US 20180004716A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2018/0004716 A1**

Hund et al. (43) **Pub. Date: Jan. 4, 2018**

(54) **METHOD FOR CONVERTING A BINARY DATA STREAM**

(71) Applicant: **SIEMENS AKTIENGESELLSCHAFT**, München (DE)

(72) Inventors: **Johannes Hund**, München (DE); **Daniel Peintner**, Meransen Mühlbach (IT)

(21) Appl. No.: **15/545,823**

(22) PCT Filed: **Jan. 26, 2015**

(86) PCT No.: **PCT/EP2015/051502**

§ 371 (c)(1),
(2) Date: **Jul. 24, 2017**

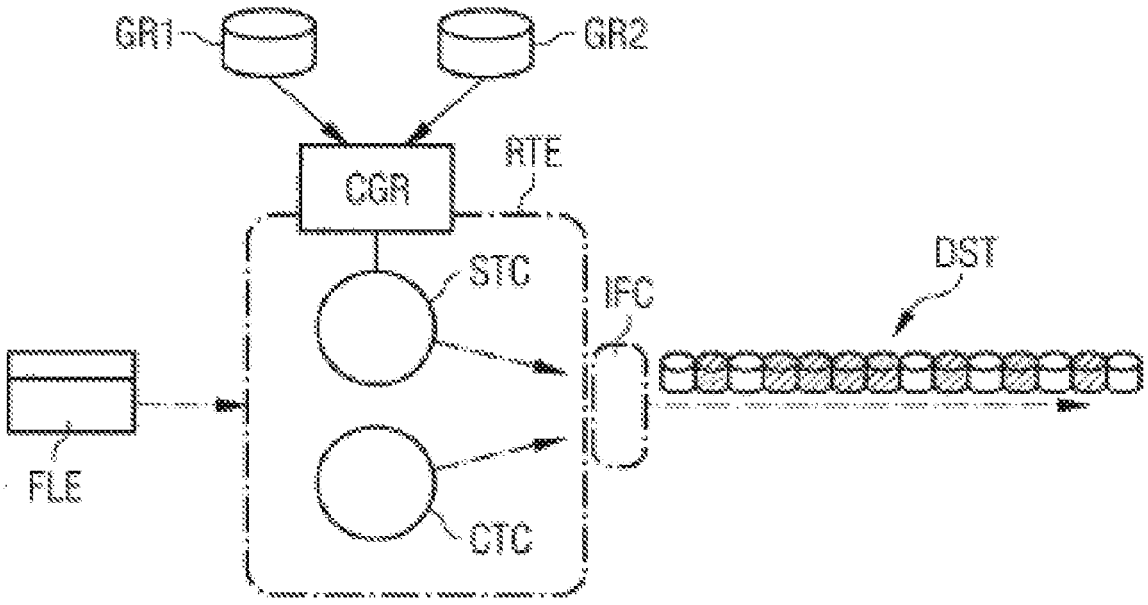
Publication Classification

(51) **Int. Cl.**
G06F 17/22 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 17/2258*** (2013.01); ***G06F 17/2247*** (2013.01); ***G06F 17/2252*** (2013.01)

(57) **ABSTRACT**

A method is provided for converting a binary data stream, (e.g., an EXI data stream). In an initialization phase of the method, a plurality of grammars, previously produced from at least one description language scheme, are read from a memory area and combined to form a combined grammar and wherein the combined grammar is supplied to a runtime environment for the purpose of converting the binary data stream. The method firstly permits substantially accelerated production of the desired grammar in comparison with a grammar produced as required from individual schemes, and secondly the memory space requirement may be kept down, because there is no need to keep a combinational variety of grammars available.



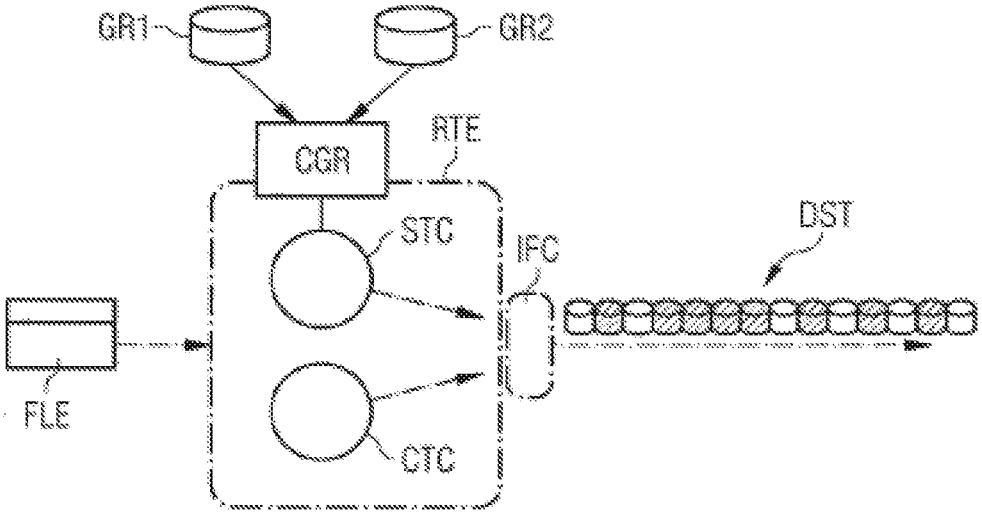


FIG. 1

METHOD FOR CONVERTING A BINARY DATA STREAM

[0001] The present patent document is a §371 nationalization of PCT Application Serial Number PCT/EP2015/051502, filed Jan. 26, 2015, designating the United States, which is hereby incorporated by reference.

TECHNICAL FIELD

[0002] The disclosure relates to a method for converting a binary data stream, e.g., an EXI data stream.

BACKGROUND

[0003] Description languages for specifying data formats and the methods needed for processing the data are known in the prior art. One known description language is "Extensible Markup Language", XML for short, which is used to describe hierarchically structured data in text form or "plain text". The description language XML is used for platform-independent interchange of data between computer systems. Owing to the textual nature of XML, the language may be read both by machines and by human beings. In addition, schemata are known that are used to describe a structure and to define data types. A schema for use for XML data is also known as an XML schema definition or XSD.

[0004] Efficient data interchange between computer systems is frequently necessary that cannot be achieved using a textual description language such as XML. Therefore, binary representations of XML have been proposed. A binary representation of XML referred to as "Efficient XML Interchange", EXI for short, is faster to process in comparison with text-based XML data and requires less transmission bandwidth for the data interchange between computer systems. Use of EXI is, moreover, not restricted solely to a binary representation of XML; EXI may be used as an interchange format for transmitting any semi-structured data.

[0005] The interchange format EXI particularly exhibits advantages when used in XML-based applications using a microcontroller having a limited supply of memory space and computation power. Devices that operate on the basis of microcontrollers, (for example, intelligent sensors on a production line or in a vehicle), may be set up for device-internal processing of data in a binary format and interchange these data with one another via appropriate communication interfaces, also, e.g., on the basis of binary data. In particular, binary data in accordance with EXI specifications allow inter-operability with XML-based systems in this case.

[0006] For the interchange format EXI too, use of schemata, (e.g., XML schema files), is advantageous. Use of schemata permits typed presentation of data being interchanged, also known as "typed data" among experts, which permits faster conversion of data into an internal representation at a receiver end. In addition, this measure permits an even more compact presentation, because knowledge already known at the receiver end, (such as, e.g., XML element names), may be transmitted no longer in a text format but rather by bilaterally known, short identifiers.

[0007] To further optimize a binary data stream that is to be transmitted in accordance with EXI, there is provision for schemata to be coded in grammars, also known as EXI grammars. In addition, a grammar also permits a coded representation or declaration of namespaces. For reasons of simplicity, the term "schemata" below covers both schemata in the sense of the preceding description and namespaces.

[0008] Coding of schemata in grammars is time-consuming and work-intensive. A further problem arises from the fact that currently established methods for converting and/or transmitting a binary EXI data stream frequently provide for multiple schemata to be used. In such cases, a combination of multiple schemata or schema files is required in order to code a grammar. In many such instances of application, the process is then unnecessarily performed repeatedly when multiple EXI data streams require like or similar combinations of schemata.

[0009] In one frequently arising case, according to which one schema from a multiplicity of schemata is subject to a change, the process of coding a grammar from the multiplicity of schemata, including unaltered schemata, needs to be repeated.

[0010] Several different combinations of schemata result in one grammar per combination each time. A technical need to keep grammars ready for different combinations of schemata on a persistent basis may currently be satisfied only by virtue of many variants of grammars being generated in advance. With an increasing number of schemata, the number of variants rises to an extreme degree on account of the great combinational diversity of possible combinations, which means that the memory space requirement for storing the grammars is accordingly high.

SUMMARY AND DESCRIPTION

[0011] The scope of the present disclosure is defined solely by the appended claims and is not affected to any degree by the statements within this description. The present embodiments may obviate one or more of the drawbacks or limitations in the related art.

[0012] The present disclosure is based on the object of providing one or more grammars corresponding to combinations of schemata associated with less involvement in terms of time and memory.

[0013] The object is achieved by a method that provides an initialization phase in a method for converting a binary data stream between a transmitter and a receiver, which initialization phase involves a plurality of grammars generated in advance being read from a memory area and said grammars being compiled to form a combined grammar.

[0014] A subsequent or simultaneously proceeding coding phase or runtime phase involves the combined grammar being used to convert binary data and/or text format data into a binary data stream or to convert a binary data stream into binary data and/or text format data. The first alternative may relate to a coding device, and the second alternative may relate to a decoding device.

[0015] Text format data are supplied to the runtime environment and text format data may be taken from the runtime environment in the form of description language schemata, for example, XML schema files, or data structures that are equivalent to XML.

[0016] Binary data are supplied to the runtime environment and binary data may be taken from the runtime environment in the form of memory representations, for example, using a format that is also known as document object model (DOM).

[0017] A memory representation is a presentation of the data described by a description language schema and transmittable by a binary data stream, which presentation is

directly interchangeable with a program and may be processed. A coding device produces a serialized binary data stream from this memory representation, e.g., for the purpose of transmission. A decoding device converts a serialized binary data stream into a memory representation. The memory representation is then used for device-internal processing of the binary data.

[0018] The term "runtime" refers to the actual conversion of the binary data stream, the conversion being realized by a coding device or EXI encoder or by a decoding device or EXI decoder, for example. The coding device and the decoding device are alternatively combined in a single device, which may be referenced by the runtime environment.

[0019] There is provision for "atomic" grammars generated from one or more schemata to be stored on a persistent basis. When a combination so requires, the grammars needed may be loaded individually and processed in combination, just as if they were already available in combination.

[0020] The method allows individual schemata or a combination of individual schemata to be converted into the appropriate grammar representation in advance or in the event of changes. From each combination of these atomic grammars, the combined grammar needed in the instance of application is generated in a runtime environment as required. This measure firstly allows substantially accelerated generation of the desired grammar, and secondly it is possible for the memory space requirement to be kept down, because there is no need to keep a combinational variety of grammars available.

[0021] The object is additionally achieved by a computer program product executed in a coding device and/or decoding device that, on execution, performs the method.

[0022] The object is additionally achieved by a coding device and by a decoding device.

BRIEF DESCRIPTION OF THE DRAWINGS

[0023] Further exemplary embodiments and advantages of the disclosure are explained in more detail below with reference to the drawing.

[0024] FIG. **1** depicts an example of a schematic depiction of functional units of a coding device in which text format data of a description language that are supplied to the coding device are converted into a binary data stream.

DETAILED DESCRIPTION

[0025] As depicted in FIG. **1**, the actual coding is effected in a unit referred to as a runtime environment RTE to which input data are supplied in file or text form. By way of example, input data in the form of an XML document FLE are supplied to the runtime environment RTE. Alternatively, memory representations or data structures equivalent to XML are supplied as input data.

[0026] Using a structure coding STC and a content coding CTC, a binary data stream DST is generated in the runtime environment RTE, which is transmitted to a receiver (not depicted) via an interface IFC of the coding device.

[0027] A plurality of atomic grammars GR**1**, GR**2** generated from at least one description language schema in advance are read from a memory area (not depicted) and compiled to form a combined grammar CGR in an initialization phase. The combined grammar CGR is supplied to the runtime environment RTE via a nonstandardized interface, e.g., an EXI grammar interface. Depending on the combinational requirement, the grammars GR**1**, GR**2** are loaded individually and, after being compiled, are processed as a combined grammar CGR, just as if a grammar were already available in combined form.

[0028] Structure coding STC and content coding CTC are used to process the input data FLE supplied to the runtime environment RTE to produce the binary data stream DST from the combined grammar CGR.

[0029] As is known to a person skilled in the art, a grammar is a coding and decoding instruction for an XML document, a data structure equivalent to XML, or a memory representation of an XML document, for example, as a document object model, DOM, of a coding device. A decoding device uses the grammar as an instruction to produce an XML document, a data structure equivalent to XML, or a memory representation of an XML document from a received data stream.

[0030] The functional units depicted in FIG. **1** may be present both in a coding device at a transmitter end and in a decoding device at a receiver end. At the receiver end, the binary data stream DST then forms the input data that are supplied via the interface IFC to the units for structure coding STC and content coding CTC to produce the structure and the content. The XML document, the memory representation, or the data structure equivalent to XML are then the output data from the receiver.

[0031] In accordance with one embodiment, the initialization phase described for the transmitter and linked to compiling the combined grammar may also be dispensed with at the receiver end, however. This is made possible in this embodiment by virtue of the grammar CGR required for the function of the transmitter and receiver being generated by the transmitter and then transmitted to the receiver in an initialization phase before the actual runtime of the data interchange. All of the information that the grammar CGR contains may be contained in a first, (e.g., characterized), data stream section that is transmitted from the transmitter to the receiver.

[0032] The text below explains the concept of a combined grammar based on an illustrative example. In this case, it is assumed that conversion of a binary data stream requires an XML schema having the file name "basket.xsd", where the file suffix "xsd" denotes an XML schema definition. In the basket schema, elements of further schemata, for example the schemata "apple.xsd", "pear.xsd", and "citrus.xsd", are used on a case-by-case basis.

[0033] Conversion of a binary data stream using previously known methods required the following grammars Gk, Gka, Gkb, . . . Gkabc to be generated and kept available using a grammar (. . .) call from a respective combination of schemata:

[0034] Gk=grammar (basket.xsd)

[0035] Gka=grammar (basket.xsd+apple.xsd)

[0036] Gkb=grammar (basket.xsd+pear.xsd)

[0037] Gkc=grammar (basket.xsd+citrus.xsd)

[0038] Gkab=grammar (basket.xsd+apple.xsd+pear.xsd)

[0039] Gkac=grammar (basket.xsd+apple.xsd+citrus.xsd)

[0040] Gkbc=grammar (basket.xsd+pear.xsd+citrus.xsd)

[0041] Gkabc=grammar (basket.xsd+apple.xsd+pear.xsd+citrus.xsd)

[0042] The grammars Gk, Gka, Gkb, . . . Gkabc denoted above either had to be produced from the respective sche-

mata by an encoder or a decoder ad hoc, that is to say immediately before their respective use, or had to be generated in advance and buffer-stored in a memory area, particularly a cache memory, until they were used. While the first is disadvantageous on account of time-consuming and resource-involving ad-hoc generation, the second variant has the disadvantage that a high memory requirement is necessary for keeping all the grammars that may be required available. In addition, a change in one of the schemata requires all the grammars affected to be produced afresh.

[0043] In addition, entry of a further schema, for example "date.xsd", requires a grammar to be generated for every possible combination with this schema, and in the above example there would be entry of 18 additional combinations of possible schema combinations. This may entail a change in the schema basket.xsd, which means that new grammars have to be generated for all combinations of schemata, in that case 26 in the example above.

[0044] In certain examples, there is, by contrast, provision for a plurality of grammars already generated from a schema in advance to be compiled to form a combined grammar. Alternatively, these grammars already generated in advance may also have been generated from a plurality of schemata.

[0045] To keep available the grammars G'k, G'a, G'b, G'c already generated from a schema in advance, subsequently also referred to as an atomic grammar or atomic combinable grammar, an atomic grammar (. . .) call is used as follows:

[0046] G'k=atomic_grammar (basket.xsd)

[0047] G'a=atomic_grammar (apple.xsd)

[0048] G'b=atomic_grammar (pear.xsd)

[0049] G'c=atomic_grammar (citrus.xsd)

[0050] These atomic grammars permit substantially more efficient combination in a runtime environment to form a combined grammar than previously known generation of a grammar from a combination of schemata, which means that multiple atomic grammars may be compiled to form a combined grammar substantially more quickly in the runtime environment.

[0051] Even when a further schema is added as explained, it is merely necessary for every directly changed schema to be produced afresh as explained below using the example of an entering date schema:

[0052] G'd=atomic_grammar (date.xsd)

[0053] Only the atomic grammar G'k associated with the changed basket schema needs to be produced afresh in the course of this entry:

[0054] G'k=atomic_grammar (basket.xsd)

[0055] If the grammar Gkbc, which, according to conventional methods, would either have to be in the cache or would have to be generated, is now needed in the application, then it may be combined from the atomic grammars as a combined grammar, e.g., using a combine (. . .) call:

[0056] Gkbc=combine(G'k +G'b +G'c)

[0057] The combined grammar is moreover identical at runtime to the conventionally generated grammar, e.g., the grammar generated and kept available from the combination of the basket, pear, and citrus schemata using a grammar (. . .) call:

[0058] Gkbc=grammar(basket.xsd+pear.xsd+citrus.xsd)

[0059] For an implementation of the runtime environment in accordance with the rules of the EXI standard, the resultant combined grammar corresponds to a grammar that would also have emerged for a combination of different schemata in accordance with XML schema definition or XSD.

[0060] The prerequisite for combinability of atomic grammars that needs to be called for in accordance with a embodiment is that global elements of respective atomic grammars, e.g., those generated in advance, may be supplied to a combined list of global elements of the combined grammar.

[0061] In accordance with one embodiment, this is achieved, by way of example, by virtue of references to qualified names, which are referred to as QNames in the EXI vernacular, of elements of the individual atomic grammars being brought together in a list, that ultimately represents the combined grammar. A qualified name is a globally explicit descriptor that is compiled from the element name and a descriptor for the namespace. Specifically, a qualified name or QName may be compiled from what are known as Non-Colonized Names (NCNames), each of the NCNames apart from the last denoting a namespace. The last NCName corresponds to the local name within the namespace. The individual NCNames are compiled by dots (.) to form a QName.

[0062] In accordance with one embodiment, the combinability is supported by virtue of global elements of the combined list of global elements of the combined grammar referring to global elements of respective grammars generated in advance by virtue of indirectness. This means, by way of example, that a combined list of global elements with the length (e.g., length G**1**+length G**2**) is kept that indirectly refers to the existing lists. A list of references to the respective atomic grammars is thus created. This list may be set up in the initialization phase when different atomic grammars are combined.

[0063] In accordance with one embodiment, the combinability is supported by virtue of the initialization phase involving elements of respective fragmentary grammars generated in advance being supplied to a combined list of global elements of the combined grammar, wherein a match for a qualified name or "QName" in different elements prompts the matching qualified name to be created precisely once in the combined list of global elements, with name conflicts being resolved in accordance with a schema-informed grammar. This embodiment relates to elements of a "FragmentContent Grammar" in accordance with the EXI standard. The approach in this regard is inherently analogous to the global elements explained above. An exception thereto is formed only by definitions having identical QNames, which are compiled from the element name and a descriptor for the namespace. The aim of converting numerous elements having the same name into a single, explicit element in the combined grammar is achieved by resolving name conflicts analogously to the approach in a schema-informed grammar or "Schema-informed Element Fragment Grammar".

[0064] In accordance with one embodiment, the combinability is supported by virtue of the initialization phase involving type attributes of respective grammars generated in advance being supplied to a combined list of type attributes that is set up as required from a combination of grammars generated in advance and grammars containing type attributes. When a type attribute "xsi:type cast" appears, an EXI codec jumps to the referenced type "Grammar", if the latter is available. In the case of combined

grammars, the list is set up afresh when first required by virtue of it setting up a combination of all type grammars (e.g., G1+G2+ . . .).

[0065] In accordance with one embodiment, the combinability is supported by virtue of the initialization phase involving at least one global element from the combined grammar with an applicable local name and/or an applicable namespace descriptor being used for wildcard expressions or wildcards. When wildcard expressions or wildcards appear, for example wildcards defined in EXI, (such as, e.g., AT(*), AT(uri, *), SE(*), SE(uri, *)), a grammar may use a list of all possible global elements and/or attributes as a possible comparison. In this case, an attempt is made to find a global element and/or attribute that matches the local name and a descriptor or Uniform Resource Identifier (URI) of the namespace in the list. In the case of multiple combined grammars, the combined list is accordingly searched.

[0066] In accordance with one embodiment, the combinability is supported by virtue of the initialization phase involving an extended substitution group, in which elements from different grammars generated in advance are assigned to one and the same substitution group, being formed for elements from substitution groups of different grammars generated in advance. In XML and EXI, an element may be substituted with all elements that belong to the same substitution group or "substitutionGroup". In the case of combined grammars, the substitution group or the list associated with the substitution group may need to be extended if elements from different grammars belong to the same substitutionGroup.

[0067] The disclosure relates, in summary, to a method for converting a binary data stream, (e.g., an EXI data stream), wherein an initialization phase involves a plurality of grammars generated from at least one description language schema in advance being read from a memory area and compiled to form a combined grammar and wherein the combined grammar is supplied to a runtime environment for the purpose of converting the binary data stream.

[0068] The disclosure firstly permits substantially accelerated generation of the desired grammar in comparison with a grammar generated as required from individual schemata, and secondly the memory space requirement may be kept down, because there is no need to keep a combinational variety of grammars available.

[0069] The disclosure provides increased flexibility by allowing new, possibly unforeseen, combinations without having to generate a new grammar for these combinations. Furthermore, faster updating is possible for changes in individual schemata, because only the atomic grammar affected rather than all combinations needs to be generated afresh.

[0070] In addition, the disclosure permits reduced management of metadata, because an atomic grammar may relate to just one schema rather than to a combination of schemata.

[0071] It is to be understood that the elements and features recited in the appended claims may be combined in different ways to produce new claims that likewise fall within the scope of the present disclosure. Thus, whereas the dependent claims appended below depend from only a single independent or dependent claim, it is to be understood that these dependent claims may, alternatively, be made to depend in the alternative from any preceding or following claim, whether independent or dependent, and that such new combinations are to be understood as forming a part of the present specification.

[0072] While the present disclosure has been described above by reference to various embodiments, it may be understood that many changes and modifications may be made to the described embodiments. It is therefore intended that the foregoing description be regarded as illustrative rather than limiting, and that it be understood that all equivalents and/or combinations of embodiments are intended to be included in this description.

1. A method for converting a binary data stream, the method comprising:

forming a combined grammar in an initialization phase from a plurality of grammars, the forming including reading the plurality of grammars from a memory area and compiling the plurality of grammars into the combined grammar, wherein the plurality of grammars is generated in advance of the forming from at least one description language schema; and

supplying the combined grammar to a runtime environment to convert the binary data stream.

2. The method of claim **1**, wherein the conversion of the binary data stream comprises supplying binary data, text format data, or both the binary data and the text format data of a description language to the runtime environment.

3. The method of claim **1**, wherein the binary data stream is converted into binary data, text format data, or both the binary data and the text format data of a description language.

4. The method of claim **1**, wherein the initialization phase involves global elements of respective grammars generated in advance being supplied to a combined list of global elements of the combined grammar.

5. The method of claim **4**, wherein the global elements of the combined list refer to global elements of respective grammars generated in advance by virtue of indirectness.

6. The method of claim **1**, wherein the initialization phase involves elements of respective fragmentary grammars generated in advance being supplied to a combined list of global elements of the combined grammar,

wherein a match for a qualified name in different elements prompts the matching qualified name to be created precisely once in the combined list of global elements, and

wherein name conflicts are resolved in accordance with a schema-informed grammar.

7. The method of claim **1**, wherein the initialization phase involves type attributes of respective grammars generated in advance being supplied to a combined list of type attributes set up from a combination of grammars generated in advance and grammars containing type attributes.

8. The method of claim **7**, wherein the initialization phase involves at least one global element from the combined grammar with an applicable local name and/or an applicable namespace descriptor being used for wildcard expressions.

9. The method of claim **1**, wherein the initialization phase involves an extended substitution group, in which elements from different grammars generated in advance are assigned to one and the same substitution group, being formed for elements from substitution groups of different grammars generated in advance.

10. A computer program product having program code configured to, when the computer program product is

executed on a coding device or decoding device, cause the coding device or decoding device to at least perform:

form a combined grammar in an initialization phase from a plurality of grammars, the forming including reading the plurality of grammars from a memory area and compiling the plurality of grammars into the combined grammar, wherein the plurality of grammars is generated in advance of the forming from at least one description language schema; and

supply the combined grammar to a runtime environment to convert the binary data stream.

11. A coding device for converting a binary data stream, the coding device comprising:

an initialization unit configured to read a plurality of grammars generated from at least one description language schema in advance from a memory area and compile the plurality of grammars to form a combined grammar; and

a runtime unit configured to convert binary data, text format data, or both the binary data and the text format data of a description language, which are supplied to the coding device, into a binary data stream by applying the combined grammar.

12. A decoding device for receiving a binary data stream, comprising

an initialization unit configured to read a plurality of grammars generated from at least one description language schema in advance from a memory area and compile the plurality of grammars to form a combined grammar; and

a runtime unit configured to convert a binary data stream supplied to the decoding device into binary data, text format data, or both the binary data and the text format data of a description language by applying the combined grammar.

13. The method of claim **3**, wherein the initialization phase involves global elements of respective grammars generated in advance being supplied to a combined list of global elements of the combined grammar.

14. The method of claim **13**, wherein the global elements of the combined list refer to global elements of respective grammars generated in advance by virtue of indirectness.

15. The method of claim **2**, wherein the initialization phase involves global elements of respective grammars generated in advance being supplied to a combined list of global elements of the combined grammar.

16. The method of claim **15**, wherein the global elements of the combined list refer to global elements of respective grammars generated in advance by virtue of indirectness.

17. The method of claim **6**, wherein the initialization phase involves at least one global element from the combined grammar with an applicable local name and/or an applicable namespace descriptor being used for wildcard expressions.

18. The method of claim **5**, wherein the initialization phase involves at least one global element from the combined grammar with an applicable local name and/or an applicable namespace descriptor being used for wildcard expressions.

19. The method of claim **4**, wherein the initialization phase involves at least one global element from the combined grammar with an applicable local name and/or an applicable namespace descriptor being used for wildcard expressions.

* * * * *