Assessment Based Legal Information Serving and Cooperative Dialogue in CLIME

Radboud Winkels, Alexander W.F. Boer,
Joost A. Breuker and Doeko J.B. Bosscher

Dept. of Computer Science & Law, University of Amsterdam
{winkels, aboer, breuker, bosscher}@lri.jur.uva.nl

Abstract

Legislation grows in number and complexity, and so does the need to get informed about them, and to be able to infer the consequences of rules and requirements for particular situations. Access to legal sources has thus far been handled in the same way as information retrieval in general by using database technology. This misses out what is the crucial issue in *legal information serving (LIS)*: reasoning about the legal consequences of a query. In the recently started ESPRIT project CLIME (EP25.414) we will build a client-server LIS architecture that will determine legal consequences of the typically abstract and underspecified legal cases expressed in queries. It is an opportunity to test our ideas on LIS in a large realistic legal domain, which has already led to refinement of reasoning modules and specification of new tools.

1. Introduction

Regulations and laws grow in number and in complexity, and so does the need to get informed about these, to find one's way in the documents, and to be able to infer the consequences of rules and requirements for particular situations. Access to legal sources has thus far been handled in the same way as – or rather following – information retrieval in general by using databases or (structured) text bases (see e.g. Turtle 1995 for an overview). The search engines at the WWW are a good example of the state of the art.

In legal information serving, key word matching has serious limitations, even if supported by conceptual retrieval techniques. Typically, the input query combines key words through Boolean and proximity operators, and the output is a list of (ranked) (parts of) documents. Moreover, the quantity and quality of the search result leaves much to be desired. One may find a lot of irrelevant documents (low precision) and probably not all relevant ones (low recall).[1] These techniques are directly borrowed from information retrieval in general, but miss out what is the crucial issue in *legal information serving (LIS)*: reasoning about the legal consequences of the query (Breuker 1992). In LIS the user is interested in the question whether some situation is allowed or required. Typical LIS requests are for example:

1 Blair & Maron (1985) found that 'full-text' databases provided only 15% of all relevant documents and 30% of critically relevant documents, while at the same time the users thought they had found 80% or more of all relevant documents.

"Can one park one's car on the left hand side in Italy?", or "Can one receive gifts while on social security benefits?".

The domain we are interested in are the requirements for a ship to be 'classified', a requirement for access to ports, insurance etc. Every ship classification society maintains a set of rules for assessing ships. These regulations cover a few thousand of pages of text and are available in electronic form to the society's employees, who are spread world wide, and their clients (ship owners).[2] Besides these regulations international treaties, e.g. on safety (SOLAS) and preventing maritime pollution (MARPOL) are applicable. Typical requests are "What is the minimal number of bilge pumps that is required on a cargo-ship", "Are passengers allowed on a bulk-carrier?" Answering these questions involves *assessing* the normative status of the request, i.e. matching norms to a situation description, in the same way as in the assessing the legal consequences in a legal case. However, LIS generally differs from the legal assessment of cases because the 'case' may be incomplete and underspecified. This is often explicitly intended. If one asks about passengers, one is not interested in bilge pumps. Moreover, if bilge pumps happen to be related to passengers in (one of) the norms, the LIS should make this explicit to the user. The normative nature of these requests transpires in the appropriate answer e.g. to the first question: "Yes, a bulk carrier may have passengers" is a correct answer, but not a very cooperative one. The LIS may explain that a bulk-carrier is a cargo-ship, and that cargo-ships may carry a limited number of passengers.

This latter kind of reasoning is necessary in LIS because the information in the request may not directly match some (set of) norm(s), but only indirectly via its *implied* knowledge. In fact, LIS requests may hardly ever give a direct match to norms, because norms are abstractly formulated to provide a large coverage of situations. For that reason, the use of database technology or text-retrieval methods leads to low precision and low recall scores. Conceptual front-ends[3] may improve the matching with the abstractions in normative statements, but they do not cover in a principled way how implied knowledge is handled. For instance, in finding out whether parking on the left-hand side is allowed, no conceptual front-end will discover that crossing the road is the major obstacle, and that therefore one-way streets are good candidates.

In this paper we will focus on the assessment function in LIS, and discuss the problems of reasoning with implied knowledge at the end. The LIS presented in this paper is under construction as part of the CLIME project ("Computerised Legal Information Management and Explanation", Esprit P25.414). Besides delivering a generic architecture for LIS, a demonstrator will be delivered for the domain of ship classification. This provides us with an opportunity to test our ideas on LIS in a large realistic (para)legal domain, which has already led to refinement of the assessment function and specification of new tools.

In the next section we will describe the architecture of CLIME. Then we will focus on the LIS module and describe its assessment function. Finally we will discuss the fact that in a cooperative dialogue it is not sufficient to present a justification of the results found, but that the user should also be

2 The classification regulations in CLIME are those of Bureau Veritas, one of the oldest classification companies. The regulations are electronically accessible on CD-ROM or via internet, coded in SGML.

3 As e.g. in the FLEXLAW system (Smith *et al.* 1995).

warned that if a more specific aspect of the query is addressed, the outcome may be legally different.

2 The CLIME architecture

The overall architecture of the CLIME system is shown in Figure 1. It consists of a central server and three functionally different clients. Communication between the CLIME server and the client interfaces is via secure http and CORBA. The CLIME server resides on the public internet (or a private intranet) in the form of a secure http server which provides a gateway to the CLIME system itself. In addition, the server supports the downloading of the client interface modules using HTML and Java protocols. This means that all that is required to use the system is a standard web browser: once a connection to the CLIME server is established, the appropriate interface module will be downloaded to the client browser automatically.

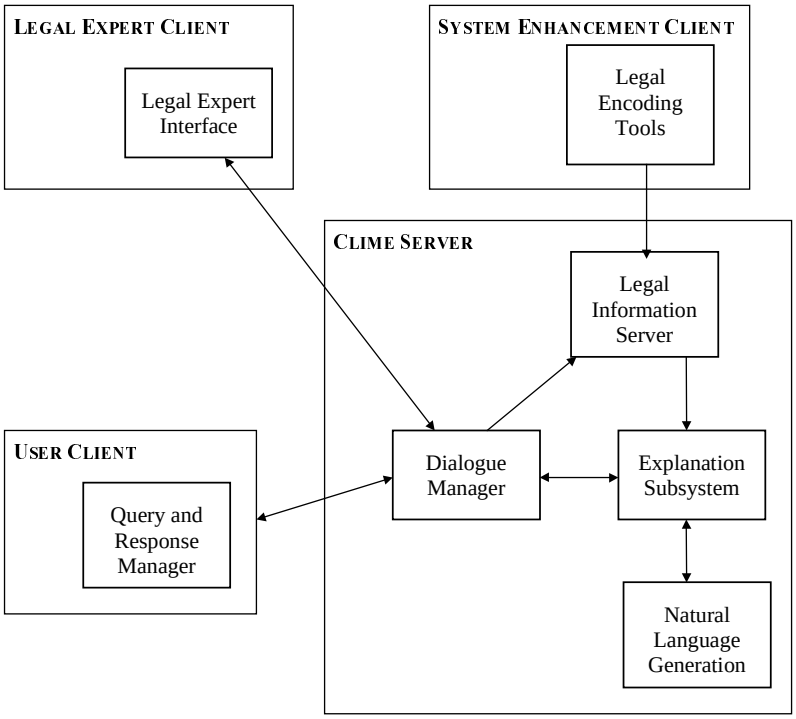

Figure 1. General CLIME architecture

The functional organisation of the CLIME system is as follows. Initially, a user establishes a network connection from a standard web browser to the CLIME server. The server responds by downloading the Query and Response Manager module to the user's browser. This module allows the user to formulate and submit queries to the CLIME system. This query is routed first to the Dialogue Manager. This module is responsible for managing the dialogue state, both within the current interaction and also over longer periods of time. The Dialogue Manager handles dialogue state queries directly, passes new queries to the Legal Information Server and follow-up requests to the Explanation Subsystem. The Legal Information Server will generally produce a response to the query and pass it to the Explanation Subsystem. The Explanation Subsystem is responsible for formulating responses to queries and follow-ups. It uses the output from the Legal Information Server to determine the structure and content of a response, and the Natural Language Generation component to compose

fluent text from the internal formalism. It formulates the response in HTML and passes it back to the Dialogue Manager and from there to the Query and Response Manager for presentation to the user. If this process fails after simple automated recovery procedures, the query can be routed to a human expert. The expert connects to the CLIME server (in a different mode to a normal user) and the Legal Expert Interface is downloaded to his web browser. Figure 1 also shows that the CLIME project will develop Legal Encoding Tools to assist the modelling and encoding of legal knowledge. This is necessary to maintain the data and knowledge used by the various CLIME modules, in particular the regulations that feed the LIS.

2.1 The Legal Information Server

The *Legal Information Server* (LIS) architecture design for CLIME is based on some fundamental ideas from ON-LINE, a prototype LIS (Valente 1995). Firstly, the LIS includes both a text representation and knowledge representation of the legal source document, and encodes the model correspondence between these modelling links. This allows the knowledge representation to differ considerably from the text structure while the system remains able to return the appropriate text with a decision of the system.

Secondly, the LIS should (1) be specific to typical legal reasoning tasks, and (2) extend the reasoning capabilities of the typical user. The core of all legal reasoning is *Normative Assessment* of a case, given a body of norms. It is this type of reasoning that creates the need for Legal Information Retrieval in general.

Thirdly, the conception of legal domains underlying LIS – the generic ontology – should be tailored to the reasoning steps in the normative assessment task. A distinction is made between world knowledge and normative knowledge (Breuker & Den Haan 1991). The ontology for the description of normative knowledge makes a distinction between the deontic type of a norm (one of *permission*, *prohibition* and *obligation*) and a normative qualification of a case (one of *allowed*, *disallowed*, *silent*) solving some common problems with deontic logics (cf. Valente 1995). This will be explained in more detail below.

2.2 Normative Assessment

The core task of a Legal Information Server (LIS) is normative assessment: Assessing whether a certain case complies with, or deviates from, a body of norms (law). The task *assess* is decomposed into subtasks *abstract* and *match*, and *match* in turn is decomposed into subtasks *evaluate* and *resolve conflicts*. The implementation of assessment is a specialisation for the domain of Law of a generic Problem Solving Method (PSM) for Assessment from the CommonKADS Library (Valente & Lockenhöff 1994). Analogous to the model it specialises, it requires a specific type of knowledge for each subtask:

1. *World knowledge for abstraction*: In the abstraction process, the terms in the case description are abstracted to those mentioned in the norms. The world knowledge is a legal abstraction of some aspect of the world. For the MILE domain of ship classification this concerns e.g. types of ships, their parts, surveys, ship class, service notation, etc. Moreover, it is not complete and self-

contained, but refers to and depends on common-sense knowledge. In the ship classification domain, many types of ships are defined, based on their constituent parts, cargo, etc., but the term 'ship' itself is not. This is left to common-sense interpretation by humans. In this sense, the world knowledge, and the models that it constitutes, are a device for translation between the 'real world' and the 'legal world'.

2. *Normative knowledge for evaluation*: When a case description has been abstracted, it will be evaluated. All norms that apply to the case will assign a normative qualification to the case. These norms can be directly applied, because the case has been transformed into terms mentioned in the conditions of these norms. A norm in the MILE domain is e.g. the *obligation* of ship owners to notify the Classification Society of any damage to the hull of the ship. This means that cases in which such a damage is not reported will be labelled 'disallowed' (or 'illegal'), while the norm remains silent on cases in which it is reported.
3. *Meta-legal knowledge to resolve conflicts*: Finally, since more than one norm may apply and conflicts between norm applications may arise, a conflict resolution mechanism will have to yield a definite qualification of the case – if it exists. Suppose a rule in the MILE domain states that damage to the hull during repairs in a dry-dock does not have to be reported. A situation in which this rule applies is labelled 'allowed' by this rule, but 'disallowed' by the one previously mentioned. Meta-legal knowledge should in that case decide which norm application is *valid*, considering the entire normative system. In the example it might be that the '*lex specialis*' principle states the second rule is more specific and therefore overrules the first rule, yielding a valid normative qualification of 'allowed'. Note that it is possible that a case is assigned more than one normative qualification, because different aspects of the case trigger independent sets of norms, e.g. a ship that has an inoperable bilge pump and damage to the hull.

The goal of the normative assessment task is qualifying a case as *allowed*, *disallowed*, or *silent*. This functionality is required to answer the most fundamental type of legal question, but other types of questions are common. Queries may be e.g. concerned with legal definitions, establishing legally qualified facts, retrieving applicable norms. The solutions to these queries correspond, however, with specific parts of the argument constructed during assessment.

2.3 The LIS Normative Assessment module

The structure of the assessment task as specified and implemented is slightly different from the standard PSM model from which it is derived. The structure closely follows the formalisation of normative reasoning given by Valente (1995), but the design and implementation differs. It collects more intermediate results – mainly for explanation purposes – and it is much more efficient.[4]

4 Besides the CLIME team, André Valente (ISI) and Jos Lehmann (Amsterdam) have redesigned and reimplemented ON-LINE, which is now already between 50 to 200 times faster.

Assess is implemented as follows:

Function	**Assess** (**Case**, **Normative-System**) returns **Decision**
Input-roles:	**Case** – the – already abstracted – Case is in abstracto described by a conjunction of grounded propositions using connectives {∧,¬}. **Normative-System** – a set of **Norm**s.
Subtasks:	**Apply**, **Resolve-Conflicts**

For each **Norm** in the **Normative-System**,
  If the **Norm** is a **Permission**,
    Apply **Norm** to **Case**, obtaining **Minimal-Cases** that are '**Allowed**', and
      Store the **Minimal-Cases** in **Matching-Cases**, without duplicating any, and store
      The associated **Minimal-Cases, Norm, and** '**Allowed**' in the **Positive-Set**.
  If the **Norm** is a **Prohibition**,
    Apply **Norm** to **Case**, obtaining **Minimal-Cases** that are '**Disallowed**', and
      Store the **Minimal-Cases** in **Matching-Cases**, without duplicating any, and store
      The associated **Minimal-Cases, Norm, and** '**Disallowed**' in the **Positive-Set**.
    Apply **Norm** to the **Opposite** of **Case**,
      Obtaining **Minimal-Cases** that are '**Possible-Conflicts**', and
      Store the **Minimal-Cases** in **Matching-Cases**, without duplicating any, and store
      The associated **Minimal-Cases, Norm, and** '**Possible-Conflicts**' in the **Negative-Set**.
  If the **Norm** is an **Obligation**,
    Apply **Norm** to **Opposite** of **Case**, obtaining **Minimal-Cases** that are '**Disallowed**', and
      Store the **Minimal-Cases** in **Matching-Cases**, without duplicating any, and store
      The associated **Minimal-Cases, Norm, and** '**Disallowed**' in the **Positive-Set**.
    Apply **Norm** to **Case**, obtaining **Minimal-Cases** that are '**Possible-Conflicts**', and
      Store the **Minimal-Cases** in **Matching-Cases**, without duplicating any, and store
      The associated **Minimal-Cases, Norm, and** '**Possible-Conflicts**'
      In the **Negative-Set**.

Resolve-Conflicts between **Matching-Cases** using the **Positive-Set**, **Negative-Set** obtaining **Decision**.

Norms are used to attribute a normative value to a case. A case can be allowed or disallowed by a norm. A special situation is when the norm does not refer to the case. In that case we say the norm is silent. We formalise the attribution of a norm by distinguishing two parts of a norm:

1. The first part of a norm is the *generic case*. Cases describe one event or situation in the real world. However norms refer to a collection of cases. E.g. a norm in the BV rules is never about the "Queen Mary", it is about passenger ships or cargo ships, bulk carriers etc. The generic case is a description in a formalised way of the cases to which a norm applies. A case can be described as a set of propositions and a generic case as a closed first order formula. If we can derive the (logical) truth of the generic case from

the case, then the norm *fires* or *matches* the case. This is not to say that the propositions of a case 'by itself' have to imply a generic case. The generic case must be derivable from the world knowledge which *results* from asserting the propositions of a case. An example of a case C is:

$C = [starship(Enterprise) \wedge speed(Enterprise, 9)]$

2. The second part of the norm is the type of the normative function. It is a function because it assigns a normative value to the truth of the generic case. A norm can be a prohibition (F), permission (P) or an obligation (O). As an example we could state that it is *dis*allowed to be a 'starship' and travel at a 'speed exceeding warp 9', e.g.:

F_{GC} where $GC = [starship(x) \wedge speed(x,y) \wedge (y > 9)]$

The type of normative function determines the behaviour of the norm when it matches. If the generic case of a norm matches a case and the norm is a prohibition, then the norm classifies the case as disallowed and otherwise is silent. If the generic case of a permission matches a case, then the case is explicitly allowed by the norm. We see a case as violating an obligation in a norm if the *opposite* of the case matches the generic case. The opposite of a case is simply the negation of all the facts. Notice that this implies that an obligation is a prohibition of the opposite case. The connections between generic cases and normative functions is given in Table 1 below.
Note that instead of the KD equivalencies $F(p) \equiv \neg P(p) \equiv O(\neg p)$ (cf. Meyer & Wieringa 1991) the following holds for case C and generic case GC:
$\forall C, F_{GC}(C) = inv(P_{GC}(C)) = O_{opp(GC)}(C)$ where 'inv' is a function that returns the inverse value of a normative qualification, i.e.:
inv(allowed) = disallowed;
inv(disallowed)= allowed; and
inv(silent) = silent.

Deontic Type	*Match Type*	*Qualification returned by function*
F Prohibition	Case = GenericCase	Disallowed
F Prohibition	Other	Silent
P Permission	Case = GenericCase	Allowed
P Permission	Other	Silent
O Obligation	Opposite(Case) = GenericCase	Disallowed
O Obligation	Other	Silent

Table 1.

Apply implements the evaluation task with respect to a single norm and calls a match between each subcase (aspect) of the case and the *generic case* of the norm.

It works as follows:

Function	**Apply** (**Norm**, **Case**) returns **Minimal-Cases**
Input-Roles:	**Norm** – a Norm has a deontic type, links to legal sources and is applicable to a Generic-Case which is a Case with at least one non-grounded proposition. **Case** – see function Assess
Subtasks:	**Match**

Obtain **Generic-Case** of **Norm**, and
For each subset **SubCase** of **Case**
Match **SubCase** to **Generic-Case**,
If **True**,
Store **SubCase** in **Minimal-Cases**,

Removing any **Case** of which the current **SubCase** is a subset.

This function implements the main part of the evaluation subtask. The implementation of match is specific to the properties of the description classifier used for case abstraction. Note that if a backward-chaining system is used, 'Match' triggers Case Abstraction.

After all norms have been applied to the case a conflict resolution process has to follow to identify all cases in which a contradictory classification has been made. Resolve conflicts returns the correct normative qualification, assuming that, if no norm applies, the default is 'Allowed'. It is implemented as follows:

Function	**Resolve-Conflicts** (**Matching-Cases**, **Positive-Set**, **Negative-Set**) returns **Decision**
Input-Roles:	**Matching-Cases** – all minimal cases of the input case that match one or more Generic-Cases. **Positive-Set** – matches that assign a 'positive' normative qualification, i.e. allowed or disallowed. **Negative-Set** – matches that may cause compliance/disaffirmation conflicts.
Subtasks:	**Select-Stronger**, **Select-Wins**

For each **Minimal-Case** in **Matching-Cases**,
If there is a **Norm** in the **Positive-Set** that **Disallows** the **Minimal-Case**, and
The **Stronger-Norm** in the **Positive-Set** that **Disallows** wins
Over the **Stronger-Norm** in the **Positive-Set** that **Allows**, and
The **Stronger-Norm** in the **Positive-Set** that **Disallows** wins
Over the **Stronger-Norm** in the **Negative-Set**,
Store **Disallowed for Case by Norm** in **Decision,**
Else store **Allowed for Case by Norm** in **Decision.**

Stronger selects the locally valid application of a norm in a set. *Wins* does the same for a pair of norms. Both access a static Knowledge Base representing Meta-Legal-Knowledge. The implementation of these two functions depends on the nature and representation of Meta-Legal-Knowledge. Our approach to this issue is discussed in the next section.

3 Cooperative Dialogue and Underspecified Case Descriptions

In a legal case, the assumption is that the case description is fully specified and complete. With complete we mean that all legally relevant facts have been described. Because in a case things have (hypothetically) happened, i.e. the facts of a case are instantiated facts, the case can be described at the lowest level of specificity. Describing e.g. a car accident in terms of two traffic participants (x,y), instead of car(x) (or: Volvo-122) and pedestrian(y), gives a different, incorrect outcome when matched against the traffic code.

This wrong outcome is not due to the assessment algorithm, but due to coding a case in too general terms, so that norms stated at a more specific level will not match. When dealing with 'traffic participants' only a few norms may match: in fact, in the Dutch traffic regulation (RVV-90) not one norm is applicable.[5]

If a case is underspecified it means that terms are used for which a regulation has more specific terms. Underspecification leads potentially to 'overlooking' relevant norms, in particular those (more specific) norms that directly change the legal status of a case: these are 'exceptions'. The nature of exceptions in regulations and handling these implicit conflicts in LIS will be discussed in the next section. For a good formal analysis of exceptions in a rule-based approach, see e.g. Prakken (1993), Verheij (1996).

Thus far we have been talking about cases. However, as we stated in the Introduction, in LIS the specification of complete cases is hardly ever relevant. Getting information at the revenue service about gifts as deductible costs does not imply, nor requires full submission of one's income tax form. Therefore almost by definition the cases in a LIS query are incomplete, or rather: focussed to only one or a few topics. Moreover, many typical LIS questions may not be specific at all. The user, who asks whether it is allowed to have passengers on board a bulk carrier, may not have a specific bulk carrier in mind, but as owner of a fleet of bulk carriers, he may consider additional exploitation of this fleet. Therefore, most LIS queries refer to 'generic' rather than to 'instantiated' cases.[6]

That LIS queries are limited to only a few topics is an advantage rather than a problem for the assessment algorithm: in general, the time spent on abstraction and matching of a case is exponential with respect to the size of a case description. Underspecification is not a problem either: it means less steps in the algorithm to find matching norms. However, underspecification may give rise to another kind of problem: the user may not understand (1) what the required level of specificity is given his intentions and his (often generic) case at hand, and (2) that the outcome should be interpreted with the caution that it is only correct with respect to his specified request.

To prevent the first problem, the CLIME user interface is constructed in such a way that the user, who inputs his request in a semi-free natural language format, is shown more specific options for the terms he used. This "What You See Is What You Meant" technology, developed by the University of Brighton (Power *et al.* 1997), is part of the Query-and-Response-Manager in CLIME (see Figure 1). Moreover, the user may start a follow up dialogue, when the answer to his request is not what he thinks he requested or needed (see Dialogue Manager in Figure 1). However, the user may also be too easily satisfied with an answer, and in particular he may not be aware that further specification may trigger exceptions. Therefore, in cooperative LIS, the user should be warned about potential exceptions.
A simple example may illustrate what is meant. Assume we have the following norms:

1. $F_{[a]}$
2. $P_{[a \wedge b]}$
3. $F_{[a \wedge b \wedge c]}$

5 In the RVV-90 the term 'traffic participant' only occurs in the definitional articles (RVV-91, Art 1): there are no general norms for traffic participants.

6 Somewhat unfortunate, in following Valente (1995)'s terminology 'generic case' refers to (generic, abstract) situation/action specification in norms.

And two meta-norms: norm 3 > ('defeats') 2 and 2 > 1. The meta-norms are an expression of the '*lex specialis derogat legi generali*' principle.[7]

Where 'a', 'b', and 'c' are statements about the world and 'F' and 'P' are normative functions ('forbidden' and 'permitted' respectively). In other words, the generic case 'a' is forbidden (disallowed); the generic case 'a and b' is permitted (allowed) – this can be seen as an exception to the first norm – the generic case 'a and b and c' is forbidden – this can again be seen as an exception to the second norm.

Now a user enters a query: [a ∧ b ∧ c] ? (allowed) where a, b and c are instantiations of the world concepts a, b and c respectively. The following matches will occur:

1. [a] will match the generic case of norm 1, assigning the qualification disallowed
2. [a ∧ b] will match the generic case of norm 2, assigning the qualification allowed
3. [a ∧ b ∧ c] will match the generic case of norm 3, assigning the qualification disallowed

These three normative qualifications conflict, but the two meta-legal principles resolve the conflict, leading to the overall qualification of not-allowed. The same way a query containing [a ∧ b] will lead to the overall qualification of allowed, and a query containing only [a] to disallowed. So far no problem with cases, the user gets a correct qualification of the cases entered.

It is only when we take potential goals of the user in mind, when we want to be *cooperative* that the answer (qualification) may not be the best one for the user. If the user asks for the normative status of case [a] he will correctly get the answer 'not allowed', but he may be helped more with the answer 'not allowed unless b is the case as well'. For the same reasons, we might go on to ask whether 'c' is also relevant, in which case the answer would be 'not allowed' again. Note that a query [d] in the example would result in the qualification 'silent', which is to be translated into the normative default of a regulation: in general this is (weakly) allowed, but for instance, in many safety prescriptions the default is (weakly) disallowed (see for an example of such a domain Hammond *et al*. 1994).

In summary, 'incomplete' or underspecified case descriptions are *not* a problem inherent to normative assessment, but emerge when we want a LIS to be cooperative in dialogue with the user. The main problem for a LIS is to warn the user about potential exceptions.

3.1 Making exceptions explicit

Exceptions cause conflicts between norms, where the generic case of the one implies the generic case of the other, but the normative qualifications differ:[8]

7 This lex specialis principle may be computed in various ways. The most obvious and well known is that subsumed concepts are more specific than the concept that governs these concepts, as e.g. in vehicle -> {car, bicycle, motorcycle,...}. Note that in the computation above the longer list of statements reflects the notion of 'more detail', and is closely related to what we said about the 'completeness' of a case.

8 Other researchers distinguish more types of exceptions, e.g. between rebutting and undercutting defeaters (Pollock 1987; Prakken 1993). In our approach, undercutting de-

General Exception Rule: if there is a norm N_1 with generic case GC_1, and a norm N_2 with generic case GC_2, and $GC_2 \rightarrow GC_1$, and the normative qualification of N_1 is not equal to that of N_2, then there exists an exception relation between N_1 and N_2 such that N_2 is an exception to N_1.

The conflict is a logical conflict. In practice, these conflicts do not pose problems as long as knowledge can be applied to resolve the conflict. In order to be general and formal instead of ad-hoc and content-dependent, this conflict resolution knowledge is often meta-knowledge. The lex specialis principle in law is a typical example of conflict resolution knowledge[9].

Sometimes, regulations contain explicit references as to which norm a norm is an exception (e.g. when a norm states "in contrast to article..." or "...unless article X is applicable." An important heuristic or clue to trace exceptions is the use of P(ermissions) as normative qualification: P-norms are always exceptions to some O- or F-norm. However, not all exceptions are P-norms. For instance $F_{[a \wedge b \wedge c]}$ is an exception to $P_{[a \wedge b]}$. Therefore, we have to infer all *implicit* exceptions for a regulation in order to foresee these in cooperative LIS. Implicit are those exceptions that are not explicitly indicated in the legal text, but are only detected *after* applying all norms.

How do we make the implicit exceptions explicit? One way to do this is on-line: in the assessment algorithm we use meta-knowledge to decide which conflicting norms prevail. However, if we make the assumption that exceptions are not 'case dependent'[10] we may be able to generate all exceptions for a regulation *off-line*. In other words we may be able to compile out all exception relations between norms. This has very important benefits. First, it will speed up the on-line assessment procedure, because no invocation of meta-knowledge is required any more in conflict resolution. Second, one can inspect whether the explicated exceptions in a regulation are the intended ones (see Breuker & den Haan 1996). Third, these explicit exceptions can be used to warn users about potential exceptions to their abstractly specified request in LIS.

In finding the exceptions we can use the *same* mechanisms for conflict resolution that is part of the assessment procedure. There are two ways to do this. The first one is to use the CLIME-LIS, and to store all conflict resolution for future use. This prevents CLIME to make the same inferences more than once with respect to conflicts. The disadvantage is that making exceptions explicit is a long term process in which we cannot be sure that we will capture all exceptions in the long run. Moreover, the regulations may change faster than a (semi-)complete exception structure has been established. Therefore, a second way to generate the exception structure of a regulation is by systematically feeding the LIS assessment submodules with cases/queries in an off-line, batch mode. Ideally, it would be sufficient to vary to all terms and relevant values for terms. However, exceptions may also emerge through world knowledge: a very simple example is:

---

featers either show up as additional (or more specific) circumstances in the generic cases of norms, or as meta-legal knowledge. This is a modelling decision.

9 Exceptions are logical conflicts and conflict resolution is not a logically sound method. However, the cause of using exceptions is not a principled one, but a pragmatic one. It is possible to re-phrase any otherwise consistent law containing exceptions without any logical inconsistencies. However, the result is a far less abstract and almost unreadable version of a regulation: the 'qualification model' (see: Den Haan 1995, Breuker & Den Haan 1996)

10 I.e. the facts of a particular case do not change the exception structure between norms.

1. $F_{[a \wedge b]}$
2. $P_{[a \wedge c]}$
3. $c \rightarrow b$

Of course, all kinds of chains of implications may occur. These indirect relations are part of the world knowledge a regulation is about. However, in practice the principle of *lex specialis* refers to a limited set of computations of implication.[11] For some norms N_{GC1} and N_{GC2}, where GC_1 and GC_2 are generic cases and x and y are propositions such that $x \in GC_1$ and $x \notin GC_2$ and $y \in GC_2$ and $y \notin GC_1$ and $\{x,y\} = ((GC_1 \cup GC_2) - (GC_1 \cup_{disjoint} GC_2))$[12], and T_{world} is a theory about the world, some exception could be computed as follows:

Subsumption: If y is-a subtype of x, then N_{GC2} is an exception to N_{GC1} because 'y is-a subtype of x'. An example:

N_{GC1}: $F_{[cargo\text{-}ship(x) \wedge nr\text{-}passengers(x, y) \wedge y>13]}$
N_{GC2}: $P_{[liquid\text{-}gas\text{-}carrier(x) \wedge nr\text{-}passengers(x, y) \wedge y>13]}$
T_{world}: liquid-gas-carrier(x) is-a cargo-ship(x)

Instantiation: If x contains an open variable ?v and y unifies with x, then N_{GC2} is an exception to N_{GC1} because 'y realises x'. An Example:

N_{GC1}: $F_{[cargo\text{-}ship(x) \wedge nr\text{-}passengers(x, y) \wedge y>13 \wedge location(x, ?v)]}$
N_{GC2}: $P_{[cargo\text{-}ship (x) \wedge nr\text{-}passengers(x, y) \wedge y>13 \wedge location(x, \text{“port of Rotterdam”})]}$

More conditions: If $(GC_2 \cup GC_{exc} = GC_1)$, then N_{GC2} is an exception to N_{GC1} because 'GC_{exc} adds detail'. An example:

N_{GC1}: $F_{[cargo\text{-}ship(x) \wedge nr\text{-}passengers(x, y) \wedge y>13]}$
N_{GC2}: $P_{[cargo\text{-}ship (x) \wedge nr\text{-}passengers(x, y) \wedge y>13 \wedge location(x, z) \wedge harbour(z)]}$

Part-whole: If is y part-of x, then N_{GC2} is an exception to N_{GC1} because 'y is part of x'. An example:

N_{GC1}: $P_{[ship(x) \wedge fire\text{-}pump(y) \wedge in(x, y)]}$
N_{GC2}: $F_{[machine\text{-}room (x) \wedge fire\text{-}pump(y) \wedge in(x, y)]}$
T_{world} : machine-room part-of ship

4 Conclusions and discussion

Legal Information Serving (LIS) is different from information retrieval in general, because obtaining the right information about legal or normative issues invariably involves assessing the legal status of a situation descrip-

11 Often other principles than lex specialis are mentioned to resolve conflicts between norms, like lex posterior and lex superior. However, we suspect that these principles are not functional for conflict resolution and the legal use of exceptions, but about the validity of a regulation, respectively to limit the scope of lex specialis notions (see Elhadj *et al.*, in preparation). Exceptions are a notion that is not exclusive to normative reasoning. All kinds of knowledge are understood or coded in such terms, as e.g. the famous penguins which are birds 'except' that they cannot fly. Invariably it seems that specificity is the rule to 'protect' exceptions to be overwritten by more abstract knowledge.

12 I.e. the rest of the generic cases are equal.

tion in a query. In many respects, this assessment procedure is identical to evaluating legal cases. As a consequence, the results and procedures in LIS differ dramatically from those obtained, respectively used in traditional information retrieval, in particular text retrieval. While in text retrieval there is strong negative relationship between the number of cues in a query and the number of matching documents or pieces of text, the opposite holds for LIS as discussed and explained here. In fact, we work from the assumption that the traditional metrics used in information retrieval – recall and precision – are both 100% in LIS. The reasons are many, but the main ones are the following:

- There is a full specification of terminology (ontology) required as a knowledge base, and this terminology is the (only) one that is to be employed for accessing queries. This is similar to using conceptual retrieval front ends to text bases. This is to be viewed more as support for the user, than a restriction in expressivity.
- The assessment function in LIS is not concerned with matching terms, but with matching situation descriptions (cases) and solves the deontics involved in applying norms to cases. The trace of assessment can be used to justify and explain the deontic solution found.
- Implied knowledge is taken into account in the assessment procedure. It may be impossible to take all possible implied knowledge into account, because the world is infinitely complex and processing combinatorics prevent exhaustive inferencing. However, a LIS provides a far more precise matching and retrieval function than text-based retrieval and for that matter also a much more reliable and explicit method than human experts. A LIS may fail in a predictable way, while human experts have to rely on intransparent methods.

In this paper we have presented a LIS, based upon the ON-LINE system (Valente 1995) as a module that is a central part of the CLIME multi-agent architecture. The assessment algorithms of this LIS are explained. In a first version, this LIS is implemented as a modification of ON-LINE in Common-LISP, using the LOOM description classifier (McGregor 1991) as the main knowledge representation service. The CLIME project provides a good opportunity to test our LIS on a large and realistic (para)legal domain. We have already changed the algorithms to increase efficiency and to provide more intermediate reasoning results for explanation purposes.

Moreover, we found out that LIS differs from straightforward assessment of legal cases, because a query is in general focussed on a particular topic. The topic may be underspecified, so that special care has to be taken that the user will not misunderstand the scope of the results. As a consequence, a cooperative dialogue may not only prevent underspecification of queries, but is sensible to point to potential exceptions to the result presented. We discussed the nature of these exceptions and how an exception structure could be generated off-line, using part of the assessment modules. This functionality will be put into the CLIME 'Legal Encoding Tools' (Figure 1).

The advantages of LIS based on normative assessment may be obvious in its results, but there is a price to pay. This price is that one is committed to:

- **Knowledge modelling.** Where text-based stores require hardly any structuring beyond text-structural features, as e.g. provided by SGML or XML, all terms and implied terms of the domain have to be modelled as conceptual definitions and their relations. In this modelling enterprise, most implicit and common sense knowledge that is part of the domain has to be modelled as well.
- **Special inference services.** Thus far we have evaded the question what is exactly meant by 'implied knowledge'. Implied knowledge may come from or cover very heterogeneous domains of inference. As every domain understanding is based upon abstraction, type and part-of hierarchies (typology and mereology) are universally used and part of knowledge representation services. Causality, time and space are next candidates that are needed to derive implications in most domains. The use of deontic operators in the assessment function can be viewed as such a service that handles inferencing in normative domains. Many problems are related to the development of these services: they are a main focus of fundamental research in AI. However, it is not only the specification and algorithmisation of these basic knowledge inference calculus, that limits our optimism of fully correct and valid LIS, but also the fact that if these services are available, the computational overheads for exhaustive inferencing are prohibitive. Which (specific) services are needed, and in what variety, is often dependent on the domain. In most domains there is a focus on the kind of knowledge use that can be captured by 'core ontologies' (Valente & Breuker, 1996). In traffic regulations, reasoning about time and causality is not required, but some rudimentary spatial reasoning that infers that left is not-right etc. is needed. In the domain of ship classification we will need inferencing about processes, causality, time and space, besides the usual abstractions (type and part-whole hierarchies).

Acknowledgements

CLIME is sponsored by the EC ESPRIT programme with project number P25.414. The CLIME partners are: British Maritime Technologies (UK), University of Brighton (UK); Bureau Veritas (France); TXT (Italy); and University of Amsterdam (Netherlands).

References

Blair & Maron (1985)
Blair, D.C. & M.E. Maron, An evaluation of retrieval effectiveness for a full-text document retrieval system, *Communications of the ACM* 28(3), 1985, pp. 289-299.

Breuker (1992)
Breuker, J.A., On Legal Information Serving, in Grütters, C.A.F.M. *et al.* (eds.), *Legal knowledge based systems: Information Technology and Law, JURIX '92*, Lelystad: Koninklijke Vermande 1992, pp. 93-102.

Breuker & den Haan (1991)
Breuker, J. A. & N. den Haan, Separating regulation from world knowledge: where is the logic?, in M. Sergot (Ed.), *Proceedings of the 4th International Conference on AI and Law*, New York: ACM 1991, pp. 41-51.

Breuker & den Haan (1996)
Breuker, J. A. & N. den Haan, Construction Normative Rules, in R. van Kralingen *et al.* (ed.), *Legal knowledge based systems: Foundations of Legal Knowledge Based Systems, JURIX '96*, Tilburg: TUP 1996, pp. 41-56.

Elhadj *et al.*
Elhadj, A., Brouwer, B. and J.A. Breuker, (in preparation), *Modelling Normative Knowledge for the Analysis of Norm Conflict*, internal publication.

Den Haan (1996)
Haan, N. den, *Automated Legal Reasoning*, dissertation, Amsterdam: University of Amsterdam 1996.

Hammond *et al.* (1994)
Hammond, P., J. Wyatt, and A. Harris, Drafting protocols, certifying clinical trial designs and monitoring compliance. *Proceedings of the ECAI workshop on Artificial Normative Reasoning*, Amsterdam 1994, pp. 124-131.

MacGregor (1991)
MacGregor, R. (1991). Inside the LOOM classifier. *SIGART Bulletin, 2*(3):70-76.

Meyer & Wieringa (1991)
Meyer, J.-J. & R. Wieringa, Deontic Logic: A Concise Overview, in *Proceedings of the 1st International Workshop on Deontic Logics in Computer Science*, 1991, pp. 2-14.

Power *et al.* (1997)
Power, R., D. Scott, and R. Evans, *What You See Is What You Meant: direct knowledge editing with natural language feedback*, Technical Report, ITRI-97-03, University of Brighton 1997.

Pollock (1987)
Pollock, J.L., Defeasible Reasoning, *Cognitive Science* 11, 1987, pp. 481-518.

Prakken (1993)
Prakken, H., *Logical Tools for Modelling Legal Argument*, dissertation, Amsterdam: Vrije Universiteit Amsterdam 1993.

Smith *et al.* (1995)
Smith, J.C., D. Gelbart, K. MacCrimmon, B. Atherton, J. McClean, M. Shinehoft, and L. Quintana, Artificial Intelligence and Legal Discourse: The Flexlaw Legal Text Management System, *Artificial Intelligence and Law* 3(1-2), 1995, pp. 55-95.

Turtle (1995)
Turtle, H., Text retrieval in the legal world, *Artificial Intelligence and Law*, 3(1-2), 1995, pp. 5-54.

Valente & Löckenhof (1994)
Valente, A. & C. Löckenhof, Assessment, in J. A Breuker and W. van de Velde (Eds.), *The CommonKADS Library for Expertise Modelling*, Amsterdam: IOS Press 1994.

Valente (1995)
Valente, A., *Legal Knowledge Engineering. A Modelling Approach*, dissertation, University of Amsterdam, Amsterdam: IOS Press 1995.

Valente & Breuker (1996)
Valente, A. & J.A. Breuker, Towards principled core ontologies, in B.R. Gaines and M. Musen (eds.), *Proceedings of 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996, pp. 301-320.

Verheij (1996)
Verheij, B., *Rules, Reasons, Arguments. Formal studies of argumentation and defeat*, dissertation, Maastricht: University of Maastricht 1996.