'metadata', such as who made the statement, how the other parties responded, and so on. The **Discussion** link returns a visualisation of the discussion so far.

With the choice menu **Maker**, the user can enter who made the claim, by choosing from the options *Plaintiff*, *Defendant* and *Judge*.

[2] The actual system is in Dutch; the English screens in this paper are created by manually editing the original HTML files.

With the choice menu **Source** the user can enter the case file document in which the claim can be found and, if desired, make a hyperlink to the relevant fragment in the document (this hyperlink feature is not yet implemented). Under **Adversary's response** and **Judge's response** the user can enter the eventual responses of the adversary, respectively the judge to the claim. These options will be explained in more detail below. Finally, at the bottom of the form there is a large **Remarks** field, for entering anything of interest that cannot be entered in the other fields or menus.

To return to arguments, they can, depending on their role in the dispute, take on several (non-exclusive) dialectical roles: they can be initial arguments, counterarguments, priority arguments, and procedural arguments. (Unless indicated otherwise, we below mean with 'argument' an elementary argument as expressed in a single form).

Counterarguments can in turn be of two types. *Rebutting* counterarguments deny the conclusion of the attacked argument, while *undercutting* arguments deny that the premises of the attacked argument support its conclusion. An example of a rebuttal is that not plaintiff but defendant owns the tent, since defendant bought and acquired the tent from the previous owner (see Figure 2, which contains a rebuttal of a (not shown) subargument for the first ground in Figure 1). An example of an undercutter is an attack on the credibility of a witness whose testimony was used in the attacked argument. Figure 5 displays an undercutter moved by plaintiff in attack of defendant's argument displayed in Figure 4. In legal disputes undercutters are very common, which is why we want to make the distinction between rebuttals and undercutters explicit, even though we are aware that this

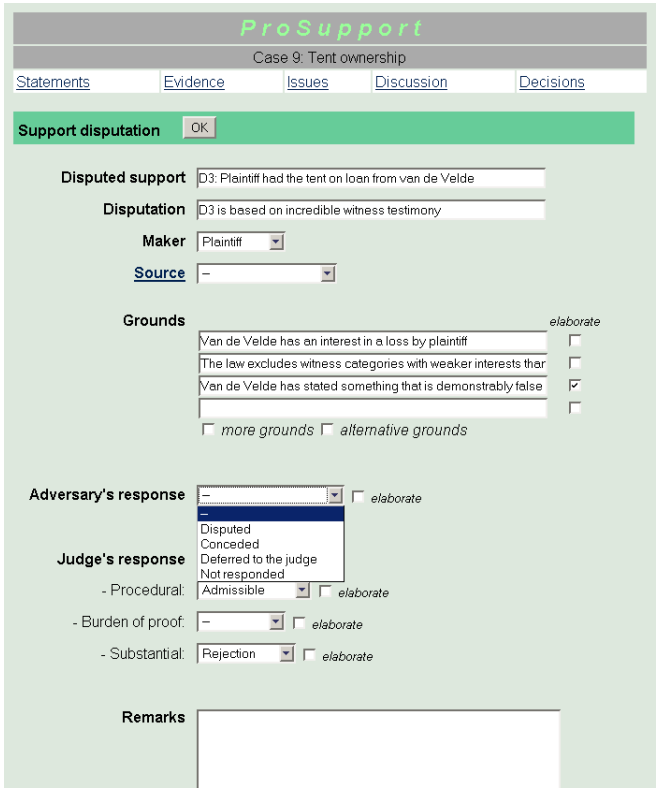

Figure 5. A support disputation form (expressing an undercutter).

complicates the encoding schemes and therefore might detract from their usability.

The system cannot automatically recognise from an argument's syntax whether it is a counterargument, since its input forms do not make negation explicit. Instead, the user must explicitly move a counterargument as an attack on another argument.

For counterarguments moved by an adversary this happens as follows. First from the **Adversary's response** choice menu the 'disputed' option must be chosen (as in Figure 1). This returns another choice menu, this time non-exclusive, with the options 'dispute claim' and 'dispute support' (not shown). The first choice makes the system return a **Claim disputation** form (See Figure 2, but note that that form was not the result of disputing plaintiff's main claim in Figure 1 but of disputing plaintiff's first ground. This disputation was entered in the subform (not shown) that elaborates this ground). The top field of a claim disputation form contains the disputed proposition, the second field is for the formulation of the disputation, and the remaining fields are for the grounds for the disputation. The system then treats the conclusions of an argument and its rebuttal as logical contraries. A choice for 'dispute support' makes instead the system return a **Support disputation** form (as in Figure 5, which resulted from disputing plaintiff's claim in Figure 4). Its top level field contains a system-generated description of the undercut support (in the current version an identifier plus the supported claim), its second field can be used to fill in the formulation of the undercutter, and the remaining fields can be used to enter the grounds for the undercutter.

A counterargument moved by the judge can be entered via the choice menu **Judge's response — substantial**, by choosing the option *rejection* (as in Figure 5). This makes the system return the same menu as with a 'disputed' choice for the adversary's response.

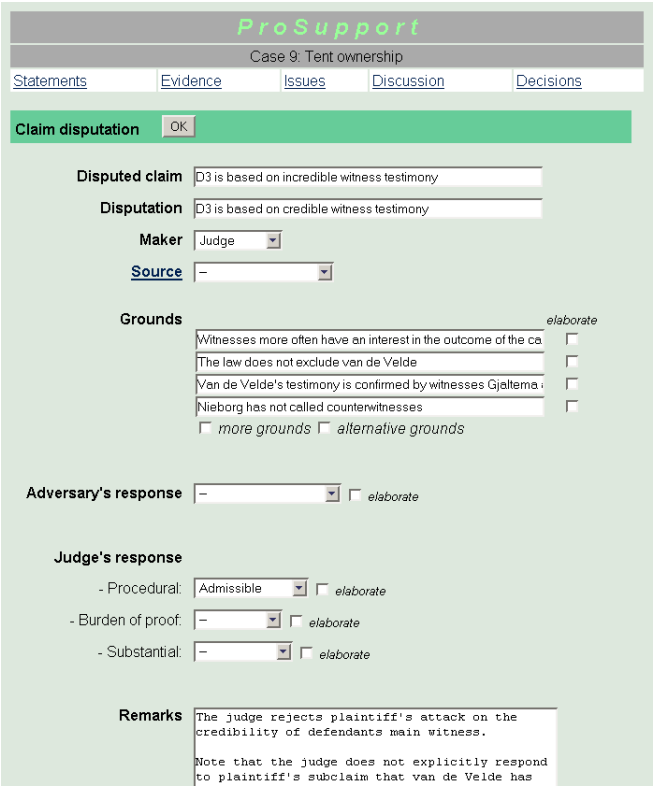

Figure 6. An implicit argument comparison by the judge

A *priority argument* is an argument that adjudicates a conflict between a rebuttal and its target argument. A priority argument of the judge can also be entered via the choice menu **judge's response — substantial**, by choosing the option *comparison* (see Figure 2). This returns a list of all rebuttals moved against the argument expressed on the form (not shown). The user can choose one of them, after which the system returns an **argument comparison** form (Figure 3). The top field mentions the identifiers and conclusions of the two arguments to be compared, the second field contains a choice menu for stating a preference between the arguments (a special form of a claim), and the rest of the form is as in the claim form. Note that thus we have slightly enriched our propositional language with the means to express preferences between arguments. In Figure 3 the judge adjudicates between two conflicting arguments concerning ownership of the tent. The judge prefers plaintiff's argument on the grounds that it is based on a legal rule which is an exception to the rule used by defendant's argument.

We do not allow priority arguments to adjudicate between an argument and its undercutter: if an undercutter is regarded as inconclusive, this should be expressed with a counterargument against the undercutter (as is done by the judge in Figure 6 with a rebuttal of plaintiff's undercutter in Figure 5). Such a counterargument can be a rebuttal (e.g. "no, the witness is credible, since . . . ") and then a prior-

ity argument can be moved on whether the undercutting argument or its rebuttal prevails (in fact, we regard a rebuttal moved by the judge as implicitly preferred over its target).

The last dialectical argument type is that of *procedural arguments*. They are subdivided into arguments on procedural correctness and arguments on allocating the burden of proof. A decision on procedural correctness can be entered with the choice menu **Judge's response — procedural** with the default *admissible* and a second option *inadmissible*. To enter an argument for an inadmissibility decision (which is optional), the box *elaborate* can be ticked, which makes the system return a form named **Violation**. Likewise for a decision on the burden of proof, via the choice menu **Judge's response — burden of proof**, which, when elaborated, returns a **Proof burden** form.

Finally, we must allow for alternative arguments for the same claim. Note that in a defeasible setting alternative arguments are not equivalent to a single argument with a disjunctive premise, since such a single argument does not capture that alternative arguments might be based on different kinds of inference schemes. For instance, one argument might be based on a statutory rule, while another argument might be based on legal policy considerations. Accordingly, below the list of grounds a box *alternative grounds* can be ticked, which returns an alternative claim form for the same claim. The alternative argument is assigned a different identifier than the original one.

4.2 How logical syntax is avoided

In our encoding scheme the user does not have to manipulate logical syntax, since logical operators are either implicit or not available. Above we already explained how negation is left implicit in the way rebuttals and undercutters are moved. Conjunction is, of course, implicit in the list of grounds. Furthermore, conditional operators are avoided since arguments do not have to be propositionally valid, so that conditional premises can be left implicit, paraphrased or named (e.g. with the name of a statutory rule as in Figures 1 and 2). Also, we think that there is no stong need for making disjunctions explicit. Firstly, as we explained above, alternative arguments for a claim (which are quite frequent) are not the same as an argument with disjunctive premises. Secondly, when a rule contains a disjunctive antecedent, we expect that in the great majority of cases to which the rule is applied, one of the disjuncts will hold. Consider, for instance, a social benefit law stating that being unemployed, ill or disabled entitles to a certain supplementary benefit. Finally, we expect that arguments that crucially depend on quantifiers or modal (such as deontic) operators will in practice be rare.

Of course, it is very likely that cases are found where our schemes are too limited. However, we think a discourse support system should not aim at 100% expressiveness, since that would conflict with the goal of usability.

4.3 How Dutch civil procedure has been modelled

In Section 2 we listed the features that our encoding schemes should capture. As can be seen from the above description, our schemes support the entering of all relevant dialectical types of arguments, as well as of all propositional attitudes (except retraction) that can be expressed by the adversaries and procedural decisions that can be taken by the judge.

We next recapitulate how the judge's substantial decisions can be entered. Completing the grounds of an adversary's argument can be simply done by adding a ground to an argument, ticking the corresponding *elaborate* box, and indicating in the elaboration form that the ground was moved by the judge. If the judge accepts an adversary's claim on alternative grounds, the user can simply check the box 'alternative grounds', enter such grounds and again indicate that they were moved by the judge. If a judge has rejected a claim or a claim's support on certain grounds, the user must choose the *rejection* option in the **Judge's response — substantial** menu, after which the claim or support can be disputed in the way explained above. Finally, the judge's comparative decisions can also be entered in a way explained above, by choosing the *comparison* option in the same menu. Note that the forms do not contain an explicit way to enter that the judge has accepted a certain claim. Such acceptance can be expressed either implicitly by doing nothing or, if the opponent had moved a counterargument, by attacking that argument in one of the available ways.

5 System architecture

We now describe the encoding schemes from a software-architecture point of view.

5.1 Design philosophy

The system architecture is based on the idea that all aspects of a case (issues, speech acts, source documents) are nodes in a network. The basic component (node) of the system's internal datastructure is called a *form*. Each form is intended to express a speech act. A form possesses several fields (or attributes), such as an ID, type, target, statement, maker, source, remarks, and typed pointers to other forms, such as grounds, adversary's response and judge's responses. Typically, each form uses only some of these attributes. For example, the main claim will have no value for the attribute 'target' because the main claim is the initial claim and by definition does not dispute other claims (see Figure 1). And a claim disputation form will have no adversary's responses, since a disputation is itself such a response (see Figure 2). When a form is presented to the user, undefined attributes are not shown, and the form takes its own "shape" depending on its type. Furthermore, depending on the type of form, its various attributes might be named in different ways. For instance, the attribute 'target', which links the form to a preceding form, is in a claim disputation form (Figure 2) called "disputed claim" and in a violation form (not shown) called "inadmissible speech act". And the attribute 'statement', which indicates the proposition a form is about, is in a claim form (Figure 1) called "claim" and in a 'comparison' form (Figure 3) called "judgement".

To prevent redundancy and preserve the logical structure of a case, every form is unique, which means that the same thing is always expressed in the same way. For example, if the statement field of a certain form is changed, and this form is used by forms A, B, and C, (e.g. as ground for their statement) then this change will be reflected if A, B or C are retrieved and presented on screen. Further, the system suggests the user to reuse forms by presenting ID's of existing forms. If the user enters a form-ID rather than plain text, the system will recognise this and will establish a link rather than create a new form. This feature can be used, for instance, to reuse old statements as grounds of a new argument.

As said above, form types are meant to stand for speech acts. We currently distinguish *Claim*, *Claim disputation*, *Support Disputation*, *Comparison*, *Violation*, and *Proof burden*. For instance, *Claim* stands

for making a claim, *Claim disputation* for disputing a claim, and *Violation* for deciding a speech act procedurally inadmissible. For some types of speech acts we do not want to allow for elaboration; such speech acts are not captured by their own form, but simply as an attribute of another form. For instance, conceding a claim is an attribute of a claim form. Finally, the speech act of moving an argument, i.e., of stating grounds in support of a claim or disputation is left implicit in the forms and how they are linked.

5.2 Aspects of human-computer interaction

Forms can be presented to the user in various formats. Currently, it is possible to view forms in isolation, and to view them all together. When viewed in isolation, all relevant attributes of a form are shown, including the contents of the statement fields of connected forms, and links to them. Showing the statement fields of connected forms increases the cohesion of the network and enables to user to quickly navigate through a case.

Viewing forms together enables a bird's-eye perspective on a case. Currently, the following global views are possible. The most obvious presentation consists of a table of all statements, accessible via the **Statements** hyperlink. This table can be sorted among various dimensions (e.g. ID number, type, time of input, time of modification). or filtered through various criteria (e.g. "show all disputed statements made by plaintiff for no burden of proof has yet been allocated"). Further, it is possible to view a tex-based summary of the case (via the **Discussion** hyperlink) and to view the case as a directed graph (not yet incorporated in the above screens). It should be noted that our architecture does not commit to a particular visualisation style of the discussion; it equally supports text-based and graph-based styles.

One of the greatest challenges of our project is to keep the layout of the input forms as simple as possible, while respecting the complexity of the case. The approach that PROSUPPORT follows is that it is kept simple and fixed for beginners, while advanced users may opt for more features and flexibility.

5.3 Current state of the implementation

The current version of our system is implemented in Mason (`http://www.masonhq.com`). Mason is a Perl-based web site development and delivery engine. With Mason it is possible to embed Perl code in HTML and construct pages from shared, reusable components. Mason requires an Apache HTTP server with a software package that embeds a Perl interpreter into the webserver (typically `mod_perl`). Forms are written to and retrieved from a Berkeley type data base, where forms are accessed by their ID.

As for the current state of implementation, the above-described form-based datastructures have been implemented, as well as a first method to navigate between the encoding screens. Of the overview facilities, only the **Statements** and **Discussion** features have been implemented. We have not yet implemented the function that is meant to compute the 'current outcome' of a case.

Some elements of our implementation are still provisional. Firstly, as for navigating between the forms, some problems still have to be solved. One problem is that the user can mark more than one text field for further elaboration. In such cases, more than one form needs to be filled out and it is not immediately clear which of these forms that should be, i.e., which of these forms must be presented next to the user. One solution is to work with a prioritised agenda, called "forms to be processed," and then to enable the user to process these forms as he sees fit. Secondly, our current way to visualise the discussion is also still provisional; in fact, a full implementation of this feature is an important research issue of the PROSUPPORT project, which will touch upon cognitive as well as technical issues.

6 Theoretical foundations

As said above, one goal of the PROSUPPORT project is to investigate how a natural encoding scheme for argumentative discourse support can be developed on a sound formal basis. We think that such a basis can be provided by combining two recent developments, viz. logics for defeasible argumentation and formal dialogue systems for critical discussion.

6.1 Logics for defeasible argumentation

Logics for defeasible argumentation (see [14] for an overview) are one approach to the formalisation of so-called defeasible, or nonmonotonic reasoning. This is reasoning where tentative conclusions are drawn on the basis of uncertain or incomplete information, which might have to be withdrawn if more information becomes available. Logical argumentation systems formalise this kind of reasoning in terms of the interactions between arguments for alternative conclusions. Nonmonotonicity arises since arguments can be defeated by stronger counterarguments.

There are several reasons why argumentation systems are a promising formal basis for argumentative discourse support systems. Clearly, modelling inference as comparing arguments and counterarguments fits very well with the dialectical nature of argumentative discourse. Moreover, argumentation systems often abstract to a large degree from the logical language in which arguments are expressed and from the rules according to which they are constructed. This makes such systems particularly suitable for dealing with natural-language input. For instance, above we saw how logical syntax can be avoided and how hidden premises can remain implicit. Finally, argumentation logics have been applied to a number of phenomena that we think are important in argumentative discourse support, such as the format of arguments as trees of inference rules (e.g. [10, 19]), the distinction between rebuttals and undercutters (due to Pollock, e.g. [10]), and priority arguments (e.g. [5, 13]). Note that all these three phenomena are captured by our encoding schemes.

6.2 Dialogue games for dispute resolution

In the introduction we said that one use of formal foundations is as a basis for computing the 'current outcome' of a dispute. Now it is important to note that the outcome of a dispute depends not only on the arguments that are stated but also on the various argumentative speech acts and procedural decisions. For instance, if a premise of an argument is disputed and no further argument for it is given, the argument does not count in determining the outcome of the dispute; likewise for an argument of which one premise was ruled to contain inadmissible evidence. And for computing the effect of priority arguments on the outcome of a dispute, it is important to know who has the burden of proof: if two conflicting arguments are decided to be equally strong, this benefits the adversary who does not have the burden of proof.

So argumentative speech acts of various kinds interact in subtle ways in determining the outcome of a dispute. Therefore, the formal basis of a discourse support system cannot be confined to argumentation logics; they need to be embedded in formal dialogue systems

for dispute, for instance, in the dialogue systems of [21]. For two examples of work of this kind see [3] and [12].

Accordingly, we have set up PROSUPPORT such that each input in the system can be formally translated as a move in such a dialogue system (although we have not yet fully carried out this translation). On the other hand, we have also designed the system such that the user needs not be aware of this translation. The reason is that we expect the intended users will find a WEB-form interface more natural than an explicit dialogue game style interface, which still seems somewhat artificial.

7 Discussion of alternatives and remaining issues

As for arguments, the expressiveness of our system lies mainly in two aspects: it can keep track of (often nested) support relations between statements, and it can identify the main dialectical relations between arguments. However, our language for expressing arguments is (deliberately) very simple. We now discuss some possible enhancements.

As explained, our system allows to distinguish three parts of (elementary) arguments: their premises, their conclusion, and their inference rule. (Actually, the nature of the inference rule is not made explicit; instead it is only named). We could, of course, have imposed more structure. One scheme that comes to mind is Toulmin's well-known generic argument scheme [17]. However, we fear that this scheme might be too rigid and too complex for practical use, since it requires that for every argument a uniform distinction between data, warrant and backing is made explicit. Especially when combined with the practical need to make the scheme recursive, this often leads to quite complex encodings of legal arguments, as was shown by [9].

In our opinion, a more promising refinement is the inclusion of a set of optional specialised argument schemes. ("Optional" means that such schemes could be offered as an advanced option to experienced users of the system.) Specialised argument schemes are an important research topic within argumentation theory (see e.g. [20]). For present purposes, some useful schemes are the use of types of evidence (such as witness testimonies, expert reports, and documents). Such specialised argument schemes are less rigid and abstract than Toulmin's scheme. Moreover, they come with specific sets of 'critical questions', which can focus a discussion. Finally, the logical interpretation of argument schemes is rather straightforward: they naturally map onto Pollock's well-known notions of defeasible reasons and defeaters. Note that a negative answer to a critical question attached to an argument scheme will in fact be a counterargument, often of the undercutting type. For instance, Walton in [20] lists as one of the critical questions of arguments from testimony, the question whether the witness is credible. Above in Figure 5 we formulated a negative answer to this question as an undercutting counterargument.

An important restriction of our generic scheme is that, as for support relations between propositions, it can only capture and-tree relations between propositions. For certain types of reasoning, such as abductive-causal reasoning or probabilistic reasoning, this may not be suitable.

Finally, we have chosen not to model the concept of propositional commitments in our system. Although this is a very important theoretical concept (cf. [21]), we think that violation of commitments will in practice not often be an issue, while modelling them makes the system more complex and thus detracts from the goal of usability.

8 Related research

In the legal field, so far been two implemented architectures for practical use have been described, viz. Loui's Room 5 system [8] and Verheij's ArguMed [18]. A related system outside the legal field is Belvedere [16], a system for teaching scientific argumentation. Furthermore, Lodder & Huygen [7] report on the ongoing development of their support tool $eADR$ for simple procedures for online dispute resolution.

All four systems support the user in drafting arguments and counterarguments (Room 5 also supports the search of legal case databases and the incorporation of retrieved case citations in arguments). ArguMed is the only system that, besides rebuttals, also supports undercutters; none of the systems supports priority arguments. Unlike PROSUPPORT, these systems do not support the entering of other relevant speech acts. Room 5 and ArguMed are, like PROSUPPORT based on logics for defeasible argumentation, and have an implemented 'current outcome function' based on such a logic. Belvedere and $eADR$ are not based on formal foundations. As for the appearance of the input forms, ArguMed and Belvedere are graph-based, while Room 5 uses encapsulated text frames and $eADR$ uses a format similar to threaded discussion boards, where replying messages can be either supporting or attacking replies (the authors do not specify whether multiple supporting replies are meant to be cumulative or alternative grounds). Neither of these projects addresses the issue of the generation of discussion overviews in formats different from their encoding schemes. Finally, Belvedere is the only of these four systems that has been subjected to systematic field studies.

Summarising, we think that, compared to these systems, our main contributions are a separation of the layouts of the input and output interfaces, an alternative, web-browser-based interface for input encoding schemes, and the modelling not only of arguments and their dialectical relations, but also of argumentative and procedural speech acts. The latter feature especially allows for an adequate modelling of reasoning under burden of proof, which in legal applications is very important. It remains to be seen whether this extra expressiveness makes the resulting extra computational power outweigh the increased complexity of use.

9 Conclusion

In this paper we have investigated to which extent a theoretically well-founded account of argumentative discourse can be implemented as an argumentative discourse support system. We have especially focused on the encoding schemes with which the user can enter his or her analysis of a dispute. The main question was how such encoding schemes can, on the one hand, be natural and easy to use and, on the other hand, support useful computational power of the system. With respect to the latter, we have especially kept in mind a feature that computes the 'current outcome' of a dispute.

We have argued that, if the expressiveness of the encoding schemes is sufficiently restricted, a natural and useful implementation is possible with a world-wide popular software tool, viz. web browsers, linked to a database. We have also argued that, with respect to expressing arguments, a suitable restriction is to encode no more than support relations between statements within arguments, and dialectical relations between arguments. Moreover, we have argued that our encoding schemes can be given a formal basis in terms of logics for defeasible argumentation and formal dialogue systems for critical discussion.

Of course, our findings are still preliminary. For one thing, we have so far tested our designs on the case files of only one case. More importantly, so far we have not obtained any substantial user experience, which yet is essential for testing usability and usefulness. Nevertheless, we think the results so far are promising enough to further develop our approach and conduct realistic field tests.

REFERENCES

[1] T.J.M. Bench-Capon, T. Geldard, and P.H. Leng, 'A method for the computational modelling of dialectical argument with dialogue games', *Artificial Intelligence and Law*, **8**, 233–254, (2000).

[2] J. Conklin, A. Selvin, S. Buckingham Shum, and M. Sierhuis, 'Facilitating hypertext for collective sensemaking: 15 years on from gIBIS', in *Proceedings of the The Twelfth ACM Conference on Hypertext and Hypermedia (Hypertext 2001)*, New York, (2001). ACM Press. In Press. Also available as Technical Report KMI-TR-112, Knowledge Media Institute, The Open University, UK.

[3] T.F. Gordon, *The Pleadings Game. An Artificial Intelligence Model of Procedural Justice*, Kluwer Academic Publishers, Dordrecht/Boston/London, 1995.

[4] J.C. Hage, 'Dialectical models in artificial intelligence and law', *Artificial Intelligence and Law*, **8**, 137–172, (2000).

[5] R.A. Kowalski and F. Toni, 'Abstract argumentation', *Artificial Intelligence and Law*, **4**, 275–296, (1996).

[6] R.E. Leenes, 'Burden of proof in dialogue games and Dutch civil procedure', in *Proceedings of the Eighth International Conference on Artificial Intelligence and Law*, pp. 109–118, New York, (2001). ACM Press.

[7] A.R. Lodder and P.E.M. Huygen, 'eADR: a simple tool to structure the information exchange between parties in online alternative dispute resolution', in *Legal Knowledge and Information Systems. JURIX 2001: The Fourteenth Annual Conference*, pp. 117–129, Amsterdam etc, (2001). IOS Press.

[8] R.P. Loui, J. Norman, J. Alpeter, D. Pinkard, D. Craven, J. Linsday, and M. Foltz, 'Progress on Room 5: A testbed for public interactive semi-formal legal argumentation', in *Proceedings of the Sixth International Conference on Artificial Intelligence and Law*, pp. 207–214, New York, (1997). ACM Press.

[9] S.E. Newman and C.C. Marshall, 'Pushing Toulmin too far: Learning from an argument representation scheme', Technical Report SSL-92-45, Xerox Palo Alto Research Center, Palo Alto, CA, (1992).

[10] J.L. Pollock, *Cognitive Carpentry. A Blueprint for How to Build a Person*, MIT Press, Cambridge, MA, 1995.

[11] H. Prakken, 'Modelling defeasibility in law: logic or procedure?', *Fundamenta Informaticae*, **48**, 253–271, (2001).

[12] H. Prakken, 'Modelling reasoning about evidence in legal procedure', in *Proceedings of the Eighth International Conference on Artificial Intelligence and Law*, pp. 119–128, New York, (2001). ACM Press.

[13] H. Prakken and G. Sartor, 'Argument-based extended logic programming with defeasible priorities', *Journal of Applied Non-classical Logics*, **7**, 25–75, (1997).

[14] H. Prakken and G.A.W. Vreeswijk, 'Logics for defeasible argumentation', in *Handbook of Philosophical Logic*, eds., D. Gabbay and F. Günthner, volume 4, 219–318, Kluwer Academic Publishers, Dordrecht/Boston/London, second edn., (2002).

[15] F.M. Shipman and C.C. Marshall, 'Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems', *Computer Supported Cooperative Work*, **8**, 333–352, (1999).

[16] D. Suthers, A. Weiner, J. Connelly, and M. Paolucci, 'Belvedere: engaging students in critical discussion of science and public policy issues', in *Proceedings of the Seventh World Conference on Artificial Intelligence in Education*, pp. 266–273, (1995).

[17] S.E. Toulmin, *The Uses of Argument*, Cambridge University Press, Cambridge, 1958.

[18] B. Verheij, 'Automated argument assistance for lawyers', in *Proceedings of the Seventh International Conference on Artificial Intelligence and Law*, pp. 43–52, New York, (1999). ACM Press.

[19] G.A.W. Vreeswijk, 'Abstract argumentation systems', *Artificial Intelligence*, **90**, 225–279, (1997).

[20] D.N. Walton, *Argumentation Schemes for Presumptive Reasoning*, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.

[21] D.N. Walton and E.C.W. Krabbe, *Commitment in Dialogue. Basic Concepts of Interpersonal Reasoning*, State University of New York Press, Albany, NY, 1995.