Universal Chemical Markup (UCM) - A new format for common chemical data

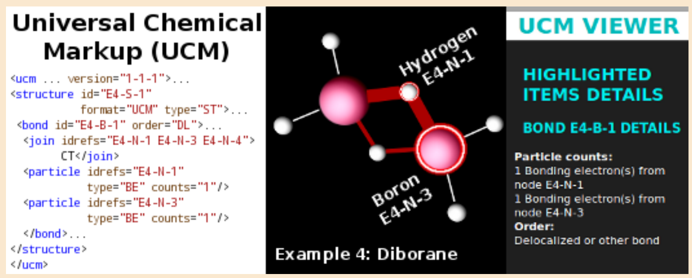
Background: We wish to introduce a new chemical format called UCM (Universal Chemical Markup). The format is based on XML (Extensible Markup Language) and its first version focuses on recording chemical structures and their properties. **Results:** UCM currently supports structures containing isotopes, ions and various types of bonding including delocalized bonds. Properties can be expressed by combining UCM with UnitsML (Units Markup Language). Using UnitsML one defines quantities with scientific units, and then refers to them in UCM when recording property values. Users can also add literature references with BibTeXML (BibTeX Markup Language) and annotate the recorded data using plain text or XHTML (Extensible Hypertext Markup Language) descriptions. In contrast to presently available general-purpose chemical formats, UCM offers built-in validation, which combines both grammar and pattern-based XML schema languages. Thus, all recorded data can be precisely validated by UCM schemas in standard XML validators. **Conclusions:** We developed the structure for UCM from scratch on the basis of an analysis described in our previous article. Starting from scratch allowed us to integrate BibTeXML, UnitsML and XHTML as well as chemical line notations and identifiers into UCM. It also helped us to avoid unnecessary redundant parts and create the implementation that aims to minimize ambiguity and is designed to be easily extensible in the future.

Universal Chemical Markup (UCM) - A new format for common chemical data

Jan Mokrý[1] **and Miloslav Nič**[2]

[1]**Department of Inorganic Chemistry, University of Chemistry and Technology Prague, Technická 5, 166 28, Prague 6, Czech Republic**
[2]**Department of Software Engineering, Czech Technical University in Prague, Thákurova 9, 160 00, Prague 6, Czech Republic**

ABSTRACT



Background
We wish to introduce a new chemical format called UCM (Universal Chemical Markup). The format is based on XML (Extensible Markup Language) and its first version focuses on recording chemical structures and their properties.

Results
UCM currently supports structures containing isotopes, ions and various types of bonding including delocalized bonds. Properties can be expressed by combining UCM with UnitsML (Units Markup Language). Using UnitsML one defines quantities with scientific units, and then refers to them in UCM when recording property values. Users can also add literature references with BibTeXML (BibTeX Markup Language) and annotate the recorded data using plain text or XHTML (Extensible Hypertext Markup Language) descriptions. In contrast to presently available general-purpose chemical formats, UCM offers built-in validation, which combines both grammar and pattern-based XML schema languages. Thus, all recorded data can be precisely validated by UCM schemas in standard XML validators.

Conclusions
We developed the structure for UCM from scratch on the basis of an analysis described in our previous article. Starting from scratch allowed us to integrate BibTeXML, UnitsML and XHTML as well as chemical line notations and identifiers into UCM. It also helped us to avoid unnecessary redundant parts and create the implementation that aims to minimize ambiguity and is designed to be easily extensible in the future.

Keywords: Universal Chemical Markup, UCM, UCM XML structure, UCM built-in validation, UCM examples, UCM VIEWER, recording chemical structures with properties, combining XML formats, combining XML schema languages

BACKGROUND

Although many chemical formats currently exist, most are tailored to specific areas of chemistry. Relatively few formats provide the general-purpose functionality, which is not limited to specific software or a specific area of chemistry, and supports the recording of common chemical data (i.e. structures, reactions and properties). This is apparent from our previous analysis of the widely used formats for common chemical data.[1] In that analysis the strengths and weaknesses of these formats were examined. Acquired knowledge were further utilized when we designed the XML structure for UCM using the most suitable concepts from explored formats.

We decided to develop UCM, because our earlier analysis suggests no widely used general-purpose chemical format offers future extensibility together with the effective processing and precise built-in validation of chemical data.[1] The analysis results also confirmed XML technology has potentially significant benefits for chemical formats.[1] Thus, the development of UCM started from scratch with the aim of creating a new general-purpose XML format for common chemical data. This as well as the strengths and weaknesses of analyzed formats is described more thoroughly in our previous article.[1]

Before we explain the advantages of UCM, it is first necessary to describe at least some issues we have found in the CTfile (Chemical Table File), CDXML (ChemDraw Exchange Markup Language) and CML (Chemical Markup Language) formats. These formats, despite their weaknesses, represent the existing XML and non-XML solutions with rich general-purpose functionality for common chemical data, as can be seen from the benefits presented in our analysis.[1]

The popular CTfile formats can together record data about chemical structures, reactions, and properties,[2,3,4] but the built-in validation is missing in all original non-XML CTfile formats (i.e. Molfile, RGfile, Rxnfile, SDfile and RDfile).[1] In the case of XDfile (XML Data File), which is the only XML-based CTfile format, its schema can be used by standard XML validators to check the format's basic structure.[1] But because the XDfile schema provides just simple grammar-based validation, and most chemical data in XDfile are stored in the embedded non-XML formats (e.g. Molfile, Rxnfile, etc.), the precise validation of chemical data is not achieved.[1] Furthermore, our analysis suggests that implementing the precise validation for CTfile formats (including XDfile) may require additional effort, as the original CTfile syntax is not based on XML or similar standard, which offers the validation infrastructure automatically.[1] Also in the non-XML CTfile formats the stored data items are less readable, as they often lack descriptions and there are only limited annotation possibilities.[1]

CDXML closely follows binary CDX (ChemDraw Exchange) specifications[5] and is this causes at least two potential problems. The first, similarly as in CDX, is the mixing of chemical information with embedded binary objects and data describing content visualization. It makes the XML structure of CDXML more difficult to understand and implement.[1] The second is that the dependency on changes in CDX specifications negatively affects the future extensibility of CDXML.[1] Another issue is in the validation functionality. Because CDXML uses DTD (Document Type Definition), a schema language with limited expressive power,[6] its basic XML structure can be validated easily, but chemical information is not validated precisely.[1] In addition DTD does not support XML namespaces.[6] Thus, CDXML will need to be redefined with a newer schema language to add a unique namespace, so that the format can be combined directly with other XML formats.[1]

CML has undergone a long evolution[7] and provides rich and flexible functionality focused on extensibility,[8,9] but also has its issues. Starting with version 2 it was redefined using XSD (W3C XML Schema Definition Language).[10,11] The flexibility of the format was increased by defining minimal restrictions and lax validation for many parts (see the CML schema 2.4 or 3[12]). In CML schema 3 any mandatory tree structures were abandoned to offer even more flexibility.[13] However, the fact that more than one way of recording the particular chemical information usually exists makes the format confusing.[1] The validation of such a format then becomes complicated, because the schema uses sometimes deliberately fuzzy concepts and enables all CML elements to contain each other.[1] To address these problems the authors of CML implemented a mechanism for denoting standard conventions for various CML elements.[8,10,14] The main issue was that until version 3 no official list of conventions was composed and CML users, including developers of chemical software, had to use their own conventions.[1] While with version 3 CML users can still create and use custom conventions, the documentation now lists some official conventions aimed at adding additional rules and bringing more order to the format.[1] Part of this effort is also the CMLLite validator service intended for flexible validation based on conventions.[15] But the listed conventions appear to be mostly unfinished. All are marked as draft recommendations and some are missing (e.g. the convention dealing with CML support for recording crystallographic information or reactions etc.).[1] Until the necessary official conventions are developed and implemented at least in the CMLLite validator service, CML will remain a very variable format that is difficult to process reliably.[1]

With the exception of CML, the general-purpose formats we explored also cannot directly record electrons participating in chemical bonds and do not properly enforce associating scientific units with the values of properties.[1] This leads to a less precise description of properties and structures, especially the structures with more complicated delocalized bonds.

UCM, the new XML format we propose, aims to avoid the issues identified in previously analyzed XML and non-XML formats.[1] Among the main benefits of UCM is the precise validation, which uses software from the standard XML tool chain and combines both grammar and pattern-based XML schema languages. The grammar-based UCM schema validates the format's tree structure, while the pattern-based schema validates constraints between UCM attributes and elements. These constraints can even check UCM data for chemical mistakes like a carbon atom participating in five different single bonds, or a formal charge of a structure not corresponding with the number of protons and electrons in it. The first version of UCM focuses on recording chemical structures and their properties. Because UCM has a unique namespace it can be directly combined with other XML formats to include complex scientific data from various domains in a single file.

IMPLEMENTATION AND OVERALL DESIGN OF UCM

In the iterative development of the XML structure for UCM we used and improved the most suitable concepts found in chemical XML formats from our earlier analysis.[1] This illustrates our aim of reusing the current knowledge and available solutions where possible, and improving upon these when necessary. Thus, the overall design philosophy of UCM is not just about creating a new format without the previously described issues. It is also about integrating the existing knowledge and solutions into UCM to avoid reinventing the wheel. Consequently, we designed UCM in a way that ensures various XML and non-XML formats can be integrated in it and provide additional specialized functionality.

The resulting XML structure for UCM 1-1-1 is in tree structure schemes 1 and 2, while our previous article describes how UCM implements various concepts compared to other XML formats we analyzed.[1] Tree structure schemes 1 and 2 follow the complete format structure defined by UCM validation schemas, which are available in additional file 1. We used very simple syntax to make UCM tree structure easily readable:

- `element` – Denotes an UCM element.
- `@attribute` – Denotes an UCM attribute.
- `[CONTEXT]` – Specifies the context of an UCM element (e.g. `node [C1]` means the *node* element that uses its first context).
- `[CONTEXT: XPATH]` – Defines the context of an UCM element (e.g. `define [C1: @format = 'UCM']` means the first context for the *define* element is the *define* element with the value "UCM" in its *format* attribute). The XPath (XML Path Language) expression only further describes the context, which is defined by the attributes and child elements enabled for the given UCM element.
- `(ATTRIBUTES)` – Specifies the enabled attributes of an UCM element (e.g. `point (@id, @x, @y, @z)` means the *point* element with the *id*, *x*, *y* and *z* attributes).
- Quantifiers "?", "*" and "+" are used to express 0 or 1, 0 or more, and 1 or more respectively.
- Keywords "OR", "IF" and "IF NOT" have their literal meaning.
- Element contents are indented by four spaces.
- Ellipsis means the attributes and contents of the element are in its definition.

As can be seen from tree structure schemes 1 and 2, some UCM elements may be utilized in more than one context. Contexts are defined in UCM Schematron schema by additional restrictions placed on the particular UCM element when it is used at the specific position in UCM tree structure, or when it has some specific attribute or a certain value of some attribute. Take for example the UCM *node* element. If it occurs inside a *define* element the following restrictions from UCM Schematron schema apply to

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1336v1 | CC-BY 4.0 Open Access | rec: 30 Aug 2015, publ: 30 Aug 2015

the *node* element: the mandatory *id* attribute is enabled, but other attributes must be omitted; the *stereo* child element must be omitted, but there must be at least one *particle* child element (for *description* and *property* child elements occurrence remains as defined in the main UCM schema). The restrictions define the first context for UCM *node* element. Source code snippet 3 demonstrates how that context is used for storing reusable UCM node definitions. This way UCM has relatively few elements with flexible and easy to learn functionality, but at the same time precise validation is ensured by exactly specifying all useful contexts for these elements.

Integrated XML formats

Using XML namespaces we integrated BibTeXML, UnitsML and XHTML into UCM. Unique XML namespaces are essential for combining two or more XML formats together. The reason is that unique XML namespaces make it possible to distinguish the attributes and elements from different formats even if these attributes or elements have the same local names.[16,17] Moreover, XML namespaces are also used by NVDL (Namespace-based Validation Dispatching Language),[18,19] which we use for the Implementation of data validation in UCM.

The purpose of XHTML integration is simply to enhance the annotation capabilities of UCM *description* elements. XHTML enables hyperlinks and other useful formatting features in annotations, as is demonstrated in our Results with examples of UCM usage. In addition it ensures that all markup inside the *description* elements is compatible with web browsers without any additional transformations.

With BibTeXML integrated in UCM *define* elements one can define literature references for the *description* elements. Both the default and extended version of BibTeXML is supported in UCM. Default BibTeXML 1.0 offers the functionality of the original BibTeX format, which is widely adopted by the scientific community and is used with LaTeX document preparation system.[20] BibTeXML 1.0 Extended adds functionality for recording additional data items more precisely.[20] This is useful for modern literature references with Digital Object Identifiers and other data items, as shown in source code snippet 1. Because software tools for automatic conversion of BibTeX bibliographies into BibTeXML are freely available at the format website,[20] users can easily add their literature references to UCM annotations from most reference managers.

Similarly, UnitsML is also integrated in UCM *define* elements, as it enables defining various quantities with their scientific units. Source code snippet 2 gives an example of how it works. At first necessary scientific units are defined using UnitsML *Unit* elements, which are encapsulated in the *UnitSet* element. Then, quantities are defined inside the *QuantitySet* element by UnitsML *Quantity* elements. Each *Quantity* element usually refers to the appropriate *Unit* element that defines the scientific unit (see the UnitsML *UnitReference* element in the *Quantity* element). Because UnitsML offers a set of root units, it is possible to easily define most scientific units in a precise way.[21] Further information about UnitsML are available at the format website.[21] We especially recommend the format documentation, as it can be easily navigated thanks to cross-references and clear formatting.

Integrated non-XML formats

UCM also integrates non-XML formats for chemical line notations and identifiers. As apparent form tree structure scheme 1, these formats are integrated in the UCM *structure* elements. The functionality of integrated chemical line notations and identifiers nicely complements the capabilities of UCM markup. The *structure* elements that use UCM format can be validated in detail and are intended for precisely recorded structures with coordinates, properties or annotations. On the other hand sometimes one may need to store just a concise chemical identifier or query, which is usually without coordinates and more or less ambiguous. In such cases the *structure* elements can use one of the integrated formats to record the chemical structure or query using:

- Preferred IUPAC Name, General IUPAC Name, Chemical Abstracts Index Name,
- CAS RN (Chemical Abstracts Service Registry Number), Reaxys Registry Number, ChemSpider ID Number, PubChem Compound ID Number, PubChem Substance ID Number,
- InChI (International Chemical Identifier), InChIKey, Standard InChI, Standard InChIKey,
- SMILES (Simplified Molecular Input Line Entry System), SMILES Arbitrary Target Specification or SYBYL Line Notation.

Examples showing how these integrated formats may be used are presented in our Results with examples of UCM usage. Further information are also available in UCM documentation provided in additional file 2.

IMPLEMENTATION OF DATA VALIDATION IN UCM

During the development of UCM we have set the precise built-in validation as one of the main design goals, because current general-purpose chemical formats often have only limited built-in validation capabilities.[1] As described in our previous article, XML technology offers significant benefits one of which is the available validation infrastructure.[1] Virtually any formally defined XML format has a schema that provides at least some basic validation capabilities. In contrast to most XML formats, we decided to further enhance the validation capabilities of the main UCM schema. Thus, the validation in UCM employs a combination of XML schema languages.

The main schema of UCM uses RELAX NG (Regular Language for XML Next Generation) compact syntax. We also offer its versions translated with Trang (a converter for common XML schema languages[22]) into RELAX NG XML syntax and XSD. This ensures very high compatibility especially when the main schema is used independently with various XML validators. The main schema provides relatively precise grammar-based validation of UCM tree structure. The values of all UCM attributes and elements are checked for correctness. In other words each value must conform to its data type and a possible set of additional restrictions. An example is the *counts* attribute value, which must be one or more whitespace separated non-negative integer numbers. For UCM elements the main schema also checks whether they have the correct attributes. Additionally, for UCM elements that should contain other elements the correct sequence and occurrence of child elements is checked. Such validation could be considered sufficient, as many current XML formats, both chemical (e.g. CDXML, XDfile, etc.) and non-chemical (e.g. BibTeXML, UnitsML, XHTML, etc.), are defined and validated by a single grammar-based schema.[1,20,21,23] However, validation using a single schema written in one of the widely popular grammar-based XML schema languages (e.g. RELAX NG, XSD, etc.) has at least two limitations. The first limitation becomes apparent when one considers validation of XML files combining two or more XML formats with unique namespaces together. The grammar-based schema for one XML format cannot check the content from other XML formats unless it somehow includes their schemas too (such combinations are for example described in Mathematical Markup Language specification[24]). The second limitation is that the grammar-based schema cannot express validation constraints between two or more attributes, elements or values (for these constraints Schematron and its pattern-based approach is usually utilized[25]).

To validate the content from UCM namespace as well as the content from BibTeXML, UnitsML and XHTML namespaces we added a NVDL (Namespace-based Validation Dispatching Language) schema to UCM. NVDL makes it possible to load the appropriate schema for the content from each namespace.[18,19] This addresses the first limitation we mentioned previously for the validation that relies on a single grammar-based schema. In addition, NVDL enables the content from one namespace to be easily validated by more than one schema.[19,26] We plan to utilize NVDL for validating the content from UCM namespace by both RELAX NG and Schematron UCM schemas as soon as there is mature support for ISO Schematron in NVDL-capable validators. NVDL-capable validators are widely available[19] and except for the limited ISO Schematron support we observed no issues with common XML schema languages (e.g. RELAX NG, XSD or DTD).

With the Schematron schema we eventually achieved a very precise and extensible validation of more complex constraints between two or more UCM attributes, elements or values. The usage of Schematron obviously solved the second limitation of the validation based on just a single grammar-based schema. While the direct implementations of Schematron exist, the official Schematron website also offers the reference ”Skeleton” implementation based on XSLT (Extensible Stylesheet Language Transformations).[27,28] We found the ”Skeleton” implementation to be most mature especially since the UCM Schematron schema uses the current ISO version of Schematron. This way UCM Schematron validation may be used separately (i.e. we chose not to integrate it in RELAX NG schema though it is possible[29]) and is widely compatible as the only required software tool is the XSLT 2.0 processor.

Among other things the Schematron schema for UCM can validate various chemical rules, which to the best of our knowledge cannot be checked by built-in validation in current general-purpose chemical formats.[1] Two main examples from UCM 1-1-1 Schematron schema include: the validation of formal charges on chemical structures and chemical nodes (i.e. nodes from a chemical graph of the structure), and the validation of bonding electrons (i.e. electrons to be used in UCM *bond* elements). To enable the easier implementation of these validation tasks we designed UCM around the principle of precisely defining all chemical nodes that represent the monoatomic particles in chemical structures. Thus, each monoatomic particle (e.g. an atom or a monoatomic ion) in UCM is defined as composed of protons, neutrons and electrons. In other words each UCM node is defined by a *node* element containing *particle* child elements, which specify the number of protons, neutrons and electrons. For an example see the definitions of hydrogen cation, deuterium and carbon nodes in source code snippet 3. Note how isotopic composition can be recorded with *fractions* attributes and how the number of bonding electrons can be distinguished by the "BE" value of the *type* attribute on UCM *particle* elements.

Using the information about the number of protons and electrons for each UCM *node* element our Schematron schema checks if the *charge* attributes on *node* and *structure* elements have the correct value. Of course in practice electrons from some *node* element may be shared with other *node* elements in a way that leads to decimal values of the *charge* attribute. An example is the structure of dihydrogen cation in source code snippet 6 or the structure of ozone in source code snippet 10. The validation then becomes more complicated, but it may be still implemented in Schematron, as one can use additional XSLT functions in Schematron validation patterns. Source code snippet 4 demonstrates this with our XSLT function and validation pattern for the *charge* attribute on an UCM *node* element. The validation pattern calculates the *charge* attribute value simply by subtracting the number of electrons (i.e. the variable "NEGATIVE-CHARGE") from the number of protons (i.e. the variable "POSITIVE-CHARGE") for the given *node* element. In order to take into account the possible shared electrons the XSLT function "lf:COUNT-NODE-NEGATIVE-CHARGE" is used. This function subtracts the number of shared electrons from the total electron count obtained from the *node* element definition. Then only the correct fractions of shared electrons are added back to get the actual negative charge for the particular *node* element. The correct fractions of shared electrons are obtained from the values of the *fractions* attributes on relevant UCM *share* elements.

Similarly the Schematron schema also verifies whether each UCM *node* element has enough bonding electrons for all its chemical bonds. Source code snippet 5 contains the core parts of that validation procedure. At first UCM node definitions are searched for the number of electrons the *node* element should provide to *bond* elements. Then the XSLT function "lf:COUNT-NODE-BONDING-ELECTRONS" calculates the number of electrons required by all UCM *bond* elements that refer to the particular *node* element. If both numbers are not equal the error occurs and the particular chemical node either participates in too many bonds (i.e. it has too few bonding electrons) or does not participate in all its bonds (i.e. it has too many bonding electrons).

In practice UCM validation should help users to determine if the given structure seems to be chemically correct or incorrect. Of course UCM 1-1-1 schemas cannot detect all possible chemical errors, but in future versions validation can be further enhanced and extra rules may be added. When compared to current general-purpose formats such as CDX, CDXML, CML and CTfile formats (including XDfile) we believe UCM validation brings significant improvements.

RESULTS WITH EXAMPLES OF UCM USAGE

To test UCM during the development we prepared examples, which use the key format functionality in practice. Complete and quite easily readable UCM files with all examples are in additional file 1. For readers of the following sections we further explain the selected examples to better illustrate how UCM can record various chemical structures and properties. In each example the chemical graph of the given structure was manually rewritten to UCM in a plain text editor. The chemical graph was constructed using information about the structure in scientific literature and chemical databases. Most examples also include 3-dimensional coordinates from available online sources and chemical databases. For some examples 3-dimensional coordinates were obtained from a skeleton structure we created with Avogadro (an open source chemical editor[30,31]) and optimized in Nwchem (an open source computational chemistry software[32,33]). We mention the original source of 3-dimensional coordinates and other information about the structure in each example, mainly to demonstrate the annotation functionality of UCM.

A simple UCM VIEWER, which we developed using JavaScript, XHTML and CSS (Cascading Style Sheets), helps to visualize the UCM examples in the following sections. In its first version UCM VIEWER supports the key functionality of UCM 1-1-1. To run UCM VIEWER one only needs a modern web browser with a built-in XSLT processor and support for JavaScript, or other ECMAScript implementation. UCM VIEWER utilizes JavaScript for parsing and rendering the content from UCM files. We use two open source JavaScript libraries in UCM VIEWER, three.js and jQuery. With its easy-to-use API (Application Programming Interface) compatible across a multitude of web browsers, jQuery is a well known library that simplifies various tasks in JavaScript (e.g. document traversal and manipulation, event handling, etc.).[34] However, equally important for UCM VIEWER is also the three.js project, which aims to create a lightweight library for 3-dimensional rendering with a very low level of complexity.[35] We have found the API of the three.js library intuitive and highly usable, while both quality and performance were sufficient for our use case even with the basic canvas renderer (the library provides WebGL and other renderers too[35]). Overall we think the source code of UCM VIEWER, available from http://www.universalchemicalmarkup.org under an open source license, demonstrates how the software support for UCM can be implemented relatively easily.

Examples of simple structures and structure fragments

In the first example we included small structures and structure fragments without 3-dimensional coordinates just to introduce the basic overall structure of an UCM file with chemical data. The complete example contains: dihydrogen cation, heavy water, ozone resonance hybrid with its resonance structures, and fragments from sodium chloride and hydrogen fluoride structures. The *ucm* root element in source code snippet 6 is used as a container for other UCM elements and it also specifies the namespaces and UCM version. Mandatory *version* attribute is intended to prevent future problems with versioning. Source code snippet 6 shows only the first two UCM *structure* elements with data about dihydrogen cation and heavy water. The *node* elements store chemical nodes, which represent the monoatomic particles (e.g. atoms or monoatomic ions) in the structures, while *bond* elements record bonds. Each *structure*, *node* or *bond* element has a mandatory *id* attribute to enable reliable cross-referencing using *idrefs* attributes. One can see how UCM enables recording and utilizing information about subatomic particles such as protons, neutrons and electrons when chemical bonds and nodes are described. The *idrefs* attribute on a *node* element provides cross-reference to the reusable node definition that specifies the number of protons, neutrons and electrons (see source code snippet 3). On a *bond* element the *idrefs* attribute provides cross-reference to *node* elements participating in the bond. In addition it is possible to describe electrons and how they are shared inside a bond using UCM *particle* and *share* elements. This is demonstrated inside the first *bond* element in source code snippet 6 and we explain it further when describing Examples of more complex structures with delocalized bonds.

Before we expound the remaining examples, it is necessary to mention how XInclude (XML Inclusions) mechanism may be utilized in UCM and how this affects the basic structure of UCM files. XInclude is a mechanism for general purpose inclusion accomplished by merging a number of XML information sets into a single composite infoset.[36] As can be seen in source code snippet 6, after XInclude namespace prefix is defined (on the *ucm* root element) it is possible to use the *include* elements from XInclude namespace to incorporate the content from external XML files. XInclude processing occurs at a low level, often by a generic XInclude processor which makes the resulting information set available to higher level applications.[36] For UCM *define* elements such mechanism is especially advantageous, because one may reuse precise UCM, BibTeXML and UnitsML definitions simply by including them in UCM files that require them. Thus, usually the main UCM file with chemical data will reference other UCM files with definitions. We assume some definitions (e.g. UCM node definitions or UnitsML quantity and units definitions) could be offered as a part of future UCM versions, and so our UCM examples already use separate files for each group of similar definitions.

Now let us continue with the second example. Although optional in UCM, the coordinates of the *node* elements positions in 3-dimensional space are essential for an accurate description of the given structure and its precise visualization. Figure 1 and source code snippet 7 with the second example demonstrate this basic UCM functionality on a simple structure of urea. All positions of the *node* elements, which represent atoms in the urea structure, are stored using attributes *x*, *y*, *z* denoting coordinates in 3-dimensional space expressed in nanometers.

The second example also contains chemical identifiers for the urea structure and an annotation with XHTML markup inside the UCM *description* element. As depicted in figure 1, XHTML enables the user to easily include links to additional online resources and to enhance the formatting of annotation text. The main part of figure 1 shows how UCM VIEWER displays the urea structure and its chemical identifiers stored in UCM *structure* elements that have the "STID" value in their *type* attributes (see source code snippet 7).

Example of structure with properties and literature references

To record literature references and measured or calculated properties UCM *define* elements support BibTeXML and UnitsML, as we already explained (see source code snippets 1 and 2). When values of properties are recorded using UCM *property* elements, as in source code snippet 8, a *quantity* attribute on the given *property* element references UnitsML quantity via its *xml:id* attribute. Similarly the *litrefs* attribute on the UCM *description* element references BibTeXML literature reference via its *id* attribute. The figure 2 depicts UCM VIEWER displaying properties loaded from *property* elements together with annotations from *description* elements, which also include literature references. In source code snippet 8 one can see different uses of the UCM *property* element. The *property* element with the "CN" value in its *type* attribute can describe a condition for the parent *property* element. If the *type* attribute has the value "ER", the *property* element describes errors in the *values* of the parent *property* element. This enables precise recording of errors in measured or calculated values using well defined statistical quantities like standard deviation, as can be seen from source code snippet 8. Although our simple example does not include it, *property* elements may be also utilized inside UCM *bond*, *node*, *particle* and *point* elements. Thus, it is possible to describe various properties related to these elements (e.g. bond length, bond dissociation energy, atomic weight, atomic radius, particle spin, etc.) as easily as in the case of UCM *structure* elements.

Examples of more complex structures with delocalized bonds

Previous examples included structures where chemical bonds could be described quite simply. However, there are many structures with more complex bonding, which current general-purpose chemical formats fail to describe in detail. It is because formats such as CDX, CDXML, CML and CTfile formats (including XDfile) do not support bond types that enforce specifying and validating electrons for more complicated bonds (e.g. aromatic and other delocalized or otherwise special bonds).[1,37,12,3] In UCM, the *bond*, *join* and *particle* elements can be used together to record both simple and complex bonding. The *particle* elements enable the usage of subatomic particles in UCM. Thus, it is possible to record and validate electrons participating in the given bond.

Source code snippet 9 and figure 3 present the example of diborane with two 3-center-2-electron bonds between the bridging hydrogen and boron atoms. These bonds are stored by two UCM *bond* elements. Each contains the *join* element with information about how to join the *node* elements that participate in the bond. The "CT" value inside both *join* elements stands for the centered interpretation of their *idrefs* attribute values. It means that the first id reference refers to the *node* element which is bonded, by the bond expressed in the parent *bond* element, to all *node* elements specified by the remaining id references. Therefore, in the case of diborane, the first id reference (in the *idrefs* attribute of the *join* element) references the *node* element that represents the bridging hydrogen atom, while the remaining id references refer to *node* elements representing the boron atoms. The id reference inside each *idrefs* attribute on the *particle* elements points to the *node* element which provided the electron. This enables the precise validation of bonding electrons, as we already described (see source code snippet 5).

Some advanced usage of UCM *bond* element can be seen also in the *structure* elements that describe ozone. Source code snippet 10 shows the relevant part of the first example here. The *structure* element with the "E1-S-3" value in its *id* attribute stores the resonance hybrid of classical ozone Lewis structures, which are presented in the next two *structure* elements. The most interesting is the first *bond* element (with the "E1-B-3-1" *id* attribute value) that records the 3-center-2-electron bond formed by two electrons from the pi bonding orbital covering all three oxygen atoms. Notice how *share* elements precisely describe sharing of electrons between the *node* elements and how it leads to non-zero formal charge values on these *node* elements. Sharing ratios for the *node* elements referenced in the *idrefs* attributes on each *share* element are stored in the *fractions* attributes. Of course the ozone structure is still a topic of new research. For example one of the recent papers suggests the classical Lewis structures contribute 82% to the resonance hybrid of ozone, while remaining 18% is contributed by the biradical resonance structure[38] (see the last *structure* element in source code snippet 10). In future UCM versions we plan to add support for mixed *structure* elements capable of recording such resonance hybrids as well as other chemical substances and mixtures.

Another example is the structure of ferrocene in figure 4. Interesting bonds in this example include the aromatic bonds in cyclopentadienyl rings and bonds between the central iron node and cyclopentadienyl rings. Details of these bonds are in source code snippet 11. The *bond* element with the "E5-B-1" value in its *id* attribute represents the aromatic bonding in the first cyclopentadienyl ring. The "CC" value of the *join* element stands for the cyclic interpretation of its *idrefs* attribute value. This means that all aromatic bonds in the first cyclopentadienyl ring are recorded in UCM using single *bond* element, without sacrificing any details. It can be still seen that fifteen electrons from carbon atoms and one extra electron are participating in the aromatic bonding. We could even distinguish the sigma and pi electrons from carbon atoms by adding one more *particle* element.

Because the UCM *structure* element can be nested, it is possible to encapsulate any useful substructure and then reference it later. This is demonstrated in source code snippet 11 by the *structure* element with the "E5-S-1-3C" value in its *id* attribute. All carbon atoms of the first cyclopentadienyl ring are encapsulated in that element. Thus, just one *particle* element may be used to reference these carbon atoms in the *bond* element representing the aromatic bonding.

Ferrocene example also shows a straightforward usage of *charge* attribute and sharing of electrons among bonds. The *charge* attribute value must correspond to the number of proton and electron particles used inside the given *node* or *structure* element. The sharing of electrons between bonds can be seen in the *bond* element with "E5-B-13" value in its *id* attribute. That *bond* element denotes the bond between the central iron *node* and the first cyclopentadienyl ring. As can be seen in source code snippet 11, three electrons in this *bond* are from the central iron *node*, but the remaining six are pi electrons shared with the *bond* element representing the aromatic bonding. Observe how the *bond* element with the *id* attribute value of "E5-B-13" connects the central iron *node* and the centroid of the first cyclopentadienyl ring denoted by the *point* element. When the example is rendered by UCM VIEWER it emphasizes the fact that both cyclopentadienyl rings rotate.[39]

Finally there is an example of trichloro(ethene)platinate(II) anion (commonly known as Zeise's salt anion). Here the interesting bond is between the central platinum atom and ethylene ligand. Figure 5 depicts both our approaches at describing this bond, while source code snippet 12 shows the relevant UCM markup. In the first approach we use the 3-center-2-electron bond to connect the central platinum atom and ethylene ligand (see the UCM *structure* element with the "E6-S-1" value in its *id* attribute). As a result only a single bond remains between the ligand carbon atoms. On the other hand our second approach emphasizes that the order of the bond between ethylene ligand carbon atoms is not completely reduced to a single bond.[40,41] Therefore, the central platinum atom and ethylene ligand in the structure "E6-S-2" are connected using a delocalized bond, which shares an electron with the partial double bond between the ligand carbon atoms.

Examples of structures with stereochemical configuration

The following examples illustrate how to describe the stereochemistry in UCM using the *stereo* element. An example in figure 6 depicts serine amino acid with chirality centre on the highlighted carbon atom. The *stereo* element in the source code snippet 13 describes the chirality centre on the *node* element with the *id* attribute value "E7-N-5". Substituents are represented by *node* elements with the *id* attribute values "E7-N-4", "E7-N-6", "E7-N-7" and "E7-N-8". References in the *idrefs* attribute on the *stereo* element are ordered by descending priority of those substituents. Because the sense of rotation from the highest to lower priority substituents is counterclockwise, when the lowest priority substituent is pointed away from the observer (as in figure 6), the *sense* attribute has the "-" value. Substituents priority was assigned according to the Cahn-Ingold-Prelog system of priority rules, so the "-" value of the *sense* attribute corresponds to the "S" configuration of the chirality centre. However, for easier implementation of UCM in software tools other algorithms can be used to assign the substituents priority.

Another basic example in figure 7 deals with the stereochemistry of a double bond. The UCM *stereo* element in source code snippet 14 stores the stereochemical configuration of the double bond represented by the *bond* element with the *id* attribute value "E8-B-1". Substituents are represented by *node* elements with the *id* attribute values "E8-N-3", "E8-N-4", "E8-N-5" and "E8-N-6". References in the *idrefs* attribute on the *stereo* element are ordered by descending priority of those substituents (assigned using

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1336v1 | CC-BY 4.0 Open Access | rec: 30 Aug 2015, publ: 30 Aug 2015

Cahn-Ingold-Prelog system of priority rules). For a double bond, the reference plane contains nodes participating in the bond and is perpendicular to the plane containing these nodes and the nodes directly bonded to them. Because both higher priority substituents are on the same side of this reference plane, the *sense* attribute has the value "+", which corresponds to the "Z" configuration of the double bond.

The last two examples in figure 8 demonstrate that the UCM *stereo* element can be used to describe even more complicated stereochemistry. Both examples contain metal tris chelate structures, where the central metal is attached to three bidentate ligands. The source code of these examples is available in additional file 1. In that source code we included *description* elements with annotations, which explain how the *stereo* element stores the absolute configuration of the given structure or the twist ligand conformation.

Further details about recording the stereochemistry in UCM are provided by the documentation in additional file 2. There the documentation of the *stereo* element and its *sense* attribute also explains how to describe the stereochemistry of the square planar or octahedral complex and the chiral axis.

DISCUSSION COMPARING UCM WITH OTHER CHEMICAL FORMATS

Compared to other general-purpose chemical formats UCM offers an alternative approach, as it also focuses on precise data validation. Current general-purpose formats such as CDX, CDXML, CML and CT-file formats (including XDfile) use chemical elements predefined as a set of enabled text symbols.[37,13,3,4] UCM on the other hand relies on reusable definitions of chemical nodes composed from protons, neutrons and electrons. This enables the detailed description of chemical bonds and the precise validation of chemical content by checking various chemical rules with the pattern-based UCM Schematron schema. Additional grammar-based schemas validate UCM tree structure including the integrated formats, which can be utilized in UCM. Our built-in validation combines grammar and pattern-based XML schema languages and similarly to the other aspects of UCM functionality it is designed with future extensibility in mind. Available general-purpose chemical formats we analyzed do not offer comparable built-in validation capabilities.[1] Our previous analysis suggests non-XML formats often lack any form of built-in validation, while XML formats usually provide some basic validation of the format structure. Of course some non-XML formats for chemistry include validation services by providing specialized software tools – an example is Crystallographic Information File[42,43] although it is not a general-purpose chemical format.[1] Among chemical XML formats, CML and its CMLLite validator is an exception, which aims to provide additional validation capabilities beyond what is possible to achieve with a basic grammar-based schema for the given XML format.[15] However, in its current state custom XSLT and Java-based CMLLite validation has significant issues described in our previous article.[1] Therefore, UCM validation uses standard XML schema languages instead of some custom-built mechanisms and because of that it is much more compatible with the available XML validation infrastructure.

Besides precise built-in validation, the clear and highly readable UCM structure provides good extensibility and flexible functionality. This was achieved by utilizing the most suitable concepts found during our earlier analysis and by implementing these concepts using relatively few UCM attributes and elements.[1] Because of that some UCM elements can be used in more than one context. Contexts, defined by UCM Schematron schema, further restrict (and also validate) the attributes and child elements enabled for the given UCM element when it is used in the specific way or at the specific position in UCM tree structure. Other XML-based formats like CDXML or CML tend to have more attributes and elements for the given functionality and these attributes and elements are sometimes even redundant, as we discovered through our analysis.[1] In addition UCM is designed to enable the integration of various XML and non-XML formats, which provide some useful specialized functionality. Our approach is quite different compared to CML and its mechanism of conventions. Conventions are available in CML from version 1 and are expressed by the *convention* attribute with originally unrestricted value.[8,10,14,44] In the current versions of CML this attribute may still have virtually any value conforming to a pattern specifying a string similar to the XML qualified name, but with required prefix (see the CML schema 2.4 or 3[12]). It effectively means that for example the "my_convention:CustomBonds-1" value denoting a convention described by someone on some website specifies a legitimate CML convention according to documentation annotations in schema version 3 or 2.4.[12] Such unrestricted approach adds flexibility for new use cases, but together with the content model removal in CML schema version 3 it also leads to a highly variable and difficult to process format.[1] Currently it remains to be seen whether the CMLLite validation and official CML conventions, which appear to be unfinished and are marked as draft recommendations, will solve at least some CML issues we described in our preceding article.[1] UCM approach, on the other hand, is to integrate well defined XML and non-XML formats with useful functionality into predefined UCM elements. By carefully choosing formats that complement the capabilities of selected UCM elements, we believe UCM can still be sufficiently flexible for most practical use cases without disrupting its validation and other benefits.

While UCM avoids many issues we identified in widely used general-purpose chemical formats,[1] it also has its limitations as a newly developed format. These limitations could be divided into two categories. The first concerns the format functionality. UCM 1-1-1 provides promising functionality for recording chemical structures and properties. In order to become a real general-purpose chemical format UCM will at least need the support for recording chemical reactions. We are preparing a new UCM version that is going to have this functionality together with the support for mixed structures to describe chemical substances and mixtures. With additional future UCM versions, we plan to add the support for recording polymers and crystallographic information. The second category of limitations is about available software. UCM can already benefit from the standard XML tool chain, which includes the infrastructure in programming languages and software for XML processing (especially for parsing, navigation, transformation and validation of XML documents), and can be used to process and implement UCM more easily. However, it is obviously too early to expect support for UCM in specialized chemical software. To help addressing this we are now working on utilizing Open Babel (an open-source chemical toolbox for conversion between chemical formats[45]) to develop a software tool for automatically converting data from current chemical formats into UCM. We also plan to extend our UCM VIEWER, so that it becomes even better reference example of how to read and display all data from various UCM files.

CONCLUSIONS

We have developed UCM as an alternative to the current formats for common chemical data. To avoid potential issues we identified in XML and non-XML formats for chemistry, the design of UCM reflects the knowledge from our previous analysis.[1] Among the main benefits of UCM is the precise grammar and pattern-based validation compatible with the standard XML tool chain; in other words one can use UCM schemas to validate not only the format's tree structure, but also various chemical constraints (e.g. whether the formal charge of a structure corresponds to the total number of protons and electrons in it). Unlike most currently available chemical formats UCM supports additional non-chemical functionality by integrating dedicated XML formats using the mechanism of XML namespaces. This way UCM is not cluttered with non-chemical functionality, while at the same time, such functionality is provided in a complete and unrestricted form by a dedicated XML format: BibTeXML for literature references, UnitsML for quantities and scientific units, and XHTML for web-compatible annotations with hyperlinks and other formatting. The first version of UCM focuses on recording chemical structures with their properties. It supports structures with isotopes, ions and various types of bonds, in which participating electrons can be recorded and validated. Besides that, chemical line notations (e.g. SLN, SMILES, etc.) and identifiers (e.g. CAS RN, InChI, InChIKey, etc.) are also supported.

Overall we believe UCM provides very promising functionality for the core of a modern open source chemical format. It is easily extensible and can be implemented even in web-based applications as we demonstrated with UCM VIEWER, which uses JavaScript for wide compatibility across modern web browsers. In future UCM versions we plan to add support for recording chemical reactions, mixed structures (to describe substances and mixtures), polymers and crystallographic information.

AVAILABILITY AND REQUIREMENTS

Project name: UCM 1-1-1
Project home page: http://www.universalchemicalmarkup.org/#UCM--1-1-1
Operating system(s): platform independent
Programming language: XML, RELAX NG, Schematron, XSLT, NVDL
Other requirements: standard XML validators (e.g. xmllint and jing), standard XSLT 2.0 processor (e.g. saxon)
License: GNU GPL 3
Any restrictions to use by non-academics: None

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1336v1 | CC-BY 4.0 Open Access | rec: 30 Aug 2015, publ: 30 Aug 2015

Project name: UCM VIEWER 1-1-1
Project home page: http://www.universalchemicalmarkup.org/#UCMV--1-1-1
Operating system(s): platform independent
Programming language: JavaScript, XSLT, XHTML, CSS
Other requirements: modern web browser with a JavaScript/ECMAScript support and built-in XSLT processor (e.g. Mozilla Firefox)
License: GNU GPL 3
Any restrictions to use by non-academics: None

SUPPLEMENTAL INFORMATION

Additional file 1 – UCM examples and schemas

All UCM examples and schemas with listed validation and processing commands.

Additional file 2 – UCM documentation

Complete documentation for all UCM attributes and elements.

ACKNOWLEDGMENTS

The authors thank Professor David Sedmidubský for helpful comments and critical review of the manuscript.

COMPETING INTERESTS

The authors declare that they have no competing interests.

AUTHOR CONTRIBUTIONS

Both authors developed UCM and prepared the examples. Each file related to the format contains information about its authors.

Jan Mokrý wrote the manuscript, developed all additional software tools and prepared the website (http://www.universalchemicalmarkup.org).

Miloslav Nič reviewed drafts of the manuscript, provided supervision and advice.

FUNDING

There was no external funding for this project. All financial resources needed to carry out this research were contributed by one of the authors, Jan Mokrý.

REFERENCES

1 MOKRÝ, J.; NIČ, M. Designing Universal Chemical Markup (UCM) through the reusable methodology based on analyzing existing related formats. *Journal of Cheminformatics*. In press.

2 DALBY, A. et al. Description of several chemical structure file formats used by computer programs developed at Molecular Design Limited. *J. Chem. Inf. Comput. Sci.* 1992, vol. 32, pp. 244–255. Available also at: ⟨http://dx.doi.org/10.1021/CI00007A012⟩.

3 *CTfile Formats*. Accelrys. Available at: ⟨http://accelrys.com/products/informatics/cheminformatics/ctfile-formats/no-fee.php⟩.

4 *Symyx - CTFile Formats*. 2007. Available at: ⟨http://web.uni-plovdiv.bg/ksx/students/ISIS/ctfile.pdf⟩.

5 *The CDXML text-based file format*. CambridgeSoft. Available at: ⟨http://www.cambridgesoft.com/services/documentation/sdk/chemdraw/cdx/IntroCDXML.htm⟩.

6 LEE, D.; CHU, W. W. *Comparative Analysis of Six XML Schema Languages*. Department of Computer Science University of California, 2000. Available at: ⟨http://www.cobase.cs.ucla.edu/tech-docs/dongwon/ucla-200008.html⟩.

7 MURRAY-RUST, P.; RZEPA, H. S. CML: Evolution and design. *Journal of Cheminformatics*. 2011, vol. 3, pp. 44. Available also at: ⟨http://dx.doi.org/10.1186/1758-2946-3-44⟩.

8 MURRAY-RUST, P.; RZEPA, H. S. Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles. *J. Chem. Inf. Comput. Sci.* 1999, vol. 39, pp. 928–942. Available also at: ⟨http://dx.doi.org/10.1021/CI990052B⟩.

9 MURRAY-RUST, P.; RZEPA, H. S. *What is CML?* Available at: ⟨http://www.ch.ic.ac.uk/rzepa/chimeral/documents/FAQ/FAQ.html⟩.

10 MURRAY-RUST, P.; RZEPA, H. S. Chemical Markup, XML, and the World Wide Web. 4. CML Schema. *J. Chem. Inf. Comput. Sci.* 2003, vol. 43, pp. 757–772. Available also at: ⟨http://dx.doi.org/10.1021/CI0256541⟩.

11 *Schemas for CML and Related Languages*. 2005. Available at: ⟨http://cml.sourceforge.net/schema⟩.

12 MURRAY-RUST, P.; RZEPA, H. S. *Chemical Markup Language Schema*. Available at: ⟨http://www.xml-cml.org/schema/⟩.

13 MURRAY-RUST, P.; RZEPA, H. S. *Chemical Markup Language Specifications*. Available at: ⟨http://www.xml-cml.org/spec⟩.

14 MURRAY-RUST, P.; RZEPA, H. S.; WRIGHT, M. Development of chemical markup language (CML) as a system for handling complex chemical content. *New J. Chem.* 2001, vol. 25, pp. 618–634. Available also at: ⟨http://dx.doi.org/10.1039/B008780G⟩.

15 TOWNSEND, J. A.; MURRAY-RUST, P. CMLLite: a design philosophy for CML. *Journal of Cheminformatics*. 2011, vol. 3, pp. 39. Available also at: ⟨http://dx.doi.org/10.1186/1758-2946-3-39⟩.

16 BRAY, T. et al. *Namespaces in XML 1.0 (Third Edition)*. The World Wide Web Consortium (W3C), 2009. Available at: ⟨http://www.w3.org/TR/2009/REC-xml-names-20091208⟩.

17 BRAY, T. et al. *Namespaces in XML 1.1 (Second Edition)*. The World Wide Web Consortium (W3C), 2006. Available at: ⟨http://www.w3.org/TR/2006/REC-xml-names11-20060816⟩.

18 PAWSON, D. *An introduction to NVDL, ISO 19757-4*. 2008. Available at: ⟨http://www.dpawson.co.uk/nvdl/index.html⟩.

19 *ISO/IEC 19757-4 NVDL (Namespace-based Validation Dispatching Language)*. 2009. Available at: ⟨http://nvdl.org⟩.

20 GUNDERSEN, V. B.; HENDRIKSE, Z. W. *BibTeX as XML markup*. 2007. Available at: ⟨http://bibtexml.sourceforge.net⟩.

21 *Units Markup Language (UnitsML)*. National Institute of Standards and Technology (NIST), 2011. Available at: ⟨http://unitsml.nist.gov⟩.

22 CLARK, J. *Trang – Multi-format schema converter based on RELAX NG*. Thai Open Source Software Center Ltd, 2008. Available at: ⟨http://www.thaiopensource.com/relaxng/trang.html⟩.

23 PEMBERTON, S. et al. *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)*. The World Wide Web Consortium (W3C), 2002. Available at: ⟨http://www.w3.org/TR/2002/REC-xhtml1-20020801⟩.

24 CARLISLE, D.; ION, P.; MINER, R. *Mathematical Markup Language (MathML) Version 3.0 2nd Edition*. The World Wide Web Consortium (W3C), 2014. Available at: ⟨http://www.w3.org/TR/2014/REC-MathML3-20140410⟩.

25 PAWSON, D.; COSTELLO, R.; GEORGES, F. *ISO Schematron tutorial*. 2007. Available at: ⟨http://www.dpawson.co.uk/schematron/index.html⟩.

26 *ISO/IEC 19757-4:2006 Information technology – Document Schema Definition Languages (DSDL) – Part 4: Namespace-based Validation Dispatching Language (NVDL)*. International Organization for Standardization (ISO), 2006. Available at: ⟨http://standards.iso.org/ittf/PubliclyAvailableStandards/c038615_ISO_IEC_19757-4_2006(E).zip⟩.

27 *Schematron – A language for making assertions about patterns found in XML documents*. Available at: ⟨http://www.schematron.com/index.html⟩.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1336v1 | CC-BY 4.0 Open Access | rec: 30 Aug 2015, publ: 30 Aug 2015

28 ROBERTSSON, E. *An Introduction to Schematron*. O'Reilly XML.com, 2003. Available at: ⟨http://www.xml.com/pub/a/2003/11/12/schematron.html⟩.

29 ROBERTSSON, E. *Combining RELAX NG and Schematron*. O'Reilly XML.com, 2004. Available at: ⟨http://www.xml.com/pub/a/2004/02/11/relaxtron.html?page=1⟩.

30 HANWELL, M. D. et al. Avogadro: An advanced semantic chemical editor, visualization, and analysis platform. *Journal of Cheminformatics*. 2012, vol. 4, pp. 17. Available also at: ⟨http://dx.doi.org/10.1186/1758-2946-4-17⟩.

31 *Avogadro: An open-source molecular builder and visualization tool*. 2012. Available at: ⟨http://avogadro.openmolecules.net⟩.

32 VALIEV, M. et al. NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*. 2010, vol. 181, pp. 1477–1489. Available also at: ⟨http://dx.doi.org/10.1016/J.CPC.2010.04.018⟩.

33 *NWChem: Delivering high-performance computational chemistry*. 2014. Available at: ⟨http://www.nwchem-sw.org⟩.

34 RESIG, J. et al. *jQuery*. The jQuery Foundation, 2014. Available at: ⟨http://jquery.com⟩.

35 CABELLO, R. et al. *three.js - JavaScript 3D library*. Available at: ⟨http://threejs.org⟩.

36 MARSH, J.; ORCHARD, D.; VEILLARD, D. *XML Inclusions (XInclude) Version 1.0 (Second Edition)*. The World Wide Web Consortium (W3C), 2006. Available at: ⟨http://www.w3.org/TR/2006/REC-xinclude-20061115⟩.

37 *CDX Format Specification*. CambridgeSoft. Available at: ⟨http://www.cambridgesoft.com/services/documentation/sdk/chemdraw/cdx/index.htm⟩.

38 MILIORDOS, E.; XANTHEAS, S. S. On the Bonding Nature of Ozone (O3) and Its Sulfur-Substituted Analogues SO2, OS2, and S3: Correlation between Their Biradical Character and Molecular Properties. *J. Am. Chem. Soc.* 2014, vol. 136, pp. 2808–2817. Available also at: ⟨http://dx.doi.org/10.1021/JA410726U⟩.

39 ABEL, E. W. et al. Dynamic NMR studies of ring rotation in substituted ferrocenes and ruthenocenes. *Journal of Organometallic Chemistry*. 1991, vol. 403, pp. 195–208. Available also at: ⟨http://dx.doi.org/10.1016/0022-328X(91)83100-I⟩.

40 BLACK, M.; MAIS, R. H. B.; OWSTON, P. G. The Crystal and Molecular Structure of Zeise's Salt, KPtCl3.C2H4.H2O. *Acta Crystallographica Section B Structural Crystallography and Crystal Chemistry*. 1969, vol. 25, pp. 1753–1759. Available also at: ⟨http://dx.doi.org/10.1107/S0567740869004699⟩.

41 KATÔ, H. The Electronic Structure of Zeise's Salt, [PtCl3C2H4]-. *Bulletin of the Chemical Society of Japan*. 1971, vol. 44, pp. 348–354. Available also at: ⟨http://dx.doi.org/10.1246/BCSJ.44.348⟩.

42 *Software for CIF and STAR*. The International Union of Crystallography (IUCr). Available at: ⟨http://www.iucr.org/resources/cif/software⟩.

43 *IUCr checkCIF*. The International Union of Crystallography (IUCr). Available at: ⟨http://checkcif.iucr.org⟩.

44 MURRAY-RUST, P. et al. The semantics of Chemical Markup Language (CML): dictionaries and conventions. *Journal of Cheminformatics*. 2011, vol. 3, pp. 43. Available also at: ⟨http://dx.doi.org/10.1186/1758-2946-3-43⟩.

45 O'BOYLE, N. M. et al. Open Babel: An open chemical toolbox. *Journal of Cheminformatics*. 2011, vol. 3, pp. 33. Available also at: ⟨http://dx.doi.org/10.1186/1758-2946-3-33⟩.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1336v1 | CC-BY 4.0 Open Access | rec: 30 Aug 2015, publ: 30 Aug 2015

FIGURES

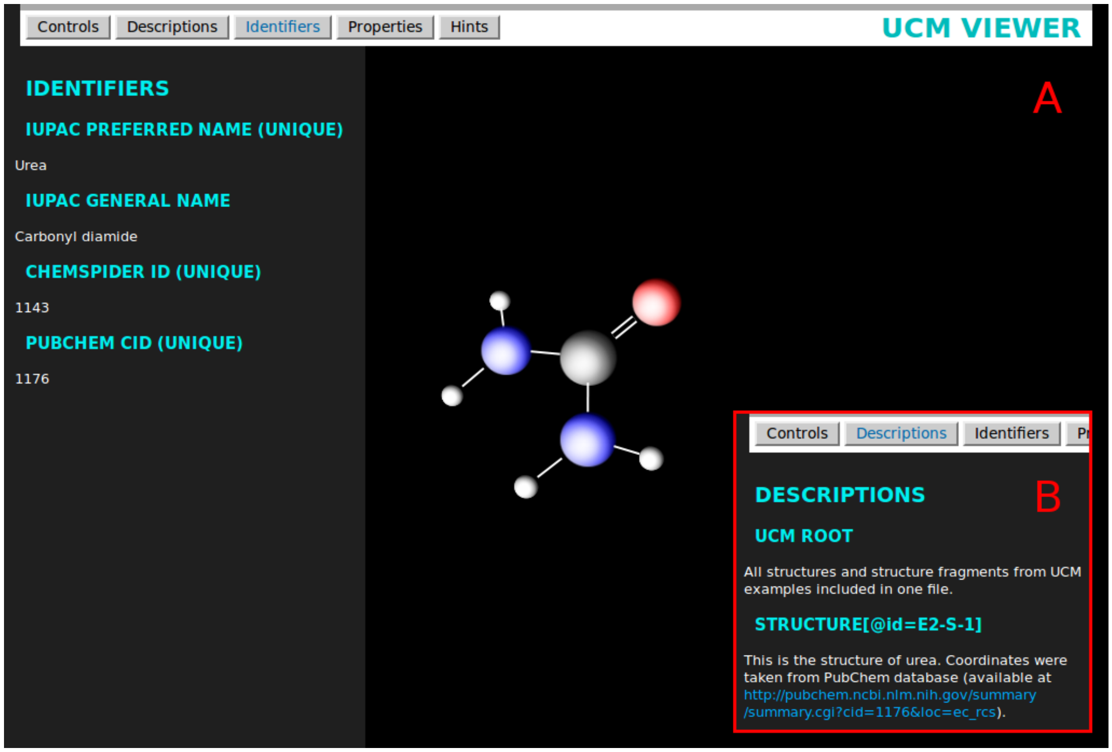


Figure 1. The screenshot of UCM VIEWER with loaded UCM example 2 shows the structure of urea and its chemical identifiers (A). An annotation loaded from a description element is depicted in bottom right corner (B).

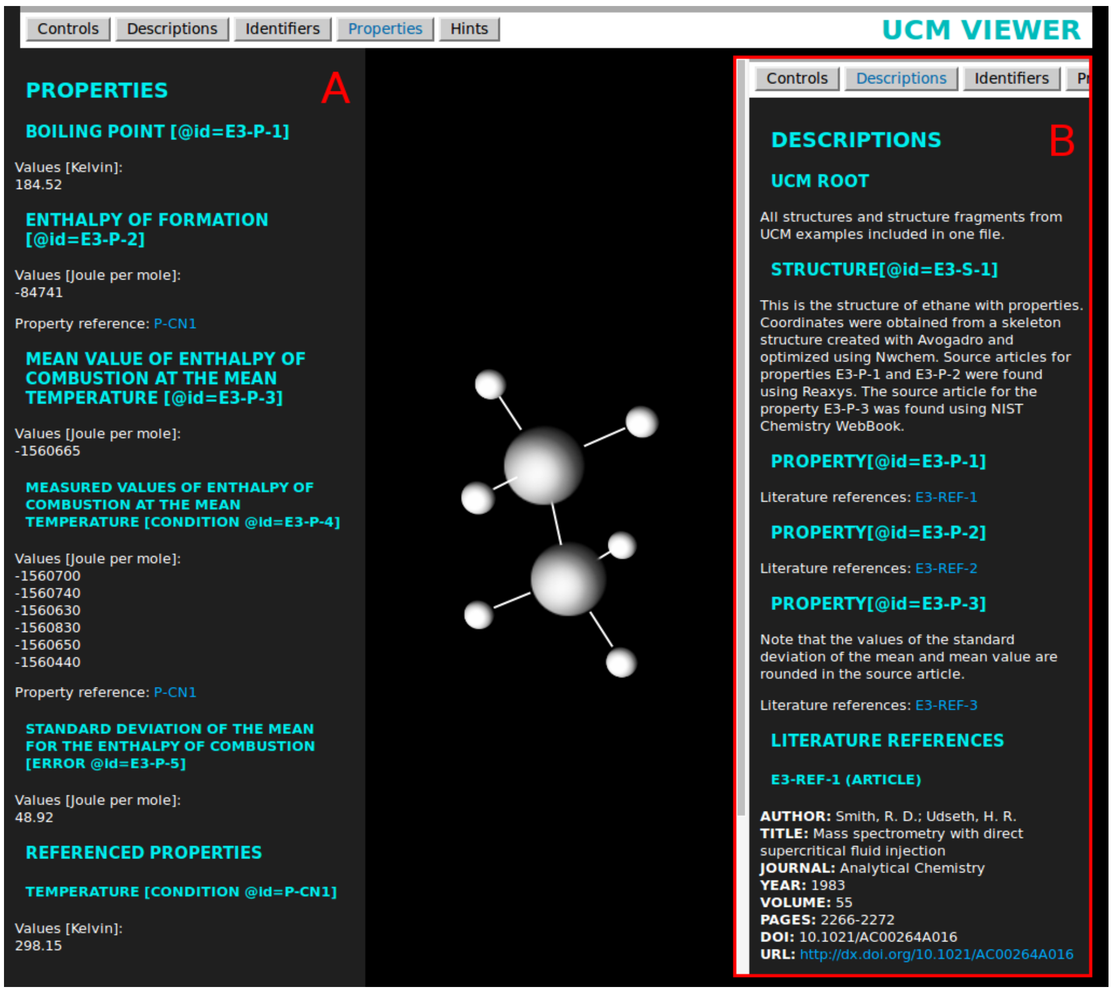


Figure 2. The screenshot of UCM VIEWER with loaded UCM example 3 shows the structure of ethane and its properties (A). Annotations with literature references loaded from description elements are depicted on right side (B).

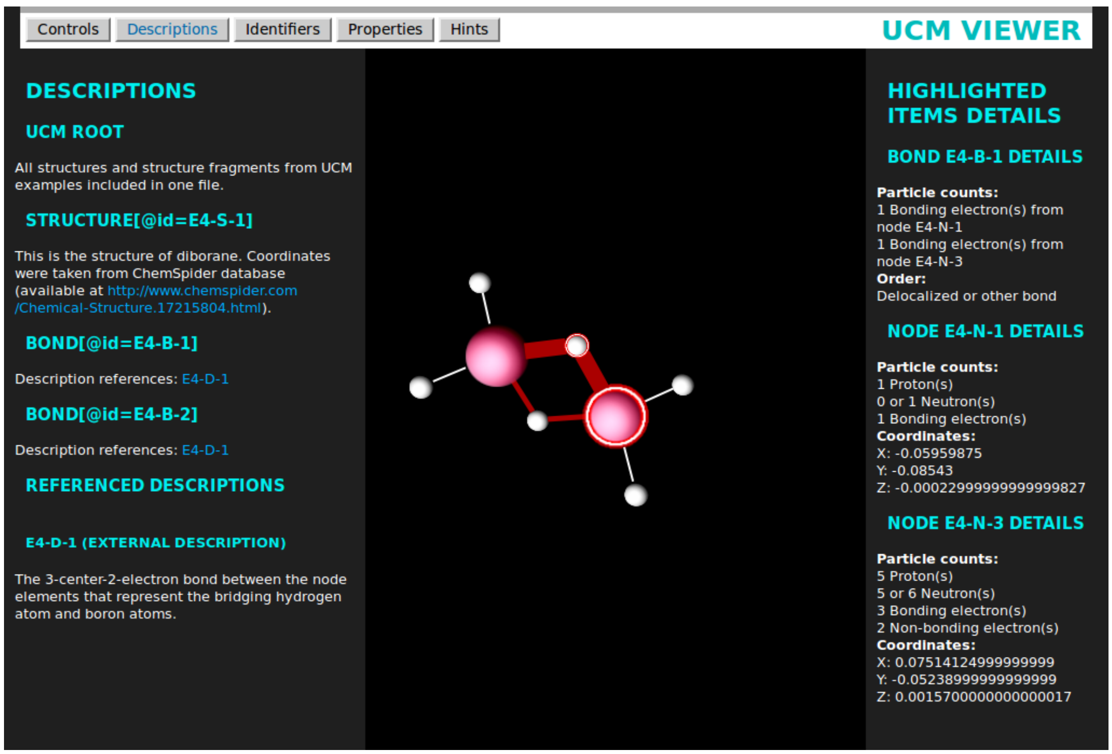


Figure 3. The screenshot of UCM VIEWER with loaded UCM example 4 shows the structure of diborane with 3-center-2-electron bonds.

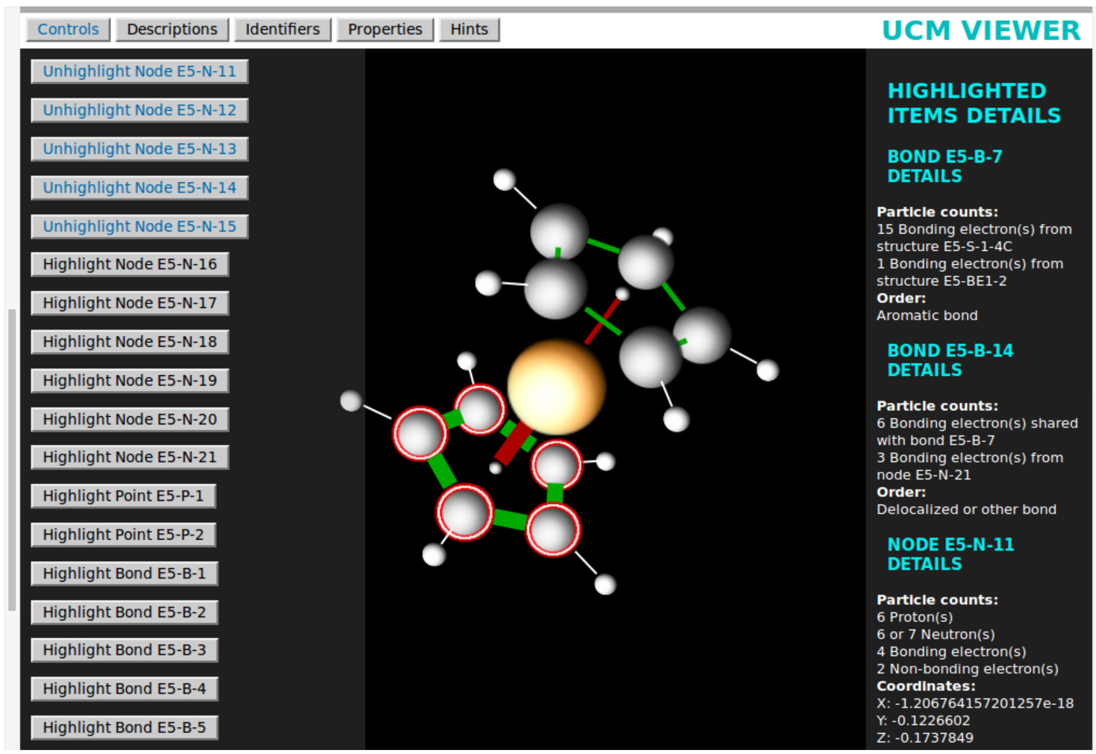


Figure 4. The screenshot of UCM VIEWER with loaded UCM example 5 depicts the structure of ferrocene with aromatic and delocalized bonds.

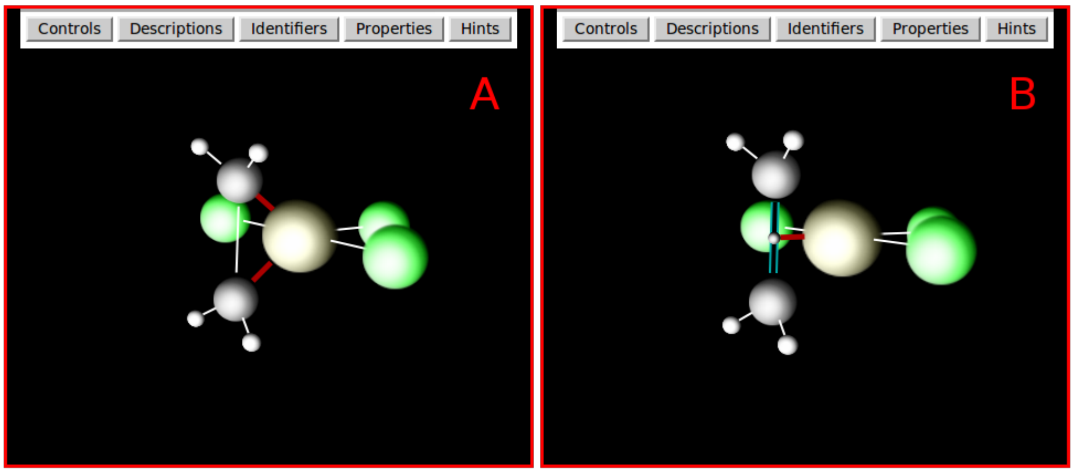


Figure 5. The screenshot of UCM VIEWER with loaded UCM example 6 depicts both structures of trichloro(ethene)platinate(II) anion (Zeise's salt anion) we recorded in UCM. In (A) the 3-center-2-electron bond connects the central platinum atom and ethylene ligand (a single bond is between the ligand carbon atoms). In (B) the central platinum atom and ethylene ligand are connected using a delocalized bond, which shares an electron with the partial double bond between the ligand carbon atoms.

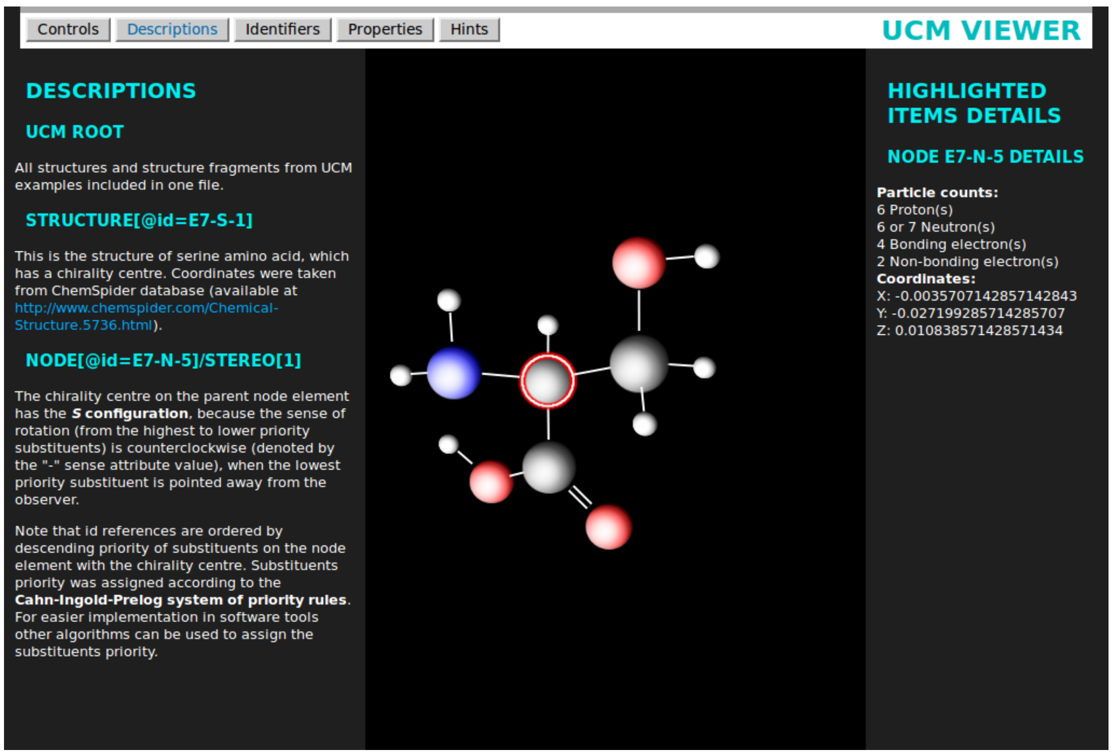


Figure 6. The screenshot of UCM VIEWER with loaded UCM example 7 shows the stereochemistry of the chirality centre in serine amino acid.

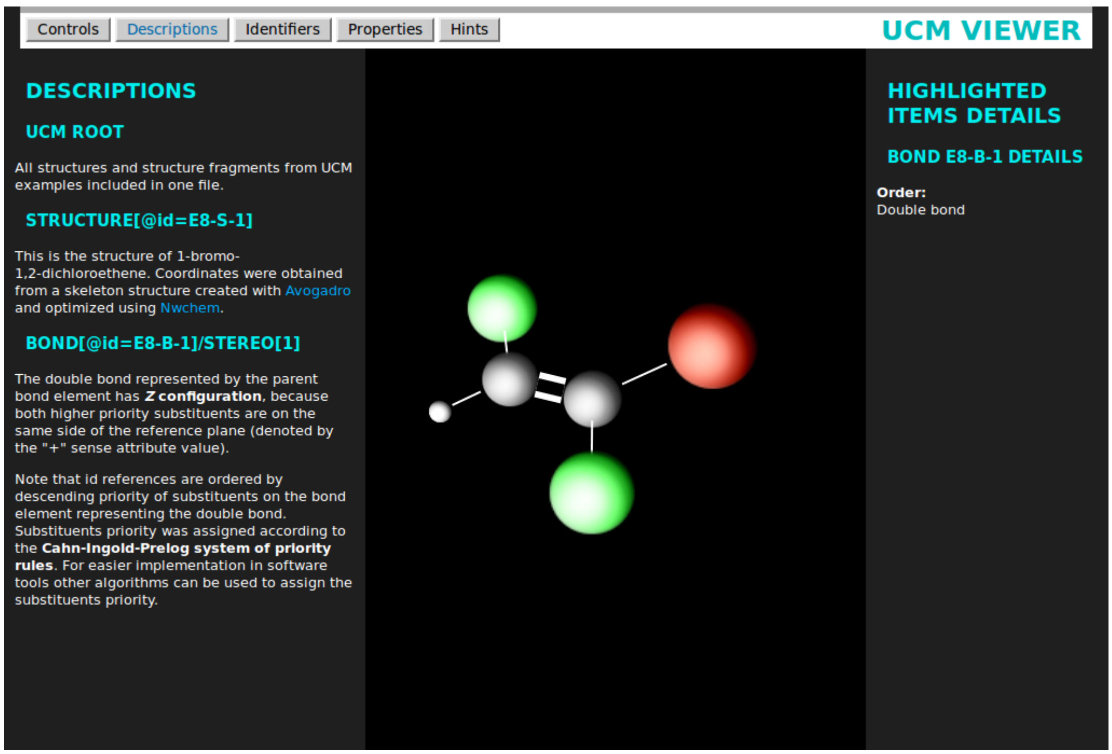


Figure 7. The screenshot of UCM VIEWER with loaded UCM example 8 shows the stereochemistry of the double bond in 1-bromo-1,2-dichloroethene.

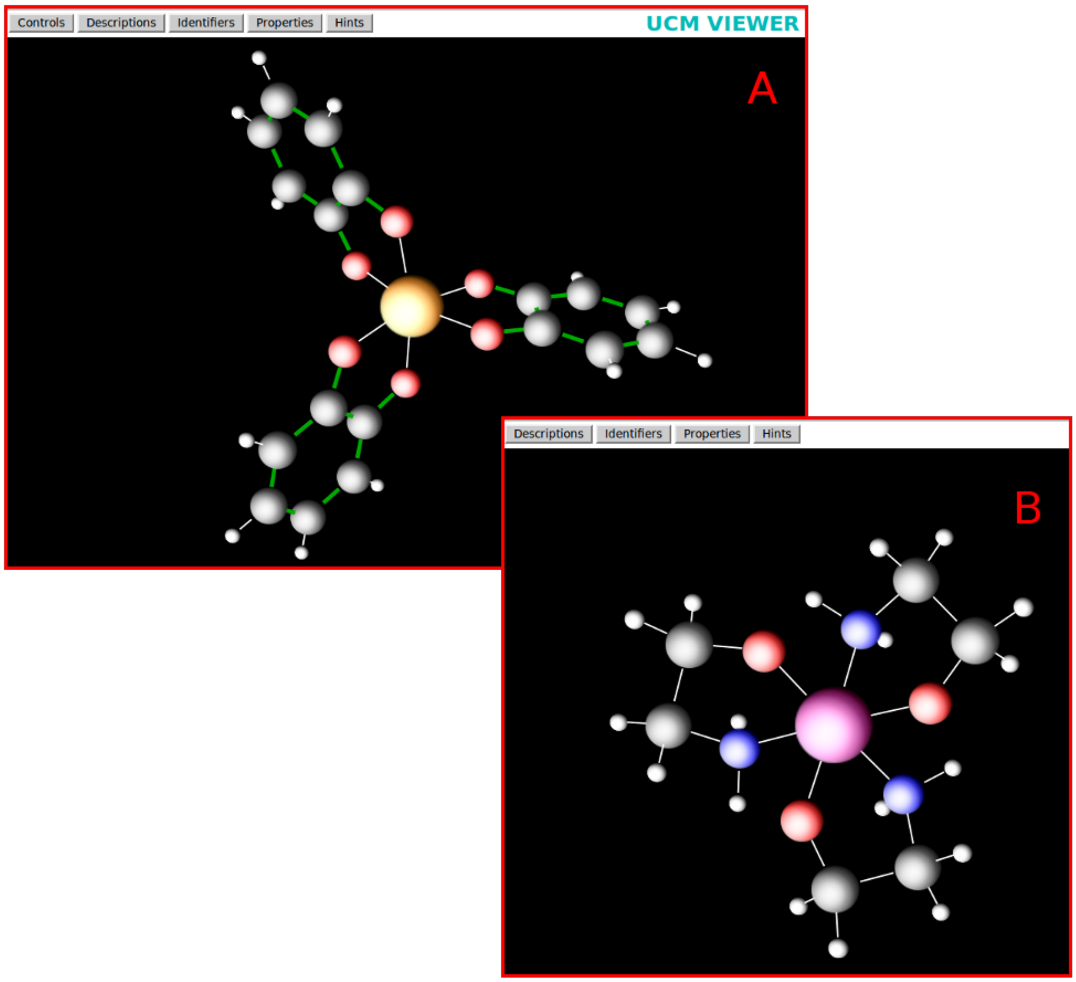


Figure 8. Screenshots of UCM VIEWER with loaded UCM examples 9 and 10 show the structure of Lambda tris(catecholato)ferrate(III) anion (A) and Lambda tris(1-hydroxy-2-aminoethane)cobalt(III) complex (B).

TREE STRUCTURE SCHEMES

Tree Structure Scheme 1 The first part of overall structure for UCM 1-1-1 shows the *ucm* root element and its possible content: zero or one *description* element as the first child followed by zero or more *define* and *structure* child elements.

```
ucm (@id?, @version)
    description [C1]? ...

    define [C1: @format = 'UCM'] (@id?, @format)*
        description [C2]* ...
        property [C1]* ...
        node [C1]* ...
    OR
    define [C2: @format = 'BIBTEXML'] (@id?, @format)*
        BIBTEXML*
    OR
    define [C3: @format = 'UNITSML'] (@id?, @format)*
        UNITSML*

    structure [C1: not(@format = 'UCM')] (@id, @format, @type)*
        IUPAC-PREFERRED-NAME-U OR IUPAC-GENERAL-NAME OR CA-INDEX-NAME
        OR CAS-RN-U OR REAXYS-RN-U
        OR CHEMSPIDER-ID-U OR PUBCHEM-CID-U OR PUBCHEM-SID
        OR INCHI OR INCHI-KEY OR S-INCHI-U OR S-INCHI-KEY
        OR SMILES OR SMARTS OR SLN
    OR
    structure [C2: @format = 'UCM'] (@id, @format, @type, @charge IF NOT 0)*
        description [C1]? ...
        structure [C1]* ... OR structure [C2]* ...
        property [C1]* ... OR property [C2]* ...
        node [C2]* ... OR node [C3]* ...
        bond [C1]* ... OR bond [C2]* ...
        point (@id, @x, @y, @z)*
            description [C1]? ...
            property [C1]* ... OR property [C2]* ...
        stereo* ...
```

Tree Structure Scheme 2 The second part of overall structure for UCM 1-1-1 shows the remaining UCM elements with their contents.

```
description [C1: not(parent::define)] (@id?, @idrefs?, @litrefs?)
    XHTML* OR PLAINTEXT*

description [C2: parent::define] (@id, @idrefs?, @litrefs?)
    XHTML* OR PLAINTEXT*

property [C1: not(@idrefs)] (@id, @type, @quantity)
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...
    values (@id?)?

property[C2: @idrefs] (@id, @idrefs)
    description [C1]? ...

node [C1: parent::define] (@id)
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...
    particle [C1]+ ...

node [C2: parent::structure and @idrefs]
        (@id, @idrefs, @charge IF NOT 0, @x?, @y?, @z?)
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...
    stereo? ...

node [C3: parent::structure and not(@idrefs)]
        (@id, @charge IF NOT 0, @x?, @y?, @z?)
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...
    particle [C1]+ ...
    stereo? ...

bond [C1: not(@idrefs)] (@id, @order)
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...
    join (@id?, @idrefs)+
    particle [C2]* ...
    stereo? ...

bond [C2: @idrefs] (@id, @idrefs, @order)
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...
    particle [C2]* ...
    stereo? ...

particle [C1: parent::node] (@id?, @type, @counts, @fractions? IF @type 'N')
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...

particle [C2: parent::bond] (@id?, @idrefs, @type, @counts)
    description [C1]? ...
    property [C1]* ... OR property [C2]* ...
    share (@id?, @idrefs, @fractions)*

stereo (@id?, @idrefs, @sense)
    description [C1]? ...
```

SOURCE CODE SNIPPETS

Source Code Snippet 1 The simplified source code of BibTeXML literature references definitions. Omitted parts are indicated by ellipses.

```
<define xmlns="http://www.universalchemicalmarkup.org"
        xmlns:bibtexml="http://bibtexml.sf.net/"
        format="BIBTEXML">

    <bibtexml:file>
        <bibtexml:entry id="REF-0-1">
            <bibtexml:article>
                <bibtexml:author>Berglund, M. and Wieser, M. E.</bibtexml:author>
                <bibtexml:title>Isotopic compositions of the elements...</bibtexml:title>
                <bibtexml:journal>Pure and Applied Chemistry</bibtexml:journal>
                <bibtexml:year>2011</bibtexml:year>
                <bibtexml:volume>83</bibtexml:volume>
                <bibtexml:pages>397-410</bibtexml:pages>
                <bibtexml:doi>10.1351/PAC-REP-10-06-02</bibtexml:doi>
                <bibtexml:url>http://dx.doi.org/10.1351/PAC-REP-10-06-02</bibtexml:url>
            </bibtexml:article>
        </bibtexml:entry>
        ...
    </bibtexml:file>

</define>
```

Source Code Snippet 2 The simplified source code of UnitsML quantities and scientific units definitions. Omitted parts are indicated by ellipses.

```
<define xmlns="http://www.universalchemicalmarkup.org"
        xmlns:unitsml="urn:oasis:names:tc:unitsml:schema:xsd:UnitsMLSchema-1.0"
        format="UNITSML">

    <unitsml:UnitSet>
        <unitsml:Unit xml:id="JoulePerMole">
            <unitsml:UnitName xml:lang="en-US">Joule per mole</unitsml:UnitName>
            <unitsml:RootUnits>
                <unitsml:EnumeratedRootUnit unit="joule"/>
                <unitsml:EnumeratedRootUnit powerNumerator="-1" unit="mole"/>
            </unitsml:RootUnits>
        </unitsml:Unit>
        ...
    </unitsml:UnitSet>

    <unitsml:QuantitySet>
        ...
        <unitsml:Quantity xml:id="EnthalpyOfFormation">
            <unitsml:QuantityName xml:lang="en-US">Enthalpy of formation</unitsml:QuantityName>
            <unitsml:UnitReference url="#JoulePerMole"/>
        </unitsml:Quantity>
        ...
    </unitsml:QuantitySet>

</define>
```

Source Code Snippet 3 The simplified source code of UCM chemical nodes definitions. Omitted parts are indicated by ellipses.

```
<define xmlns="http://www.universalchemicalmarkup.org" format="UCM">
    ...
    <node id="H-PLUS1-BE0">
        <particle type="P" counts="1"/>
        <particle type="N" counts="0 1" fractions="0.999885 0.000115"/>
    </node>

    <node id="H2-BE1">
        <particle type="P" counts="1"/>
        <particle type="N" counts="1"/>
        <particle type="BE" counts="1"/>
    </node>
    ...
    <node id="C-BE4">
        <particle type="P" counts="6"/>
        <particle type="N" counts="6 7" fractions="0.9893 0.0107"/>
        <particle type="BE" counts="4"/>
        <particle type="NBE" counts="2"/>
    </node>
    ...
</define>
```

Source Code Snippet 4 The simplified source code for the Schematron validation of UCM *charge* attributes. Omitted parts are indicated by ellipses.

```
...
<pattern name="UCM:*@CHARGE">
    ...
    <rule context="//ucm:node[parent::ucm:structure and @idrefs]">
        <let name="ID" value="./@id"/>
        <let name="IDREFS" value="./@idrefs"/>
        <let name="NODE-DEFINITION" value="//ucm:node[parent::ucm:define and @id = $IDREFS]"/>
        <let name="DEFINED-ELECTRON-COUNT"
             value="xs:integer(sum($NODE-DEFINITION/ucm:particle[contains(@type,
                    'E')]/@counts))"/>
        <let name="SHARED-ELECTRON-COUNT"
             value="xs:integer(sum(//ucm:particle[ucm:share and @idrefs = $ID]/@counts))"/>
        <let name="RELEVANT-SHARE-ELEMENTS" value="//ucm:share[contains(@idrefs, $ID)]"/>
        <let name="NEGATIVE-CHARGE"
             value="lf:COUNT-NODE-NEGATIVE-CHARGE($ID, $DEFINED-ELECTRON-COUNT,
                    $SHARED-ELECTRON-COUNT, $RELEVANT-SHARE-ELEMENTS)"/>
        <let name="POSITIVE-CHARGE"
             value="sum($NODE-DEFINITION/ucm:particle[@type = 'P']/@counts)"/>
        <let name="TOTAL-CHARGE" value="$POSITIVE-CHARGE - $NEGATIVE-CHARGE"/>
        <let name="CURRENT-CHARGE" value="if (./@charge) then ./@charge else 0"/>
        <assert test="$TOTAL-CHARGE = $CURRENT-CHARGE">
            <value-of select="lf:OUTPUT-ERROR-MESSAGE(.,
                              (concat('The charge attribute value on this element must be ',
                              $TOTAL-CHARGE, ', because it contains'),
                              concat($POSITIVE-CHARGE, ' proton(s) and ', $NEGATIVE-CHARGE,
                              ' electron(s).')))"/>
        </assert>
    </rule>
    ...
</pattern>
...
<xsl:function name="lf:COUNT-NODE-NEGATIVE-CHARGE" as="xs:decimal">

    <xsl:param name="NODE-ID" as="xs:string"/>
    <xsl:param name="DEFINED-ELECTRON-COUNT" as="xs:integer"/>
    <xsl:param name="SHARED-ELECTRON-COUNT" as="xs:integer"/>
    <xsl:param name="RELEVANT-SHARE-ELEMENTS" as="node()*"/>

    <xsl:variable name="FRACTIONS-OF-SHARED-ELECTRON-COUNTS">
        <xsl:for-each select="$RELEVANT-SHARE-ELEMENTS">
            <xsl:value-of select="xs:decimal(subsequence(tokenize(./@fractions, '\s+'),
                                  index-of(tokenize(./@idrefs, '\s+'), $NODE-ID), 1))
                                  * xs:decimal(./parent::ucm:particle/@counts)"/>
            <xsl:if test="position() != last()">
                <xsl:text> </xsl:text>
            </xsl:if>
        </xsl:for-each>
    </xsl:variable>

    <xsl:value-of select="$DEFINED-ELECTRON-COUNT - $SHARED-ELECTRON-COUNT + sum(
                          for $i in tokenize($FRACTIONS-OF-SHARED-ELECTRON-COUNTS,' ')
                          return xs:decimal($i))"/>

</xsl:function>
...
```

Source Code Snippet 5 The simplified source code for the Schematron validation of bonding electrons in UCM. Omitted parts are indicated by ellipses.

```
...
<pattern name="UCM:NODE-2">

    <rule context="//ucm:node[parent::ucm:structure and @idrefs]">
        <let name="IDREFS" value="./@idrefs"/>
        <let name="NODE-BONDING-ELECTRONS"
             value="sum(//ucm:node[parent::ucm:define and @id = $IDREFS]/ucm:particle[
                    @type = 'BE']/@counts)"/>
        <let name="REQUIRED-NODE-BONDING-ELECTRONS"
             value="lf:COUNT-NODE-BONDING-ELECTRONS(.,
                    //ucm:bond[not(ucm:particle)],
                    //ucm:particle[@type = 'BE'])"/>
        <assert test="$NODE-BONDING-ELECTRONS = $REQUIRED-NODE-BONDING-ELECTRONS">
            <value-of select="lf:OUTPUT-ERROR-MESSAGE(.,
                              (concat('The node definition of this element contains ',
                              $NODE-BONDING-ELECTRONS, ' bonding electron(s), but all'),
                              concat('bonds of this element require ',
                              $REQUIRED-NODE-BONDING-ELECTRONS, ' bonding electron(s).')))"/>
        </assert>
    </rule>
    ...
</pattern>
...
<xsl:function name="lf:COUNT-NODE-BONDING-ELECTRONS" as="xs:integer">

    <xsl:param name="NODE-ELEMENT" as="node()"/>
    <xsl:param name="RELEVANT-BOND-ELEMENTS" as="node()*"/>
    <xsl:param name="RELEVANT-PARTICLE-ELEMENTS" as="node()*"/>

    <xsl:variable name="PARENT-STRUCTURE-ELEMENT"
                  select="$NODE-ELEMENT/parent::ucm:structure"/>
    <xsl:variable name="PARENT-STRUCTURE-ELEMENT-ID-REGEX"
                  select="concat('(^| )', $PARENT-STRUCTURE-ELEMENT/@id, '( |$)')"/>
    <xsl:variable name="NODE-ELEMENT-ID-REGEX"
                  select="concat('(^| )', $NODE-ELEMENT/@id, '( |$)')"/>
    <xsl:variable name="NODE-COUNT" select="count($PARENT-STRUCTURE-ELEMENT/ucm:node)"/>

    <xsl:variable name="SINGLE-BONDS-ELECTRON-COUNT"
                  select="count($RELEVANT-BOND-ELEMENTS[matches(@idrefs,
                          $NODE-ELEMENT-ID-REGEX) and @order = 'S'])"/>
    <xsl:variable name="DOUBLE-BONDS-ELECTRON-COUNT"
                  select="count($RELEVANT-BOND-ELEMENTS[matches(@idrefs,
                          $NODE-ELEMENT-ID-REGEX) and @order = 'D']) * 2"/>
    <xsl:variable name="TRIPLE-BONDS-ELECTRON-COUNT"
                  select="count($RELEVANT-BOND-ELEMENTS[matches(@idrefs,
                          $NODE-ELEMENT-ID-REGEX) and @order = 'T']) * 3"/>
    <xsl:variable name="QUADRUPLE-BONDS-ELECTRON-COUNT"
                  select="count($RELEVANT-BOND-ELEMENTS[matches(@idrefs,
                          $NODE-ELEMENT-ID-REGEX) and @order = 'Q']) * 4"/>
    <xsl:variable name="OTHER-BONDS-ELECTRON-COUNT"
                  select="sum($RELEVANT-PARTICLE-ELEMENTS[matches(@idrefs,
                          $NODE-ELEMENT-ID-REGEX)]/@counts)
                          + (sum($RELEVANT-PARTICLE-ELEMENTS[matches(@idrefs,
                          $PARENT-STRUCTURE-ELEMENT-ID-REGEX)]/@counts) div $NODE-COUNT)"/>

    <xsl:value-of select="$SINGLE-BONDS-ELECTRON-COUNT + $DOUBLE-BONDS-ELECTRON-COUNT
                          + $TRIPLE-BONDS-ELECTRON-COUNT + $QUADRUPLE-BONDS-ELECTRON-COUNT
                          + $OTHER-BONDS-ELECTRON-COUNT"/>

</xsl:function>
...
```

Source Code Snippet 6 The simplified source code of UCM example 1 with small structures and structure fragments without 3-dimensional coordinates. Omitted parts are indicated by ellipses.

```
<ucm xmlns="http://www.universalchemicalmarkup.org"
     xmlns:xi="http://www.w3.org/2001/XInclude" version="1-1-1">
...
<xi:include href="Definitions-1-1-UCM-NODES.xml"/>
...
<structure id="E1-S-1" format="UCM" type="ST" charge="1">
    ...
    <node id="E1-N-1-1" idrefs="H-BE1" charge="0.5"/>
    <node id="E1-N-1-2" idrefs="H-PLUS1-BE0" charge="0.5"/>

    <bond id="E1-B-1-1" idrefs="E1-N-1-1 E1-N-1-2" order="PS">
        <particle idrefs="E1-N-1-1" type="BE" counts="1">
            <share idrefs="E1-N-1-1 E1-N-1-2" fractions="0.5 0.5"/>
        </particle>
    </bond>

</structure>

<structure id="E1-S-2" format="UCM" type="ST">
    ...
    <node id="E1-N-2-1" idrefs="O-BE2"/>
    <node id="E1-N-2-2" idrefs="H2-BE1"/>
    <node id="E1-N-2-3" idrefs="H2-BE1"/>

    <bond id="E1-B-2-1" idrefs="E1-N-2-1 E1-N-2-2" order="S"/>
    <bond id="E1-B-2-2" idrefs="E1-N-2-1 E1-N-2-3" order="S"/>

</structure>
...
</ucm>
```

Source Code Snippet 7 The simplified source code of UCM example 2 with the structure of urea and its chemical identifiers. Omitted parts are indicated by ellipses.

```
<ucm xmlns="http://www.universalchemicalmarkup.org"
     xmlns:xhtml="http://www.w3.org/1999/xhtml"
     xmlns:xi="http://www.w3.org/2001/XInclude" version="1-1-1">
...
<structure id="E2-S-1" format="UCM" type="ST">

    <description><xhtml:p>This is the structure of urea. Coordinates were taken from PubChem
    database (available at <xhtml:a href="http://...">...</xhtml:a>).</xhtml:p></description>

    <structure id="E2-S-1-1" format="IUPAC-PREFERRED-NAME-U" type="STID">Urea</structure>
    ...
    <structure id="E2-S-1-4" format="PUBCHEM-CID-U" type="STID">1176</structure>

    <node id="E2-N-1" idrefs="N-BE3" x="0.06903" y="-0.11479" z="0.00001"/>
    <node id="E2-N-2" idrefs="C-BE4" x=... y=... z=.../>
    <node id="E2-N-3" idrefs="N-BE3" x=... y=... z=.../>
    <node id="E2-N-4" idrefs="O-BE2" x=... y=... z=.../>
    <node id="E2-N-5" idrefs="H-BE1" x=... y=... z=.../>
    <node id="E2-N-6" idrefs="H-BE1" x=... y=... z=.../>
    <node id="E2-N-7" idrefs="H-BE1" x=... y=... z=.../>
    <node id="E2-N-8" idrefs="H-BE1" x=... y=... z=.../>

    <bond id="E2-B-1" idrefs="E2-N-1 E2-N-2" order="S"/>
    <bond id="E2-B-2" idrefs="E2-N-1 E2-N-5" order="S"/>
    <bond id="E2-B-3" idrefs="E2-N-1 E2-N-6" order="S"/>
    <bond id="E2-B-4" idrefs="E2-N-2 E2-N-3" order="S"/>
    <bond id="E2-B-5" idrefs="E2-N-2 E2-N-4" order="D"/>
    <bond id="E2-B-6" idrefs="E2-N-3 E2-N-7" order="S"/>
    <bond id="E2-B-7" idrefs="E2-N-3 E2-N-8" order="S"/>

</structure>

</ucm>
```

Source Code Snippet 8 The simplified source code of UCM example 3 with the recorded properties of ethane. Omitted parts are indicated by ellipses.

```
...
<xi:include href="Definitions-..."/>
...
<structure id="E3-S-1" format="UCM" type="ST">
    ...
    <property id="E3-P-1" type="PR" quantity="#BoilingPoint">
        <description litrefs="REF-1"/>
        <values>184.52</values>
    </property>

    <property id="E3-P-2" type="PR" quantity="#EnthalpyOfFormation">
        <description litrefs="REF-2"/>
        <property id="E3-P-CN1-1" idrefs="P-CN1"/>
        <values>-84741</values>
    </property>

    <property id="E3-P-3" type="PR" quantity="#MeanValueOfEnthalpyOfCombustion...">
        <description litrefs="REF-3">...</description>
        <property id="E3-P-4" type="CN" quantity="#MeasuredValuesOfEnthalpy...">
            <property id="E3-P-CN1-2" idrefs="P-CN1"/>
            <values>-1560700 -1560740 -1560630 ...</values>
        </property>
        <property id="E3-P-5" type="ER" quantity="#StandardDeviationOfMean...">
            <values>48.92</values>
        </property>
        <values>-1560665</values>
    </property>
    ...
</structure>
...
```

Source Code Snippet 9 The simplified source code of UCM example 4 with 3-center-2-electron bonds in diborane. Omitted parts are indicated by ellipses.

```
...
<structure id="E4-S-1" format="UCM" type="ST">
    ...
    <node id="E4-N-1" idrefs="H-BE1" .../>
    <node id="E4-N-2" idrefs="H-BE1" .../>
    <node id="E4-N-3" idrefs="B-BE3" .../>
    <node id="E4-N-4" idrefs="B-BE3" .../>
    ...
    <bond id="E4-B-1" order="DL">...
        <join idrefs="E4-N-1 E4-N-3 E4-N-4">CT</join>
        <particle idrefs="E4-N-1" type="BE" counts="1"/>
        <particle idrefs="E4-N-3" type="BE" counts="1"/>
    </bond>
    <bond id="E4-B-2" order="DL">...
        <join idrefs="E4-N-2 E4-N-3 E4-N-4">CT</join>
        <particle idrefs="E4-N-2" type="BE" counts="1"/>
        <particle idrefs="E4-N-4" type="BE" counts="1"/>
    </bond>
    ...
</structure>
...
```

Source Code Snippet 10 The simplified source code of UCM example 1 with ozone resonance hybrid and its resonance structures without 3-dimensional coordinates. Omitted parts are indicated by ellipses.

```
...
<structure id="E1-S-3" format="UCM" type="ST">
    ...
    <node id="E1-N-3-1" idrefs="O-BE1" charge="-0.5"/>
    <node id="E1-N-3-2" idrefs="O-BE4" charge="1"/>
    <node id="E1-N-3-3" idrefs="O-BE1" charge="-0.5"/>

    <bond id="E1-B-3-1" order="DL">
        <description>The 3-center-2-electron bond between ... all oxygen atoms.</description>
        <join idrefs="E1-N-3-1 E1-N-3-2 E1-N-3-3">SQ</join>
        <particle idrefs="E1-N-3-2" type="BE" counts="1">
            <share idrefs="E1-N-3-1 E1-N-3-2" fractions="0.5 0.5"/>
        </particle>
        <particle idrefs="E1-N-3-2" type="BE" counts="1">
            <share idrefs="E1-N-3-2 E1-N-3-3" fractions="0.5 0.5"/>
        </particle>
    </bond>
    <bond id="E1-B-3-2" idrefs="E1-N-3-1 E1-N-3-2" order="S"/>
    <bond id="E1-B-3-3" idrefs="E1-N-3-2 E1-N-3-3" order="S"/>

</structure>

<structure id="E1-S-3-1" format="UCM" type="ST">
    ...
    <node id="E1-N-3-1-1" idrefs="O-BE0" charge="-1"/>
    <node id="E1-N-3-1-2" idrefs="O-BE4" charge="1"/>
    <node id="E1-N-3-1-3" idrefs="O-BE2"/>

    <bond id="E1-B-3-1-1" idrefs="E1-N-3-1-1 E1-N-3-1-2" order="S">
        <particle idrefs="E1-N-3-1-2" type="BE" counts="2">
            <share idrefs="E1-N-3-1-1 E1-N-3-1-2" fractions="0.5 0.5"/>
        </particle>
    </bond>
    <bond id="E1-B-3-1-2" idrefs="E1-N-3-1-2 E1-N-3-1-3" order="D"/>

</structure>

<structure id="E1-S-3-2" format="UCM" type="ST">...</structure>

<structure id="E1-S-3-3" format="UCM" type="ST">
    ...
    <node id="E1-N-3-3-1" idrefs="O-BE1"/>
    <node id="E1-N-3-3-2" idrefs="O-BE2"/>
    <node id="E1-N-3-3-3" idrefs="O-BE1"/>

    <bond id="E1-B-3-3-1" idrefs="E1-N-3-3-1 E1-N-3-3-2" order="S"/>
    <bond id="E1-B-3-3-2" idrefs="E1-N-3-3-2 E1-N-3-3-3" order="S"/>

</structure>
...
```

Source Code Snippet 11 The simplified source code of UCM example 5 shows the aromatic bonding in the first cyclopentadienyl ring and also the bond between the central iron node and the first cyclopentadienyl ring. Omitted parts are indicated by ellipses.

```
...
<structure id="E5-S-1" format="UCM" type="ST">
    ...
    <structure id="E5-S-1-3" format="UCM" type="SBST" charge="-1">
        ...
        <structure id="E5-S-1-3C" format="UCM" type="SBST">
            <node id="E5-N-1" idrefs="C-BE4" .../>
            <node id="E5-N-2" idrefs="C-BE4" .../>
            <node id="E5-N-3" idrefs="C-BE4" .../>
            <node id="E5-N-4" idrefs="C-BE4" .../>
            <node id="E5-N-5" idrefs="C-BE4" .../>
        </structure>
        ...
        <node id="E5-BE1-1" charge="-1">...
            <particle type="BE" counts="1"/>
        </node>

        <bond id="E5-B-1" order="A">...
            <join idrefs="E5-N-1 E5-N-2 E5-N-3 E5-N-4 E5-N-5">CC</join>
            <particle idrefs="E5-S-1-3C" type="BE" counts="15"/>
            <particle idrefs="E5-BE1-1" type="BE" counts="1"/>
        </bond>
        ...
        <point id="E5-P-1" x="0" y="0" z="0.173016"/>

    </structure>

    <structure id="E5-S-1-4" format="UCM" type="SBST" charge="-1">...</structure>

    <node id="E5-N-21" idrefs="Fe-PLUS2-BE6" charge="2" ...>...</node>

    <bond id="E5-B-13" idrefs="E5-N-21 E5-P-1" order="DL">...
        <particle idrefs="E5-B-1" type="BE" counts="6"/>
        <particle idrefs="E5-N-21" type="BE" counts="3"/>
    </bond>
    ...
</structure>
...
```

Source Code Snippet 12 The simplified source code of UCM example 6 shows both our approaches at describing the structure of trichloro(ethene)platinate(II) anion (Zeise's salt anion). Omitted parts are indicated by ellipses.

```
...
<structure id="E6-S-1" format="UCM" type="ST" charge="-1">
    ...
    <structure id="E6-S-1C" format="UCM" type="SBST">
        <node id="E6-N-1-1" idrefs="C-BE4" .../>
        <node id="E6-N-1-2" idrefs="C-BE4" .../>
    </structure>
    ...
    <node id="E6-N-1-10" idrefs="Pt-BE2" .../>
    ...
    <bond id="E6-B-1-8" idrefs="E6-N-1-1 E6-N-1-2" order="S">...</bond>
    <bond id="E6-B-1-9" order="DL">...
        <join idrefs="E6-N-1-10 E6-N-1-1 E6-N-1-2">CT</join>
        <particle idrefs="E6-S-1C" type="BE" counts="2"/>
    </bond>

</structure>

<structure id="E6-S-2" format="UCM" type="ST" charge="-1">
    ...
    <structure id="E6-S-2C" format="UCM" type="SBST">
        <node id="E6-N-2-1" idrefs="C-BE4" .../>
        <node id="E6-N-2-2" idrefs="C-BE4" .../>
    </structure>
    ...
    <node id="E6-N-2-10" idrefs="Pt-BE2" .../>
    ...
    <bond id="E6-B-2-8" idrefs="E6-N-2-1 E6-N-2-2" order="PD">...
        <particle idrefs="E6-S-2C" type="BE" counts="3"/>
    </bond>
    <bond id="E6-B-2-9" idrefs="E6-N-2-10 E6-P-2-1" order="DL">...
        <particle idrefs="E6-B-2-8" type="BE" counts="1"/>
        <particle idrefs="E6-S-2C" type="BE" counts="1"/>
    </bond>

    <point id="E6-P-2-1" x="0.4755" y="0.35813" z="-0.10135"/>

</structure>
...
```


PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1336v1 | CC-BY 4.0 Open Access | rec: 30 Aug 2015, publ: 30 Aug 2015

Source Code Snippet 13 The simplified source code of UCM example 7 with the recorded stereochemical configuration of the chirality centre in serine amino acid. Omitted parts are indicated by ellipses.

```
...
<structure id="E7-S-1" format="UCM" type="ST">
    ...
    <node id="E7-N-1" idrefs="O-BE2" .../>
    <node id="E7-N-2" idrefs="O-BE2" .../>
    <node id="E7-N-3" idrefs="O-BE2" .../>
    <node id="E7-N-4" idrefs="N-BE3" .../>
    <node id="E7-N-5" idrefs="C-BE4" ...>
        <stereo idrefs="E7-N-4 E7-N-7 E7-N-6 E7-N-8" sense="-">...</stereo>
    </node>
    <node id="E7-N-6" idrefs="C-BE4" .../>
    <node id="E7-N-7" idrefs="C-BE4" .../>
    <node id="E7-N-8" idrefs="H-BE1" .../>
    <node id="E7-N-9" idrefs="H-BE1" .../>
    <node id="E7-N-10" idrefs="H-BE1" .../>
    <node id="E7-N-11" idrefs="H-BE1" .../>
    <node id="E7-N-12" idrefs="H-BE1" .../>
    <node id="E7-N-13" idrefs="H-BE1" .../>
    <node id="E7-N-14" idrefs="H-BE1" .../>

    <bond id="E7-B-1" idrefs="E7-N-1 E7-N-6" order="S"/>
    <bond id="E7-B-2" idrefs="E7-N-1 E7-N-13" order="S"/>
    <bond id="E7-B-3" idrefs="E7-N-2 E7-N-7" order="S"/>
    <bond id="E7-B-4" idrefs="E7-N-2 E7-N-14" order="S"/>
    <bond id="E7-B-5" idrefs="E7-N-3 E7-N-7" order="D"/>
    <bond id="E7-B-6" idrefs="E7-N-5 E7-N-4" order="S"/>
    <bond id="E7-B-7" idrefs="E7-N-4 E7-N-11" order="S"/>
    <bond id="E7-B-8" idrefs="E7-N-4 E7-N-12" order="S"/>
    <bond id="E7-B-9" idrefs="E7-N-5 E7-N-6" order="S"/>
    <bond id="E7-B-10" idrefs="E7-N-5 E7-N-7" order="S"/>
    <bond id="E7-B-11" idrefs="E7-N-5 E7-N-8" order="S"/>
    <bond id="E7-B-12" idrefs="E7-N-6 E7-N-9" order="S"/>
    <bond id="E7-B-13" idrefs="E7-N-6 E7-N-10" order="S"/>

</structure>
...
```

Source Code Snippet 14 The simplified source code of UCM example 8 with the recorded stereochemical configuration of the double bond in 1-bromo-1,2-dichloroethene. Omitted parts are indicated by ellipses.

```
...
<structure id="E8-S-1" format="UCM" type="ST">
    ...
    <node id="E8-N-1" idrefs="C-BE4" .../>
    <node id="E8-N-2" idrefs="C-BE4" .../>
    <node id="E8-N-3" idrefs="Cl-BE1" .../>
    <node id="E8-N-4" idrefs="Cl-BE1" .../>
    <node id="E8-N-5" idrefs="Br-BE1" .../>
    <node id="E8-N-6" idrefs="H-BE1" .../>

    <bond id="E8-B-1" idrefs="E8-N-1 E8-N-2" order="D">
        <stereo idrefs="E8-N-5 E8-N-4 E8-N-3 E8-N-6" sense="+">...</stereo>
    </bond>
    <bond id="E8-B-2" idrefs="E8-N-1 E8-N-3" order="S"/>
    <bond id="E8-B-3" idrefs="E8-N-2 E8-N-4" order="S"/>
    <bond id="E8-B-4" idrefs="E8-N-1 E8-N-5" order="S"/>
    <bond id="E8-B-5" idrefs="E8-N-2 E8-N-6" order="S"/>

</structure>
...
```