A CNL for Contract-Oriented Diagrams

John J. Camilleri, Gabriele Paganelli, and Gerardo Schneider

Department of Computer Science and Engineering,
Chalmers University of Technology and the University of Gothenburg, Sweden
{john.j.camilleri,gerardo}@cse.gu.se, gabpag@chalmers.se

Abstract. We present a first step towards a framework for defining and manipulating normative documents or contracts described as *Contract-Oriented (C-O) Diagrams*. These diagrams provide a visual representation for such texts, giving the possibility to express a signatory's obligations, permissions and prohibitions, with or without timing constraints, as well as the penalties resulting from the non-fulfilment of a contract. This work presents a CNL for verbalising *C-O Diagrams*, a web-based tool allowing editing in this CNL, and another for visualising and manipulating the diagrams interactively. We then show how these proof-of-concept tools can be used by applying them to a small example.

Keywords: normative texts, electronic contracts, c-o diagrams, controlled natural language, grammatical framework.

1 Introduction and Background

Formally modelling normative texts such as legal contracts and regulations is not new. But the separation between logical representations and the original natural language texts is still great. CNLs can be particularly useful for specific domains where the coverage of full language is not needed, or at least when it is possible to abstract away from some irrelevant aspects.

In this work we take the *C-O Diagram* formalism for normative documents [1], which specifies a visual representation and logical syntax for the formalism, together with a translation into timed automata. This allows model checking to be performed on the modelled contracts. Our concern here is how to ease the process of writing and working with such models, which we do by defining a CNL which can translate unambiguously into a *C-O Diagram*. Concretely, the contributions of our paper are the following:

1. Syntactical extensions to *C-O Diagrams* concerning executed actions and cross-references (section 2.3);
2. A CNL for *C-O Diagrams* implemented using the Grammatical Framework (GF), precisely mapping to the formal grammar of the diagrams (section 3).
3. Tools for visualising and manipulating *C-O Diagrams* (section 2):
   (a) A web-based visual editor for *C-O Diagrams*;
   (b) A web-based CNL editor with real-time validation;
   (c) An XML format COML used as a storage and interchange format.

B. Davis et al. (Eds.): CNL 2014, LNAI 8625, pp. 135–146, 2014.

$$
\begin{aligned}
C &:= (agent, name, g, tr, O(C_2), R) \\
&\quad \mid (agent, name, g, tr, P(C_2), \epsilon) \\
&\quad \mid (agent, name, g, tr, F(C_2), R) \\
&\quad \mid (\epsilon, name, g, tr, C_1, \epsilon) \\
C_1 &:= C\ (And\ C)^+ \mid C\ (Or\ C)^+ \mid C\ (Seq\ C)^+ \mid Rep(C) \\
C_2 &:= a \mid C_3\ (And\ C_3)^+ \mid C_3\ (Or\ C_3)^+ \mid C_3\ (Seq\ C_3)^+ \\
C_3 &:= (\epsilon, name, \epsilon, \epsilon, C_2, \epsilon) \\
R &:= C \mid \epsilon
\end{aligned}
$$

Fig. 1. Formal syntax of *C-O Diagrams* [1]

We also present a small example to show our CNL in practice (section 4) and an an initial evaluation of the CNL (section 5). In what follows we provide some background for *C-O Diagrams* and GF.

1.1 *C-O Diagrams*

Introduced by Martínez et al. [2], *C-O Diagrams* provide a means for visualising normative texts containing the modalities of obligation, permission and prohibition. They allow the representation of complex clauses describing these norms for different signatories, as well as *reparations* describing what happens when obligations and prohibitions are not fulfilled.

The basic element is the *box* (see Fig. 4), representing a basic contract clause. A box has four components: i) *guards* specify the conditions for enacting the clause; ii) *time restrictions* restrict the time frame during which the contract clause must be satisfied; iii) the *propositional content* of a box specifies a modality applied over actions, and/or the actions themselves; iv) a *reparation*, if specified, is a reference to another contract that must be satisfied in case the main norm is not. Each box also has an *agent* indicating the performer of the action, and a unique *name* used for referencing purposes. Boxes can be expanded by using three kinds of refinement: *conjunction*, *choice*, and *sequencing*.

The diagrams have a formal definition given by the syntax shown in Fig. 1. For an example of a *C-O Diagram*, see Fig. 5 (this example will be explained in more detail in section 4).

1.2 Grammatical Framework

GF [3] is both a language for multilingual grammar development and a type-theoretical logical framework, which provides a mechanism for mapping abstract logical expressions to a concrete language. With GF, the language-independent structure of a domain can be encoded in the abstract syntax, while language-specific features can be defined in potentially multiple concrete languages.

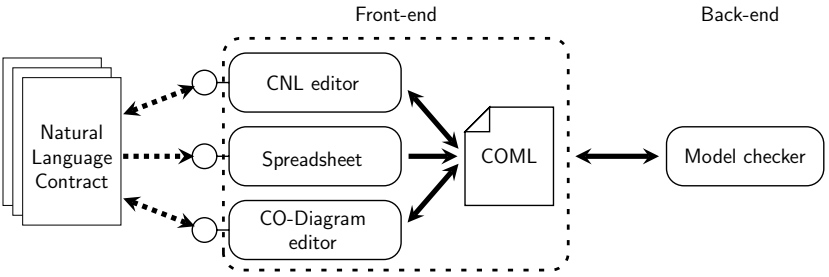


Fig. 2. The contract processing framework. Dashed arrows represent manual interaction, solid ones automated interaction.

Since GF provides both a *parser* and *linearisers* between concrete and abstract languages, multi-lingual translation can be achieved using the abstract syntax as an interlingua.

GF also comes with a standard library called the *Resource Grammar Library* (RGL) [4]. Sharing a common abstract syntax, this library contains implementations of over 30 natural languages. Each resource grammar deals with low-level language-specific details such as word order and agreement. The general linguistic descriptions in the RGL can be accessed by using a common language-independent API. This work uses the English resource grammar, simplifying development and making it easier to port the system to other languages.

2 Implementation

2.1 Architecture

The contract processing framework presented in this work is depicted in Fig. 2. There is a *front-end* concerned with the modelling of contracts in a formal representation, and a *back-end* which uses formal methods to detect conflicts, verify properties, and process queries about the modelled contract. The back-end of our system is still under development, and involves the automatic translation of contracts into timed automata which can be processed using the UPPAAL tool [5].

The front-end, which is the focus of this paper, is a collection of web tools that communicate using our XML format named COML.[1] This format closely resembles the *C-O Diagram* syntax (Fig. 1). The tools in our system allow a contract to be expressed as a CNL text, spreadsheet, and *C-O Diagram*. Any modification in the diagram is automatically verbalised in CNL and vice versa. A properly formatted spreadsheet may be converted to a COML file readable by the other editors. These tools use HTML5 [6] local storage for exchanging data.

[1] An example of the format, together with an XSD schema defining the structure, is available online at `http://remu.grammaticalframework.org/contracts`

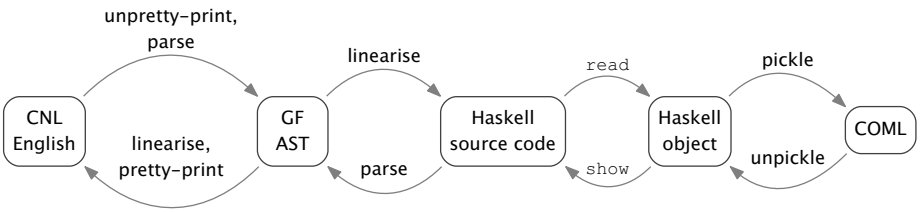


Fig. 3. Conversion process from CNL to COML and back

Translation Process. The host language for all our tools is Haskell, which allows us to define a central data type precisely reflecting the formal *C-O Diagram* grammar (Fig. 1). We also define an abstract syntax in GF which closely matches this data type, and translate between CNL and Haskell source code via two concrete syntaxes. As an additional processing step after linearisation with GF, the generated output is passed through a pretty-printer, adding newlines and indentations as necessary (section 3.2). The Haskell source code generated by GF can be converted to and from actual objects by deriving the standard `Show` and `Read` type classes. Conversion to the COML format is then handled by the HXT library, which generates both a parser and generator from a single *pickler* function. The entire process is summarised in Fig. 3.

2.2 Editing Tools

The visual editor allows users to visually construct and edit *C-O Diagrams* of the type seen in section 4. It makes use of the mxGraph JavaScript library providing the components of the visual language and several facilities such as converting and sending the diagram to the CNL editor, validation of the diagram, conversion to PDF and PNG format.

The editor for CNL texts uses the ACE JavaScript library to provide a text-editing interface within the browser. The user can verify that their CNL input is valid with respect to grammar, by calling the GF web service. Errors in the CNL are highlighted to the user. A valid text can then be translated into COML with the push of a button.

2.3 Syntactic Extensions to *C-O Diagrams*

This work also contributes two extensions to *C-O Diagram* formalism:

1. To the grammar of guards, we have add a new condition on whether an action a has been performed ($done(a)$);
2. We add also a new kind of box for cross-references. This enhances *C-O Diagrams* with the possibility to have a more modular way to "jump" to other clauses. This is useful for instance when referring to *reparations*, and to allow more general cases of "repetition".

Our tool framework also includes some additional features for facilitating the manipulation of *C-O Diagrams*. The most relevant to the current work is the automatic generation of clocks for each action. This is done by implicitly creating a clock `t_name` for each box `name`. When the action or sub-contract `name` is completed, the clock `t_name` is reset, allowing the user to refer to the time elapsed since the completion of a particular box.

3 CNL

This section describes some of the notable design features of our CNL. Examples of the CNL can be found in the example in section 4.

3.1 Grammar

The GF abstract syntax matches closely the Haskell data type designed for *C-O Diagrams*, with changes only made to accommodate GF's particular limitations. Optional arguments such as guards are modelled with a category `MaybeGuard` having two constructors `noGuard` and `justGuard`, where the latter is a function taking a list of guards, `[Guard]`. The same solution applies to timing constraints. Since GF does not have type polymorphism, it is not possible to have a generalised `Maybe` type as in Haskell. To avoid ambiguity, lists themselves cannot be empty; the base constructor is for a singleton list.

In addition to this core abstract syntax covering the *C-O Diagram* syntax, the GF grammar also imports phrase-building functions from the RGL, as well as the large-scale English dictionary `DictEng` containing over 64,000 entries.

3.2 Language Features

Contract Clauses. A simple contract verbalisation consists of an **agent**, **modality**, and an **action**, corresponding to the standard subject, verb and object of predication. The modalities of obligation, permission and prohibition are respectively indicated by the keywords `required`, `may` (or `allowed` when referring to complex actions) and `mustn't` (or `forbidden`).

Agents are noun phrases (NP), while actions are formed from either an intransitive verb (V), or a transitive verb (V2) with an NP representing the object. This means that every agent and action must be a grammatically-correct NP/VP, built from lexical entries found in the dictionary and phrase-level functions in the RGL. This allows us to correctly inflect the modal verb according to the agent (subject) of the clause:

```
1 : Mary is required to pay
2 : Mary and John are required to pay
```

Constraints. The arithmetic in the *C-O Diagram* grammar covering guards and timing restrictions is very general, allowing the usual comparison operators between variable or clock names and values, combined with operators for negation and conjunction. Their linearisation can be seen in line 9 of Fig. 6.

Each contract clause in a *C-O Diagram* has an implicit timer associated with it called `t_name`, which is reset when the contract it refers to is completed. These can be referred to in any timing restriction, effectively achieving relative timing constraints by referring to the time elapsed since the completion of another contract.

Conjunction. Multiple contracts can be combined by conjunction, choice and sequencing. GF abstract syntax supports lists, but linearising them into CNL requires special attention. Lists of length greater than two must be bulleted and indented, with the entire block prefixed with a corresponding keyword:

```
1 : all of
  - 1a : Mary may eat a bagel
  - 1b : John is required to pay
```

When unpretty-printed prior to parsing, this is converted to:

```
1 : all of { - 1a : Mary ... bagel - 1b : John ... pay }
```

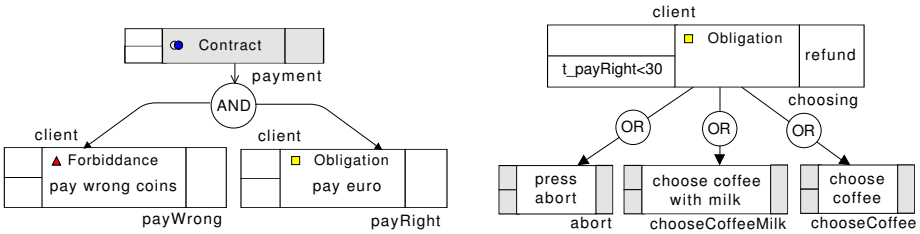
For a combination of exactly two contracts, the user has the choice to use the bulleted syntax above, or inline the clauses directly using the appropriate combinator, e.g. `or` for choice. This applies to combination of contracts, actions and even guards and timing restrictions.

In the case of actions the syntax is slightly different since there is a single modality applied to multiple actions. Here, the actions appear in the infinitive form and the combination operator appears at the end of each line (except the final one):

```
2 : Mary is allowed
  - 2a : to pay , or
  - 2b : to eat a bagel
```

This list syntax allows for nesting to an arbitrary depth.

Names. The *C-O Diagram* grammar dictates that all contract clauses should have a name (*label*). These provide modularity by allowing referencing of other clauses by label, e.g. in reparations and relative timing constraints. Since the CNL cannot be lossy with respect to the COML, these labels appear in the CNL linearisation too (see Fig. 6). Clause names are free strings, but must not contain any spaces. This avoids the need for double quotes in the CNL. These labels do reduce naturalness somewhat, but we believe that this inconvenience can be minimised with the right editing tool.



(a) Payment options (b) Choices in selection

```
1  payment :
2    payWrong : client mustn't pay wrong coins otherwise see refund and
3    payRight : client is required to pay euro
4  choosing : when clock t_payRight less than 30 client is required
5      - abort : to press abort , or
6      - chooseCoffeeMilk : to choose coffee with milk , or
7      - chooseCoffee : to choose coffee  otherwise see refund
```

Fig. 4. Different kinds of complex contracts and their verbalisation

4 Coffee Machine Example

A user Eva must analyse the following description of the operation of a coffee machine, and construct a formal model for it. She will do this interactively, switching between editing the CNL and the visual representation.

> *To order a drink the client inputs money and selects a drink. Coffee can be chosen either with or without milk. The machine proceeds to pour the selected drink, provided the money paid covers its price, returning any change. The client is notified if more money is needed; they may then add more coins or cancel the order. If the order is cancelled or nothing happens after 30 seconds, the money is returned. The machine only accepts euro coins.*

Eva first needs to identify: i) the *actors* (client and machine), ii) the *actions* (pay, accept, select, pour, refund), iii) and the *objects* (beverage, money, timer). The first sentence suggests that to obtain a drink the client *must* insert coins. Eva therefore drops an `obligation` box in the diagram editor and fills the name, agent and action fields. Only accepting euro is modelled as a prohibition to the client using a `forbiddance` box. The two boxes are linked using a `contract` box as shown in Fig. 4a.

Eva now wants to model the choice of beverage, and the possibility the aborting of the process. She creates an `obligation` box named `choosing`, adding the timed constraint `t_payRight < 30` to model the 30 second timeout. She then appends two action boxes using the `or` refinement, corresponding to the choice of drinks (see Fig. 4b). Eva translates the diagram to CNL and modifies the text, adding the action `abort : to press abort` as a refinement of `choosing`. The result is shown in line 4 of Fig. 6.

The *C-O Diagram* for the final contract is shown in Fig. 5. It includes the handling of the `abort` action and gives an ordering to the sub-contracts. Note how there are two separate contracts in the CNL verbalisation: `coffeeMachine` and `refund`, the latter being referenced as a reparation of the former.

The *C-O Diagram* editor allows changes to be made locally while retaining the contract's overall structure, for instance inserting an additional option for a new beverage. The CNL editor is instead most practical for replicating patterns or creating large structures such as sequences of clauses, that are faster to outline in text and rather tedious to arrange in a visual language. The two editors have the same expressive power and the user can switch between them as they please.

5 Evaluation

5.1 Metrics

The GF abstract syntax for basic *C-O Diagrams* contains 48 rules, although the inclusion of large parts of the RGL for phrase formation pushes this number up to 251. Including the large-scale English dictionary inflates the grammar to 65,174 rules. As a comparison, a previous similar work on a CNL for the contract logic $\mathcal{CL}$ [7] had a GF grammar of 27 rules, or 2,987 when including a small verb lexicon.

5.2 Classification

Kuhn suggests the PENS scheme for the classification of CNLs [8]. We would classify the CNL presented in the current work as $P^5E^1N^{2\text{-}3}S^4$, F W D A. P (precision) is high since we are implementing a formal grammar; E (expressivity) is low since the CNL is restricted to the expressivity of the formalism; N (naturalness) is low as the overall structure is dominated with clause labels and bullets; S (simplicity) is high because the language can be concisely described as a GF grammar. In terms of CNL properties, this is a written (W) language for formal representation (F), originating from academia (A) for use in a specific domain (D).

The P, E and S scores are in line with the problem of verbalising a formal system. The low N score of between 2–3 is however the greatest concern with this CNL. This is attributable to a sentence structure is not entirely natural, somewhat idiosyncratic punctuation, and a bulleted structure that could restrict readability. While these features threaten the naturalness of the CNL in raw form, we believe that sufficiently developed editing tools have a large part to play in dealing with the structural restrictions of this language. Concretely, the ability to hide clause labels and fold away bulleted items can significantly make this CNL easier to read and work with.

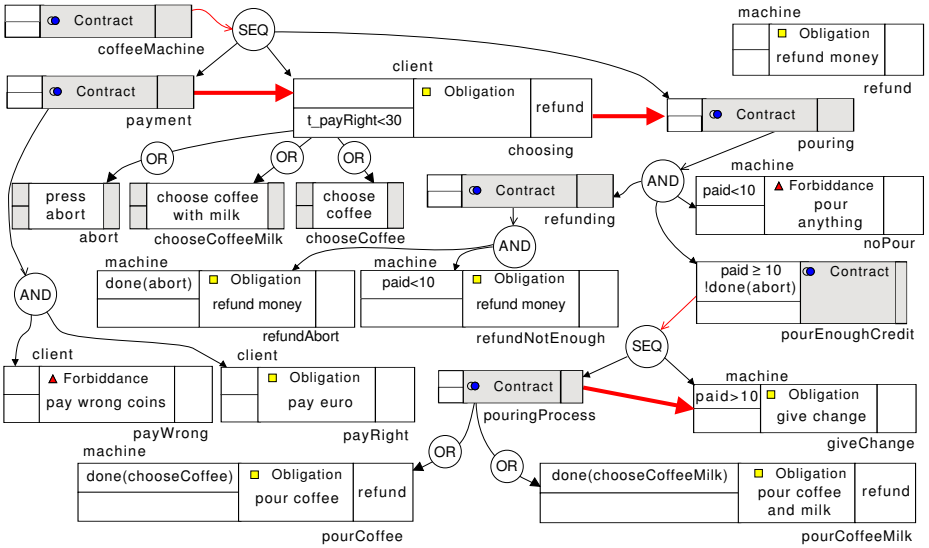


Fig. 5. The complete *C-O Diagram* for the coffee machine example

```
coffeeMachine : the following, in order
  - payment : payWrong : client mustn't pay wrong coins otherwise
      see refund and payRight : client is required to pay euro
  - choosing : when clock t_payRight less than 30 client is required
    - abort : to press abort , or
    - chooseCoffeeMilk : to choose coffee with milk , or
    - chooseCoffee : to choose coffee  otherwise see refund
  - pouring : all of
    - pourEnoughCredit : when abort is not done and variable paid
        not less than 10 first pouringProcess : pourCoffee : if
        chooseCoffee is done machine is required to pour coffee
        otherwise see refund or pourCoffeeMilk : if chooseCoffeeMilk
         is done machine is required to pour coffee and milk
        otherwise see refund , then giveChange : if variable paid
        greater than 10 machine is required to give change
    - noPour : if variable paid less than 10 machine mustn't pour
        anything
    - refunding : refundNotEnough : if variable paid less than 10
        machine is required to refund money and refundAbort : if
        abort is done machine is required to refund money
refund : machine is required to refund money
```

Fig. 6. The final verbalisation for the coffee machine example

6 Related Work

C-O Diagrams may be seen as a generalisation of $\mathcal{CL}$ [9,10,11] in terms of expressivity.[2] In a previous work, Angelov et al. introduced a CNL for $\mathcal{CL}$ in the framework AnaCon [7]. AnaCon allows for the verification of conflicts (contradictory obligations, permissions and prohibitions) in normative texts using the CLAN tool [12]. The biggest difference between AnaCon and the current work, besides the underlying logical formalism, is that we treat agents and actions as linguistic categories, and not as simple strings. This enables better agreement in the CNL which lends itself to more natural verbalisations, as well as making it easier to translate the CNL into other natural languages. We also introduce the special treatment of two-item co-ordination, and have a more general handling of lists as required by our more expressive target language.

Attempto Controlled English (ACE) [13] is a controlled natural language for universal domain-independent use. It comes with a parser to discourse representation structures and a first-order reasoner RACE [14]. The biggest distinction here is that our language is specifically tailored for the description of normative texts, whereas ACE is generic. ACE also attempts to perform full sentence analysis, which is not necessary in our case since we are strictly limited to the semantic expressivity of the *C-O Diagram* formalism.

Our CNL editor tool currently only has a basic user interface (UI). As already noted however, it is clear that UI plays a huge role in the effectiveness of a CNL. While our initial prototypes have only limited features in this regard, we point to the ACE Editor, AceRules and AceWiki tools described in [15] as excellent examples of how UI design can help towards solving the problems of writability with CNLs.

7 Conclusion

This work describes the first version of a CNL for the *C-O Diagram* formalism, together with web-based tools for building models of real-world contracts.

The spreadsheet format mentioned in Fig. 2 was not covered in this paper, but we aim to make it another entry point into our system. This format shows the mapping between original text and formal model by splitting the relevant information about modality, agent, object and constraints into separate columns. As an initial step, the input text can be separated into one sentence per row, and for each row the remaining cells can be semi-automatically filled-in using machine learning techniques. This will help the first part of the modelling process by generating a skeleton contract which the user can begin with.

We plan to extend the CNL and *C-O Diagram* editors with better user interfaces for easing the task of learning to use the respective representations and helping with the debugging of model errors. We expect to have more integration

[2] On the other hand, $\mathcal{CL}$ has three different formal semantics: an encoding into the μ-calculus, a trace semantics, and a Kripke-semantics.

between the two applications, in particular the ability to focus on smaller subsections of a contract and see both views in parallel. While the CNL editor already has basic input completion, it must be improvemed such that completion of functional keywords and content words are handled separately. Syntax highlighting for indicating the different constituents in a clause will also be implemented.

We currently use the RGL *as is* for parsing agents and actions without writing any specific constructors for them, which creates the potential for ambiguity. While this does not effect the conversion process, ambiguity is still an undesirable feature to have in a CNL. Future versions of the grammar will contain a more precise selection of functions for phrase construction, in order to minimise ambiguity.

Finally, it is already clear from the shallow evaluation in section 5 that the CNL presented here suffers from some unnaturalness. This can to some extent be improved by simple techniques, such as adding variants for keywords and phrase construction. Other features of the *C-O Diagram* formalism however are harder to linearise naturally, in particular mandatory clause labels and arbitrarily nested lists of constraints and actions. We see this CNL as only the first step in a larger framework for working with electronic contracts, which must eventually be more rigorously evaluated through a controlled usability study.

Acknowledgements. The authors wish to thank the Swedish Research Council for financial support under grant nr. 2012-5746. We are also very grateful to the anonymous reviewers for their suggestions, in particular with regards to CNL evaluation and classification using the PENS scheme.

References

1. Díaz, G., Cambronero, M.E., Martínez, E., Schneider, G.: Specification and Verification of Normative texts using C-O Diagrams. IEEE Transactions on Software Engineering (2013)
2. Martínez, E., Cambronero, E., Diaz, G., Schneider, G.: A Model for Visual Specification of e-Contracts. In: IEEE SCC 2010, pp. 1–8. IEEE Computer Society (2010)
3. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011)
4. Ranta, A.: The GF Resource Grammar Library. Linguistic Issues in Language Technology 2(2) (2009)
5. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (2014)
6. Navara, E.D., Pfeiffer, S., Berjon, R., Faulkner, S., Leithead, T., O'Connor, E.: HTML5. Candidate recommendation, W3C (2014), `http://www.w3.org/TR/2014/CR-html5-20140204/`
7. Angelov, K., Camilleri, J.J., Schneider, G.: A Framework for Conflict Analysis of Normative Texts Written in Controlled Natural Language. Journal of Logic and Algebraic Programming 82(5-7), 216–240 (2013)
8. Kuhn, T.: A Survey and Classification of Controlled Natural Languages. Computational Linguistics 40(1) (2014)

9. Prisacariu, C., Schneider, G.: A Formal Language for Electronic Contracts. In: Bonsangue, M.M., Johnsen, E.B. (eds.) FMOODS 2007. LNCS, vol. 4468, pp. 174–189. Springer, Heidelberg (2007)
10. Prisacariu, C., Schneider, G.: $\mathcal{CL}$: An Action-Based Logic for Reasoning about Contracts. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS, vol. 5514, pp. 335–349. Springer, Heidelberg (2009)
11. Prisacariu, C., Schneider, G.: A dynamic deontic logic for complex contracts. Journal of Logic and Algebraic Programming 81(4), 458–490 (2012)
12. Fenech, S., Pace, G.J., Schneider, G.: CLAN: A Tool for Contract Analysis and Conflict Discovery. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 90–96. Springer, Heidelberg (2009)
13. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English (ACE) Language Manual, Version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich (1999)
14. Fuchs, N.E.: First-Order Reasoning for Attempto Controlled English. In: Rosner, M., Fuchs, N.E. (eds.) CNL 2010. LNCS, vol. 7175, pp. 73–94. Springer, Heidelberg (2012)
15. Kuhn, T.: Controlled English for Knowledge Representation. Doctoral thesis, University of Zurich (2010)