Open Geospatial Consortium Inc.

Date: 2008-11-5

Reference number of this document: OGC 08-132

Version: 0.3.0

Category: OGC Discussion Paper

Editors: Thomas Everding, Johannes Echterhoff

Event Pattern Markup Language (EML)

Warning

This document is not an OGC Standard. This document is an OGC Discussion Paper and is therefore not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, an OGC Discussion Paper should not be referenced as required or mandatory technology in procurements.

Document type:	OpenGIS® Discussion Paper
Document subtype:	
Document stage:	Approved as Discussion Paper
Document language:	English

Contents

Page

Figures

Page

Tables

Page

i. Preface

This specification introduces a markup language with which patterns for the detection of interesting events in an event cloud can be specified.

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

ii. Forward

The patterns and functionality described in this document can be applied to create filters that operate on event streams / clouds. Therefore, it can be regarded as an enhancement of the OGC Filter Encoding Standard (FES).

The discussion in this document is related to all services that store or process data, because there are lots of events generated or received by these services (e.g. whenever a new objects is created / inserted at the service). Applying the techniques described in this specification to those events enables on-the-fly filtering of data and derivation of higher-level information.

This work was supported by the OSIRIS project. OSIRIS is an Integrated Project (contract number 0033475) co-funded by the Information Society and Media DG of the European Commission within the RTD activities of the Thematic Priority Information Society Technologies.

This document includes three annexes; Annexes A and B are normative, and Annex C is informative.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

iii. Document terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 06-121r3], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this specification.

iv. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Johannes Echterhoff	IFGI
Thomas Everding	IFGI

v. Revision history

Date	Release	Editor	Primary clauses modified	Description
2008-08-19	0.0.1	Johannes Echterhoff		Initial draft
2008-08-26	0.0.1	Thomas Everding	throughout	Initial composition
2008-9-30	0.3.0	Carl Reed	Various	Get ready for posting as Discussion Paper

vi. Changes to the OGC Abstract Specification

It is not clear whether the notion of event stream / cloud processing is already covered by the OpenGIS® Abstract Specification, considering also that event pattern detection usually involves an asynchronous, push-based mechanism to notify clients of pattern matches.

vii. Future work

The specification contained in this paper is intended to fuel discussion about the following topics that should be considered for future OGC developments:

- Definition and application of patterns to detect events of interest in event streams / clouds.
- Integration of event pattern detection functionality in all kinds of data storage / processing services.

- Development of a general pub-sub mechanism that allows content based filtering of event (streams / clouds).
- General improvement of EML patterns.
- Harmonization with ISO Specifications (e.g. ISO 19136).
- Enhancement of compliance tests.
- Merging of multiple select function results into single event objects.
- Enhancement of the NotifyOnSelect usage (e.g. definition of service operation calls).

Introduction

The Event Pattern Markup Language (EML) allows one to describe event patterns for event (stream) processing and analysis. It can be used to build multi stage filters for incoming events but also to derive higher information through combining and correlating multiple events. It can be applied on single events but is focused on handling of continuous event streams.

EML consists of an extensible set of methods and functions. Therefore a capabilities document describes the features that are supported by an application using EML.

Additionally a data type for EML event objects is specified to enable applications to send and receive event objects between different applications without loosing information details such as the origin of derived data values.

Event Pattern Markup Language (EML)

1 Scope

This OGC® Discussion Paper specifies a markup language for the description of event patterns which can be used to filter incoming event objects but also to derive higher level information.

The specification defines four types of event patterns. Simple patterns can be used to filter a single event stream whereas complex patterns allow one to combine data from multiple event streams for the filtering process. Timer patterns can be used to define time-dependent filters. Repetitive patterns are designed to react on repetitions of patterns in event streams. EML enables the generation and storage of causality relations between event objects.

This specification defines views that can be attached to simple, complex and timer patterns. The size of these specified views can be altered by a maximum number of contained event objects, via time constraints or both. The views can operate in sliding or in batch mode. Additional views can be defined by the user.

Eight select functions are specified in this document which can be utilized to define what information can be used for further processing. There are three basic types: Type 1 derives information from a set of event objects, type 2 selects information contained in the available event object and type 3 delivers information independent from the content of the available event objects. Additional select functions can be defined by the user.

This document also specifies the EML capabilities used to define which of the introduced features are supported by an application. Furthermore a data type for representation of event objects is defined.

This specification does not rely upon a specific pattern matcher implementation.

This OGC® document is applicable in scenarios where one or multiple incoming event streams are correlated and filtered.

2 Compliance

Compliance with this specification shall be checked using all the relevant tests specified in Annex A (normative).

3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*

ISO DIS 19136:2006, *Geographic Information – Geography Markup Language*

OpenGIS® Implementation Specification *Filter Encoding OGC Document 04-095*

OpenGIS® Implementation Specification *OGC Web Services Common Specification OGC Document 06-121r3*

OpenGIS® Reccomendation Paper *OpenGIS® Geography Markup Language (GML) Implementation Specification OGC Document 03-105r1*

In addition to this document, this specification includes several normative XML Schema Document files as specified in Annex B.

4 Terms and definitions

For the purposes of this specification, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] shall apply. In addition, the following terms and definitions apply.

4.1
filter
defines conditions that are checked against incoming data

4.2
event
action that occurs at an instant or over an interval of time [ISO 19136]

NOTE There is another definition given in ISO 19108, where an event is an action that occurs in an instant. An instant is a zero-dimensional geometric primitive representing position in time. More specifically, an instant is an interval whose duration is less than the resolution of the time scale. For this specification, we will follow the definition provided in ISO 19136 (where the event definition that we are using is given in the description of the gml:AbstractTimeSlice element).

4.3
higher-level event
complex event
event that contains information derived by processing the information of one or more other events

NOTE Higher-level events represent specialized types of events which may contain the source events from which they have been derived in a causality vector. The information contained in the causality vector allows decision makers to determine why a higher-level event was created (with knowledge of the processing instructions).

4.4
event object
representation of an event for filtering and analysis purposes

4.5
causal vector
attribute of an event object storing the causal history of an event

4.6
event pattern
rule for the filtering and analysis of events

4.7
event cloud
partially ordered set of all available events

4.8
event stream
continuous sequence of events ordered by time

NOTE An event cloud can contain multiple event streams.

4.9
pattern matching
operation in Complex Event Processing (CEP) comparing events with event patterns

4.10
pattern matcher
program performing pattern matching

4.11
match
pattern match
event object that satisfies an event pattern

5 Conventions

5.1 Abbreviated terms

Most of the abbreviated terms listed in Subclause 5.1 of the OWS Common Implementation Specification [OGC 06-121r3] apply to this document, plus the following abbreviated terms.

CEP	Complex Event Processing

EML	Event Pattern Markup Language
ESP	Event Stream Processing
FIFO	First In, First Out
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extensible Markup Language

5.2 UML notation

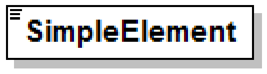
Some diagrams that appear in this specification are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 06-121r3].

5.3 XMLSpy notation

Most diagrams that appear in this specification are presented using an XML schema notation defined by the XMLSpy[1] product and described in this subclause. XML schema diagrams are for informative use only though they shall reflect the accompanied UML and schema perfectly.

5.3.1 Element

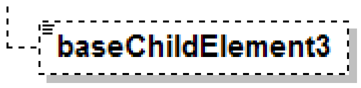
A named rectangle representing the most basic part of the XML Schema notation. Each represents an XML "Element" token. Each Element symbol can be elaborated with extra information as shown in the examples below.



This is a mandatory simple element. Note the upper left corner of the rectangle indicates that data is contained in this element.

5.3.2 Optional Element

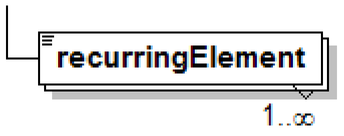
Optional (non mandatory) elements are specified with dashed lines used to frame the rectangle.



[1] XML Spy: http://www.altova.com

5.3.3 Recurring Element

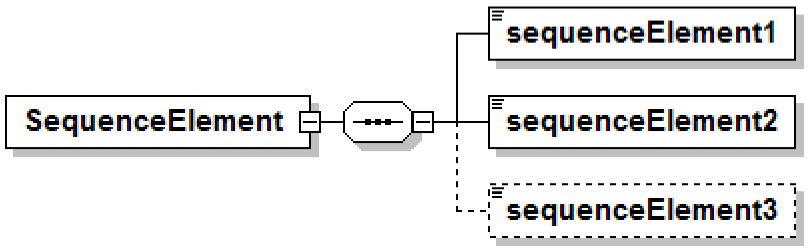
This element (and its child elements if it has any) can occur multiple times.



This example shows a recurring element that must occur at least once but can occur an unlimited amount of times. The upper bound here is shown with the infinity symbol.

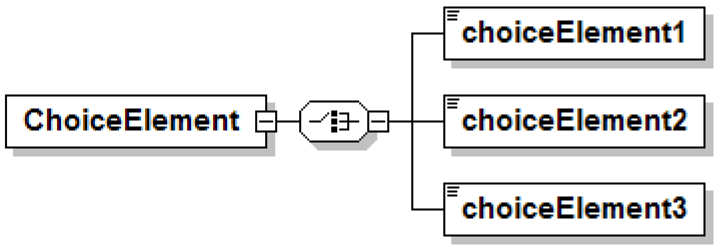
5.3.4 Sequence Connector

The connection box, called a sequence indicator, indicates that the "SequenceElement" data is made up of three elements. In this example, the first two elements are mandatory and the third element is optional



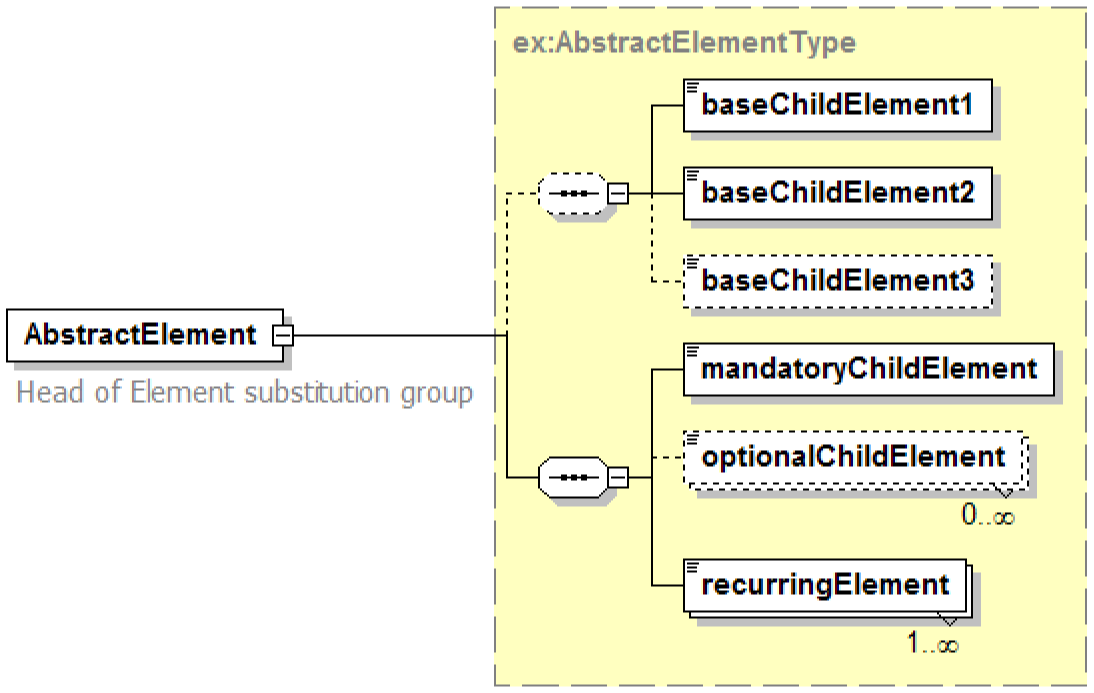
5.3.5 Choice Connector

The connection box here is a "choice" indicator, indicating that there is always going to be exactly one of the child elements listed on the right.



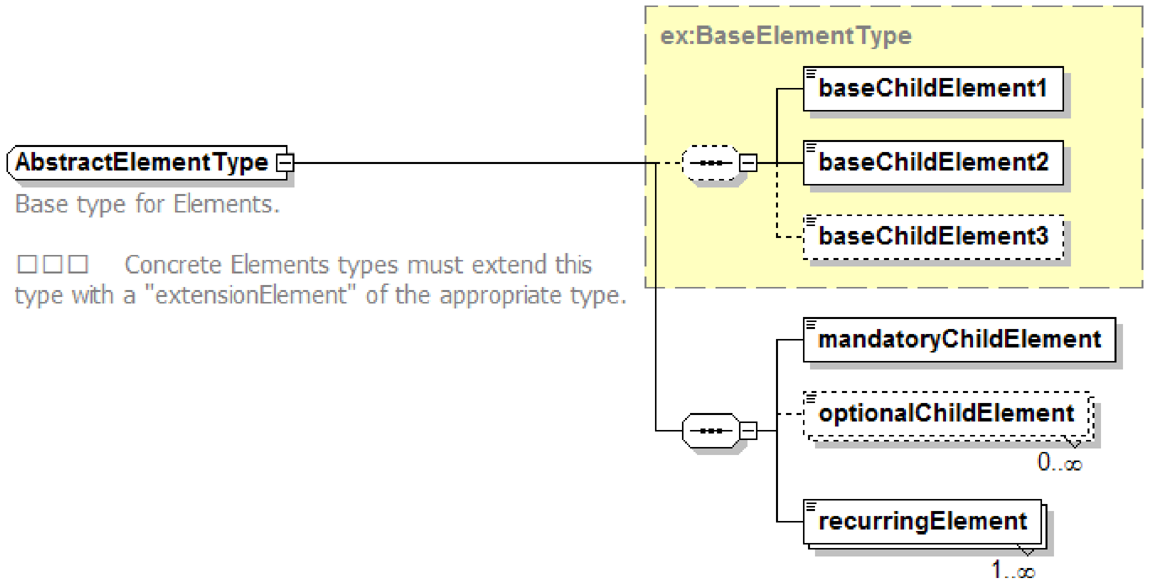
5.3.6 Definition with Complex Type

This diagram illustrates the use of a complex type (i.e., "ex:AbstractElementType") for defining an XML element (e.g., "AbstractElement").



5.3.7 Complex Type

This diagram illustrates the definition of a complex type (i.e., “AbstractElementType”), extending another complex type (i.e.,“ex:BaseElementType”) with three additional elements. Complex types can be reused to specify that different elements are of the same type.



6 Event Pattern Markup Language overview

Event processing is a technique for creating, deleting, reading, editing of and reacting to events. A special form is Complex Event Processing (CEP). It uses relations between multiple events to derive higher level information. The derived data can be represented by complex event objects for further processing. The main operation of CEP is pattern matching. It is executed by a pattern matcher who searches the cloud of available event objects for events which attributes correspond to rules defined by event patterns. Event Stream Processing (ESP) can be seen as a sub-form of CEP. Instead of an event cloud it searches continuous streams of events for matches. Such an event stream can be seen as an ordered (e.g. by time) part of an event cloud. Event clouds on the other hand may contain several event streams (see Figure 1). When processing event streams, views (also windows) are often used to reduce the possible infinite amount of data and to speed up the processing (see Figure 2).

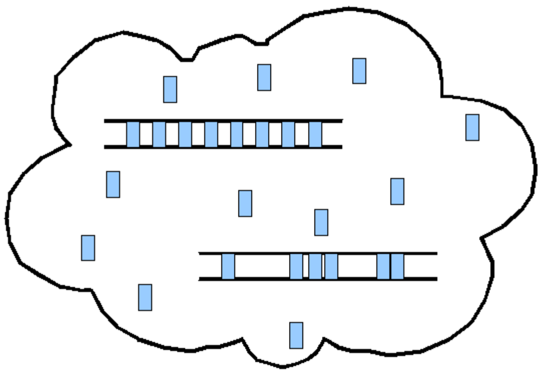


Figure 1: Event cloud containing events and event streams

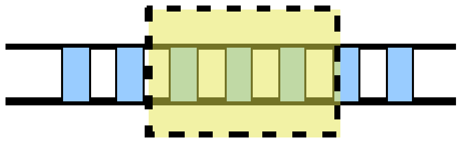


Figure 2: Event stream with a view

Every event is represented by an event object. These objects contain the necessary attributes to describe the activity that has taken place. They include time stamps as well as other data. The causal vector of an event contains all those events which led to the event at hand. It may contain events from which new data was derived or events which were aggregated to higher level events (see Figure 3). The causal vector can be used to trace back the origin of an event. These higher level events are also called complex events because they are generated by Complex Event Processing (see [1]).

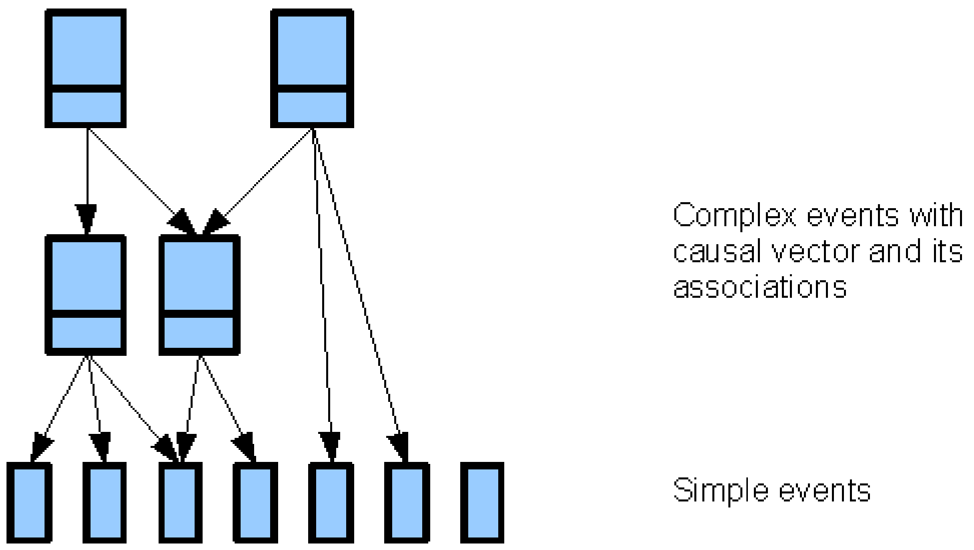


Figure 3: Event hierarchy

The Event Pattern Markup Language (EML) allows description of event patterns to perform Event Stream Processing. These patterns are single filter rules which are matched against incoming events by a pattern matcher. There are four types of patterns in EML:

1. simple patterns for filtering of a single input stream,
2. complex patterns to correlate events with each other,
3. timer patterns to generate time events and
4. repetitive patterns to count events.

Every pattern has an ID for referencing it in other patterns and a select function which defines what is used as the result of the pattern matching process and what to do with the results. Guards can be used in some patterns to restrict events which are relevant for the given pattern. In addition, views may be applied to further define constraints which of the events arriving at the service are relevant for the pattern, thereby forming a subset of events on which the pattern operates. This can be compared to a SQL statement like "`SELECT <select function> FROM <view> WHERE <guard>`". Figure 4 gives an overview of the conceptual model of EML.

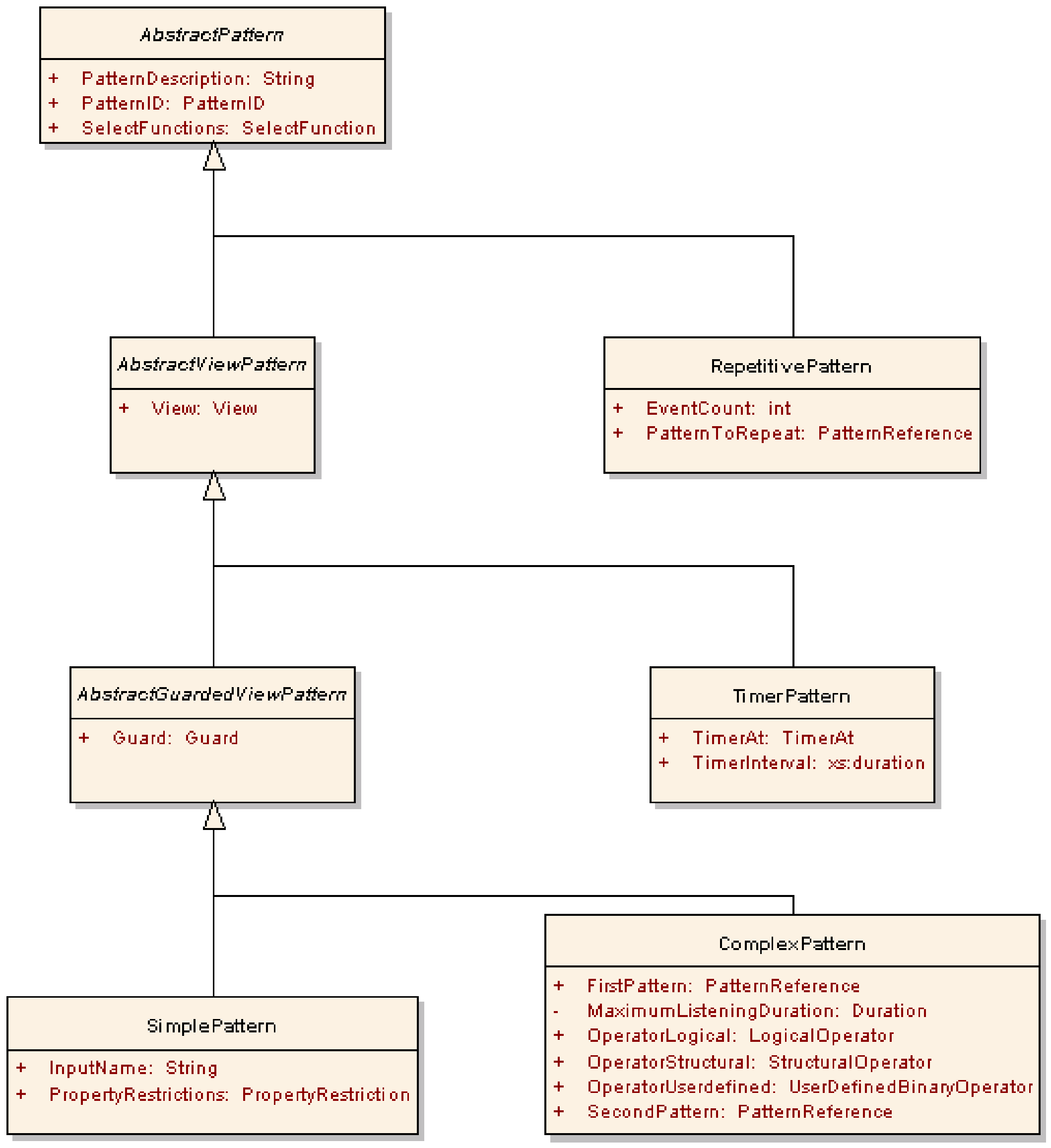

AbstractPattern
+ PatternDescription: String
+ PatternID: PatternID
+ SelectFunctions: SelectFunction
AbstractViewPattern
+ View: View
RepetitivePattern
+ EventCount: int
+ PatternToRepeat: PatternReference
AbstractGuardedViewPattern
+ Guard: Guard
TimerPattern
+ TimerAt: TimerAt
+ TimerInterval: xs:duration
SimplePattern
+ InputName: String
+ PropertyRestrictions: PropertyRestriction
ComplexPattern
+ FirstPattern: PatternReference
- MaximumListeningDuration: Duration
+ OperatorLogical: LogicalOperator
+ OperatorStructural: StructuralOperator
+ OperatorUserdefined: UserDefinedBinaryOperator
+ SecondPattern: PatternReference

Figure 4: Conceptual Model of EML

Furthermore the EML capabilities allow defining what features of EML are supported by an application or service. These capabilities are much like the capabilities of the filter encoding (OGC 04-095). One can use them to validate if an EML event pattern can be used in a specific EML application. Also a data type is proposed for the representation of (complex) events. It enables one to store an event object with the time stamps, the causal vector and all of its attributes in a uniform way.

The following paragraphs describe the EML event pattern, the EML capabilities and the EML event data type in more detail.

7 EML patterns

7.1 Simple patterns

Simple patterns form the first stage of stream filtering. Incoming data tupels are stored in an event object and published to the pattern matcher.

The inputName specifies which data source (e.g. a sensor) is used for the pattern via an ID. It shall be unique for the application. The wildcard '*' can be used instead for all known sources. Every service specification using EML has to define how to map these IDs to data sources. The input name is also used in subsequent processing stages for referencing event objects and accessing their attributes.

Specifications using EML shall also define how incoming collections of data are handled. One could split them into several simple event objects and then pass them to the pattern matcher (for instance useful when receiving O&M observation collections) or use the whole collection as a single event object (e.g. filtering of imagery data).

The filtering in a simple pattern can be described either by using property restriction, a guard or both. A property restriction contains the name and a value of a property (see Figure 5). Only event objects with the correct value for the property are a match of the pattern and therefore used for further filtering or as result. There can be multiple property restrictions for every simple pattern. In this case every restriction must be satisfied for an event object to be a pattern match.

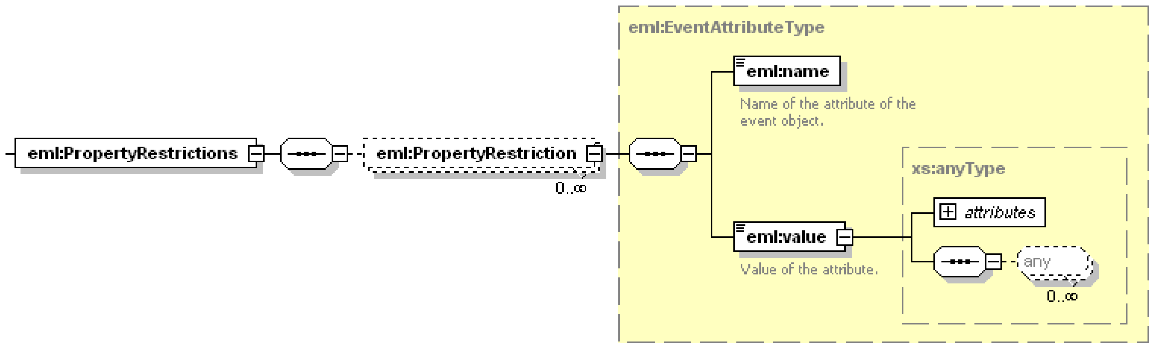


Figure 5: Property restrictions for simple patterns in EML

7.2 Guards

Guards can be compared to the `WHERE` clause from SQL. They are used to define more complex restrictions. EML uses the OGC filter encoding (OGC 04-095) to define these guards. Like the property restrictions the received event objects need to pass the guard for further processing.

7.3 Views

Views are used to restrict the processing of event objects by a pattern to a subset of the set of all event objects available to that pattern. They can be used with simple, complex and timer patterns. Table 1 gives an overview of the predefined views in EML.

Table 1: Predefined views in EML

View name	Parameter	Is parameter mandatory
AllView	-	-
LengthView	EventCount isBatch	Yes No
TimeView	Duration isBatch	Yes No
TimeLengthView	EventCount Duration isBatch	Yes Yes No

The AllView is used if no other view is specified (so this is the implicit default view) and does not restrict the set of event objects. The LengthView restricts the number of event objects available to a pattern to a given maximum (specified via the mandatory parameter EventCount), the TimeView contains only those event objects which have been received (or generated) in no more than a given duration (specified via the mandatory Duration parameter). The TimeLengthView is a combination of the two previous views. Therefore, all event objects must meet the requirements of both views in order to be part of the event set that the pattern can operate on. The last three views can either be used in sliding or batch mode.

If the sliding mode is used, the queries (see select functions defined in the next paragraph) are executed whenever a new event object is inserted into the view. If a view defines a certain size for event objects of a pattern (either by time or length constraints) and if that size is exceeded, the oldest event object is released before the insertion of the new object. If the arrival of an event object dates back longer than allowed for a view, the object is released and the query is executed on the retained objects.

The batch mode can be used by setting the optional parameter isBatch to true. In this mode the pattern collects event objects until the condition (or one of the conditions) defined by the view is satisfied. This can be that the first insertion dates back long enough or the given number of event objects is reached. In this case the query for the view is executed and all contained objects are released.

Figure 6 shows an example of a length view in sliding mode (length = 3). At the beginning it contains no event objects (a). As some arrive they are added to the view (b) until it is full. If new event arrive they are added to the view and the oldest event is released (c). Every time the content of the views changes the select function (see 7.4) is performed.

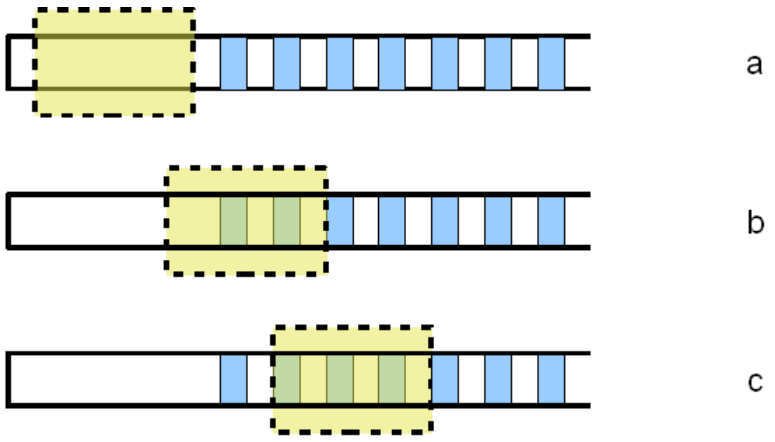


Figure 6: Sliding mode view example

Figure 7 shows an example of the same length view in batch mode. At the beginning the view is also empty (a). It starts collecting event objects until it desired size is reached (b). If further event objects arrive the select function is performed, the contained event objects are released and the view starts collecting again (c).

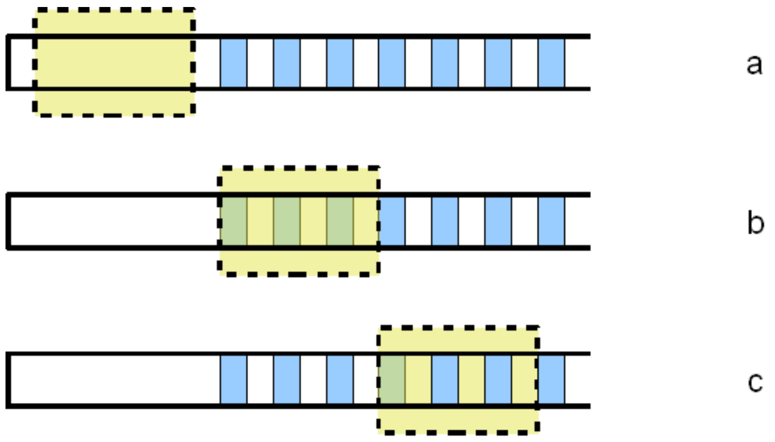


Figure 7: Batch mode view example

In addition to the predefined views it is possible to use custom views. These user defined views can have multiple parameters as key value pairs (see Figure 8) and – if used by implementations – need to be specified separately.

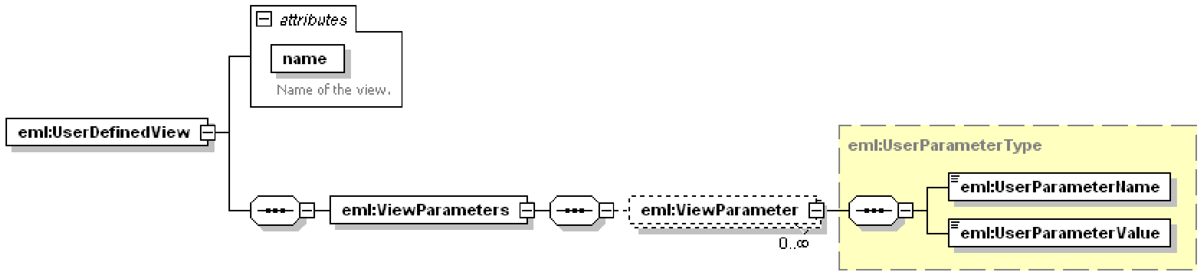


Figure 8: Structure of the UserDefinedView element which can be used in EML

7.4 Select Functions

Select functions are used to specify what parts of the received and matched event objects should be selected and how the results are further processed. It is possible to define multiple select functions for every event pattern. In the following, at first the data selection is described while the description of the possibilities for further processing will be described afterwards.

7.4.1 Data selection

There are eight predefined select functions of three major types available in EML, as illustrated in the following table.

Table 2: Predefined select functions in EML

Function	Parameter	Is Parameter mandatory
Type 1		
SelectSum	propertyName	Yes
SelectMax	propertyName	Yes
SelectMin	propertyName	Yes
SelectAvg	propertyName	Yes
Type 2		
SelectEvent	eventName	Yes
SelectProperty	propertyName	Yes
Type 3		
SelectCount	-	-
NotifyOnSelect	Message	Yes

Functions of the first type derive data from all (defined by the view that is applied to the pattern) available event objects. The functions of this type are SelectSum to select the sum of a property, SelectMax and SelectMin to select the maximum or the minimum of a property as well as SelectAvg to select the average of a property. For all of them a reference to a property must be given via the propertyName parameter. It is recommended to create the causality (see clause 7.4.2) when using these select functions in order to produce valuable results.

The second type of select functions does not derive data but simply selects what already exists. Using SelectEvent one can select a complete event object. However, this selection might be ambiguous if multiple events are available. The only parameter for this function is eventName to specify which type of event object should be selected. This is important when using SelectEvent in complex patterns which can use multiple types of event objects. If everything should be selected, '*' can be used as wildcard. If only a single property rather than the whole event object should be selected one can use the function SelectProperty. This function works in nearly the same way as SelectEvent but needs a full property name as parameter.

The third type of select functions generates results independent of the content of the available event objects. SelectCount simply returns the number of available event objects. NotifyOnSelect returns a given message as result if the associated pattern matches. This can for instance be used to send alert messages to administrating staff but also to call operations of a web service. The specification of an application shall define how such a message has to be handled.

Further select functions can be used like the user defined views as UserDefinedSelectFunctions. (see Figure 9)

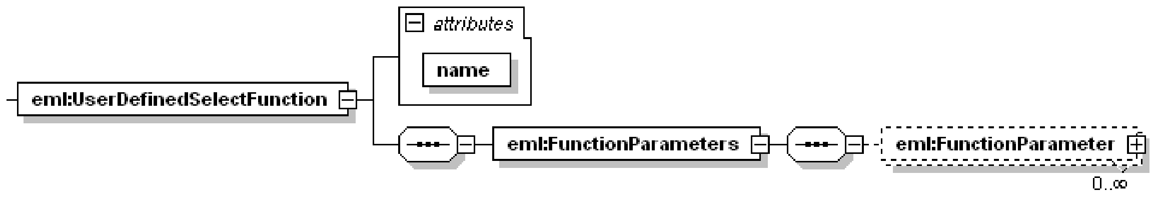


Figure 9: UserDefinedSelectFunction in EML

7.4.2 Further processing

Having performed select functions, the results can be processed further. This can be controlled via three attributes (see following figure).

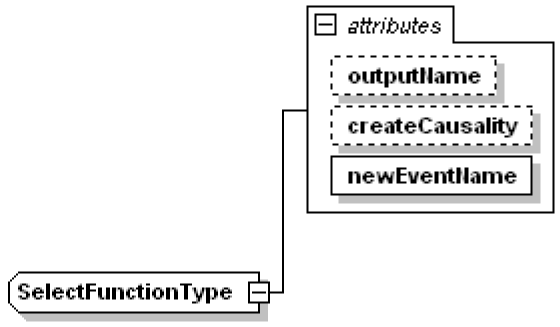


Figure 10: Attributes of every select function in EML

The result of the set of select functions applied by a pattern is stored in a new event object for further use in other event patterns. To reference these new event objects it is necessary to assign a unique name to the new event type via the newEventName attribute.

If createCausality is set to true all the event objects used to generate the result are stored in the causal vector of the new event object. In case of a select function of type 1 (see Table 1) all available event objects are used for generation. This feature enables tracing of events and results. If the value of createCausality is set to false or it is not used, the causal vector of the new event object stays empty.

In addition, the result of the select function can be a result of the entire processing or filtering. Therefore one can use the optional attribute outputName to specify the target for the output. The encoding of the results may depend on the EML implementation or its use. It shall be defined in the application's specification. For the support of the different event processing features it is strongly recommended to use the EML event (see chapter 9) as result encoding. If the outputName is missing or empty, the result of the select function will only be available internally to further patterns in the EML processing chain. Only select functions with a valid ouputName generate results that can be used outside the EML processing.

7.4.3 Property access

When using guards or some views and select functions it is necessary to specify property names. These names must follow an exact schema in order to access the correct properties. Therefore all property names must satisfy the following regular expression:

```
<EventName>/<PropertyName>(/<NestedPropertyName>)*
```

<EventName> is the name of an event object specified by the inputName for simple patterns or the newEventName for other patterns. The <PropertyName> is the name of the property to access. If the property is a collection or again an event object (which may be the case when using SelectEvent) nested properties can be accessed via the <NestedPropertyName>. The names are separated by a forward slash.

If an event object contains only one property (besides the generic properties like time stamps and the causal vector) this property can be accessed via the name „value". Instead of the event name it is also possible to use '*' as wildcard to access the properties of different available event objects.

The names of the generic event properties are „startTime" for the first time stamp, „endTime" for the second time stamp (if the event object represents a point in time the first time stamp is returned when accessing the second, i.e. they are equal) and „causality" for the causal vector. Further fixed names can be defined by applications using EML (e.g. for the geometry of an event).

If some of the properties are not present in an event object the access shall be ignored. This means in case of a guard that the event object does not satisfy it and such an event object will not be added to a restricted view or selected by a select function.

7.5 Complex patterns

Complex patterns are used to correlate data from different sources. This means in EML that one can correlate event objects created by different other event patterns such as simple patterns, timer or repetitive patterns or again complex patterns. The main parts of a complex pattern are the two references to the source patterns and an operator.

The references contain a pattern ID and the number of the select function of the pattern generating the input for the complex pattern (see following figure).

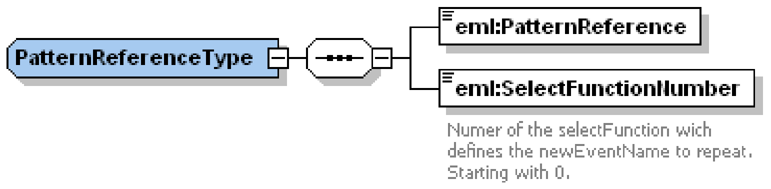

PatternReferenceType
eml:PatternReference
eml:SelectFunctionNumber
Numer of the selectFunction wich defines the newEventName to repeat. Starting with 0.

Figure 11: Reference to an event pattern in EML

Via this number the newEventName parameter of the select function can be referenced which is needed to process the complex event. The numbering of the select function starts with zero and preserves the sequence of the select functions.

Table 3 an overview of the predefined pattern operators that can be used to correlate event objects including three structural operators (CAUSE, PARALLEL and BEFORE) and three logical operators (AND, AND_NOT and OR).

Table 3: Predefined pattern operators in EML

Operator	Description
AND	Match if both sub patterns match.
AND_NOT	Match if the first but not the second pattern matches.
OR	Match if one of the two sub patterns matches.
CAUSE	Match if the matching event object of the first sub pattern is included in the causal vector of the matching event object of the second sub pattern.
PARALLEL	Match if the matching event object of the first sub pattern is not included in the causal vector of the matching event object of the second sub pattern.
BEFORE	Match if a matching event object for the first sub pattern enters the pattern matcher and is followed by a matching event object for the second sub pattern. Note: This operator may generate different results as a complex pattern using AND together with a guard comparing the time stamps of the event objects.

Additional pattern operators can be defined by a service as a new UserDefinedBinaryOperator which needs a name as the only parameter (see chapter 8).

One has to keep in mind that every complex pattern is evaluated sequentially and not in parallel. This means that a complex pattern „A AND B“ generates as result the pair „{A_1, B_1}“ for the event sequence „A_1, A_2, B_1“. The event object A_2 will be skipped because the placeholder for event objects of type A is already in use. A following event object A_3 would be recognized again.

A complex pattern can further make use of select functions, views and guards. A special type of a guard only for complex patterns is the maximum listening duration. It determines a time duration after which a pattern releases all event objects used in placeholders, returns no match and restarts. If for instance this duration would be ten seconds and the time between the arrival of the event objects A_1 and A_2 in the example above would be more than ten seconds, the result of the pattern would be the pair „{A_2, B_1}“.

7.6 Timer patterns

This pattern type defines specialized patterns that generate matches with reference to the system clock. One can either specify a duration after which a timer match is generated (TimerInterval) or specify a position in time when a match is generated (TimerAt, see following figure).

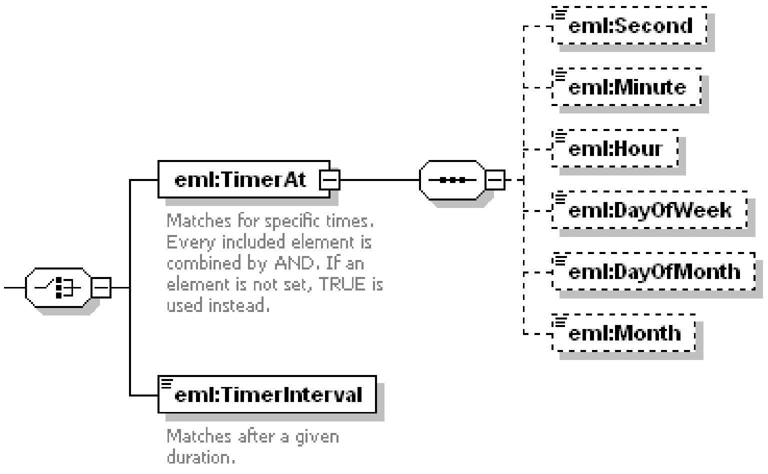

eml:TimerAt
Matches for specific times. Every included element is combined by AND. If an element is not set, TRUE is used instead.
eml:TimerInterval
Matches after a given duration.
eml:Second
eml:Minute
eml:Hour
eml:DayOfWeek
eml:DayOfMonth
eml:Month

Figure 12: Timer patterns in EML

For a position in time, one can specify the value for the second (0 – 59), the minute (0 – 59), the hour (0 – 23), the day of week (1 = monday – 7 = sunday, according to ISO 8601:2004), the day of month (1 – 31) and the month (1 – 12). If every given value matches the actual time the pattern matches.

Timer patterns can make use of select functions and views. This can be useful if one uses a pattern with an interval and wants to know about the absolute date. Guards are not allowed because the specific time position or the duration already defines a guard statement.

7.7 Repetitive patterns

Another pattern in EML is the repetitive pattern. It can be used to count the matches for a sub pattern. Therefore a pattern reference (see complex patterns for details on pattern references) and the desired repetition count must be specified (see Figure 13). If the event count is reached, the last matching event object of the sub pattern is used as result for the repetitive pattern. For example, if a pattern searches for every fifth event of type A and the event sequence is A_1, A_2, B_1, A_3, C_1, A_4, A_5 then only A_5 will match, not the other events of type A.

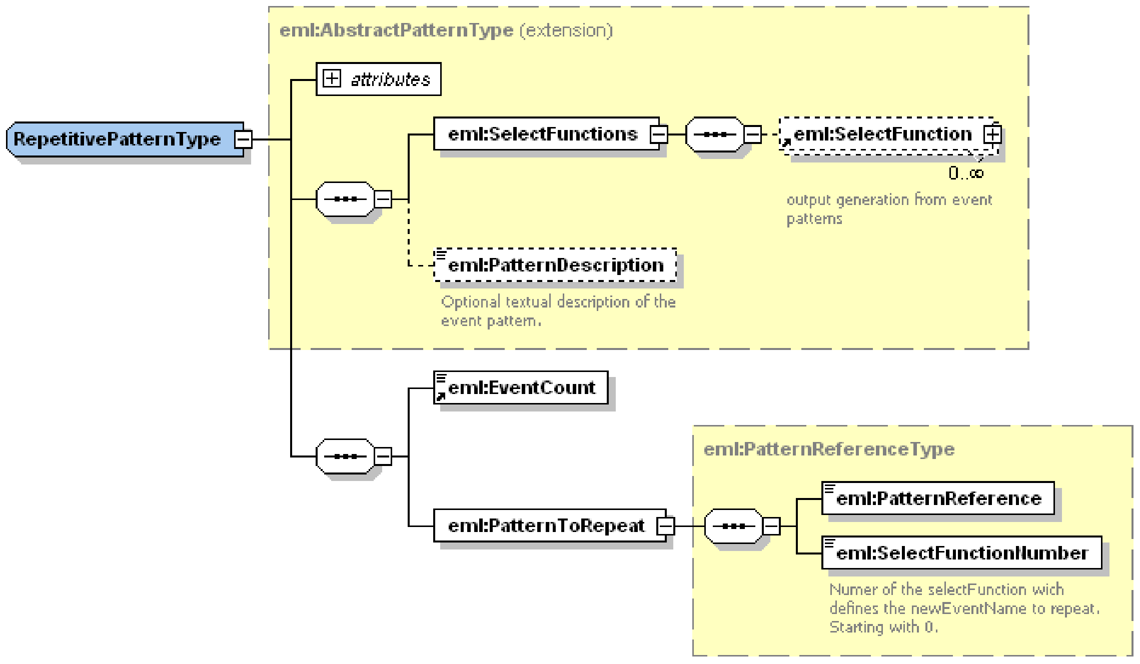

eml:AbstractPatternType (extension)
attributes
RepetitivePatternType
eml:SelectFunctions
eml:SelectFunction
0..∞
output generation from event patterns
eml:PatternDescription
Optional textual description of the event pattern.
eml:EventCount
eml:PatternReferenceType
eml:PatternReference
eml:PatternToRepeat
eml:SelectFunctionNumber
Numer of the selectFunction wich defines the newEventName to repeat. Starting with 0.

Figure 13: Repetitive patterns in EML

Only select functions can be used with repetitive patterns. A view and a guard are not allowed because the repetitive pattern already uses special kinds of them.

8 EML capabilities

Each application using EML shall deliver its EML capabilities to describe the supported views, select functions, complex pattern operators, specialized patterns and the support of wildcards (see Figure 14 to Figure 18). If the EML capabilities contain no entries at least simple patterns, the select function SelectEvent and the view AllView shall be supported.

For each view its name and the names of all parameters shall be given, for every select function its name, the names of all parameters and a declaration whether the creation of the causality between events is supported or not shall be provided. For the supported pattern operators only their names shall be given. If no pattern operator is supported complex patterns are not available at the application. In the section for the specialized pattern capabilities one can specify if repetitive patterns and timer patterns are supported.

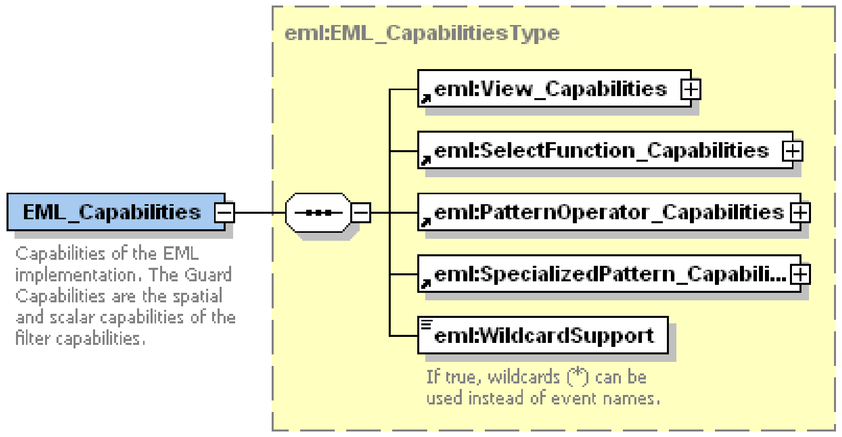

eml:EML_CapabilitiesType
EML_Capabilities
Capabilities of the EML implementation. The Guard Capabilities are the spatial and scalar capabilities of the filter capabilities.
eml:View_Capabilities
eml:SelectFunction_Capabilities
eml:PatternOperator_Capabilities
eml:SpecializedPattern_Capabili...
eml:WildcardSupport
If true, wildcards (*) can be used instead of event names.

Figure 14: Structure of the EML capabilities

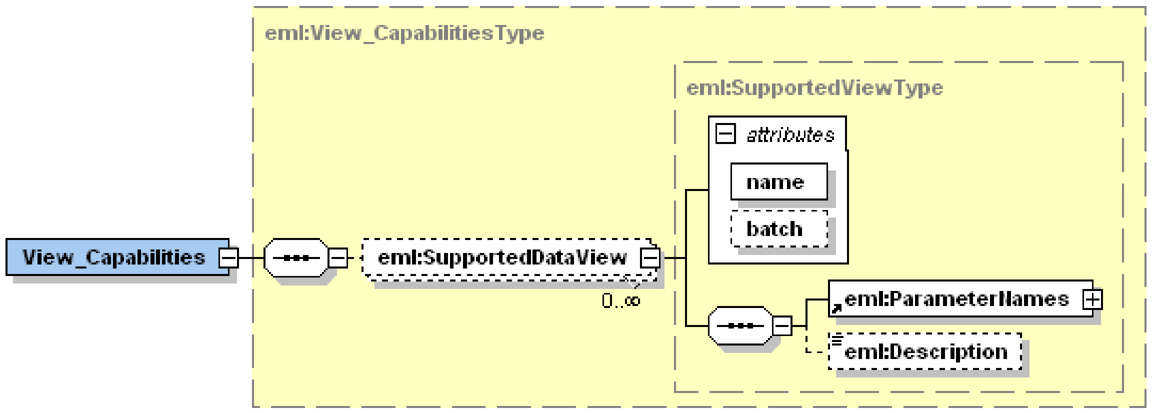


Figure 15: Structure of the EML view capabilities

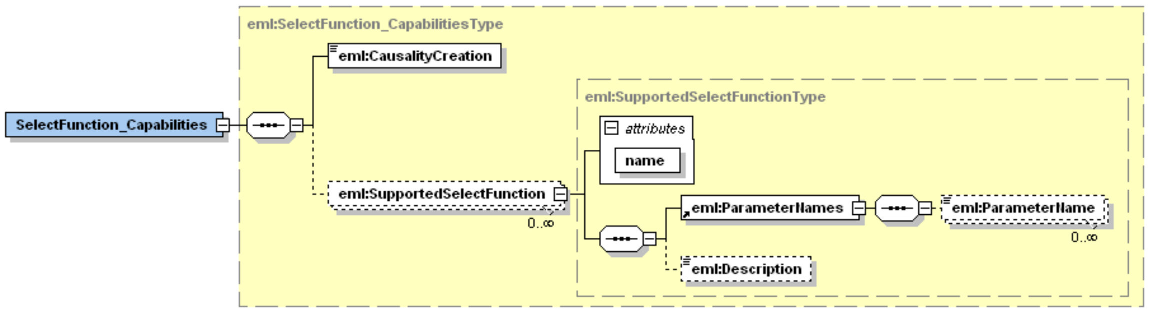

eml:SelectFunction_CapabilitiesType
eml:CausalityCreation
eml:SupportedSelectFunctionType
attributes
name
SelectFunction_Capabilities
eml:SupportedSelectFunction
0..∞
eml:ParameterNames
eml:ParameterName
0..∞
eml:Description

Figure 16: Structure of the EML select function capabilities

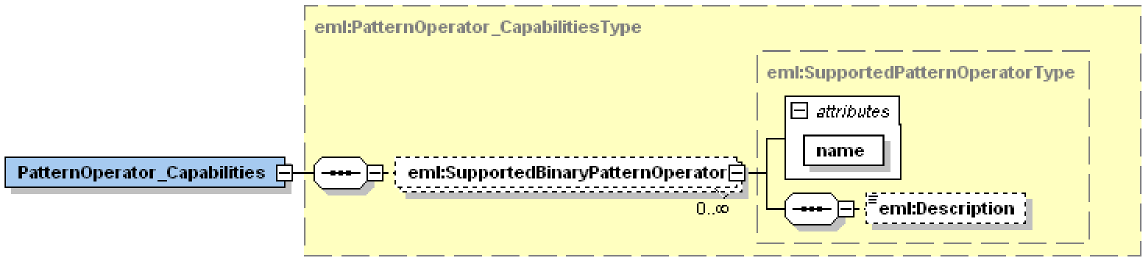

eml:PatternOperator_CapabilitiesType
eml:SupportedPatternOperatorType
attributes
name
PatternOperator_Capabilities
eml:SupportedBinaryPatternOperator
0..∞
eml:Description

Figure 17: Structure of the EML pattern operator capabilities

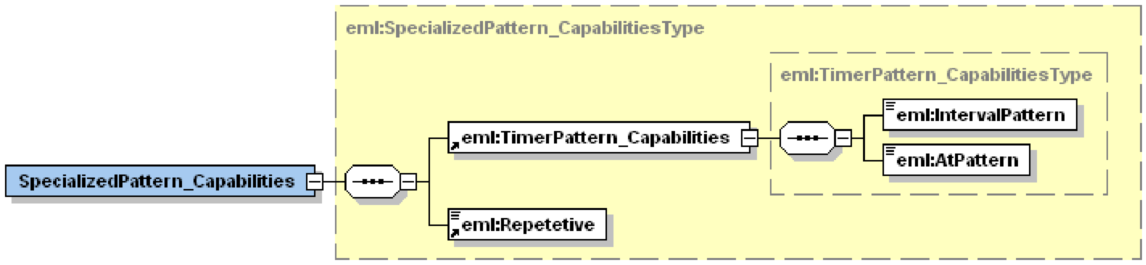

eml:SpecializedPattern_CapabilitiesType
eml:TimerPattern_CapabilitiesType
eml:IntervalPattern
eml:TimerPattern_Capabilities
eml:AtPattern
SpecializedPattern_Capabilities
eml:Repetetive

Figure 18: Structure of the EML specialized pattern capabilities

9 EML event

The EML event defines a data structure for hierarchical event objects. It is strongly recommended to use it for the encoding of event objects, especially when they are used in multiple applications and services. The hierarchy is given by a tree like structure. An EML event can either be a node event object or a leaf event object.

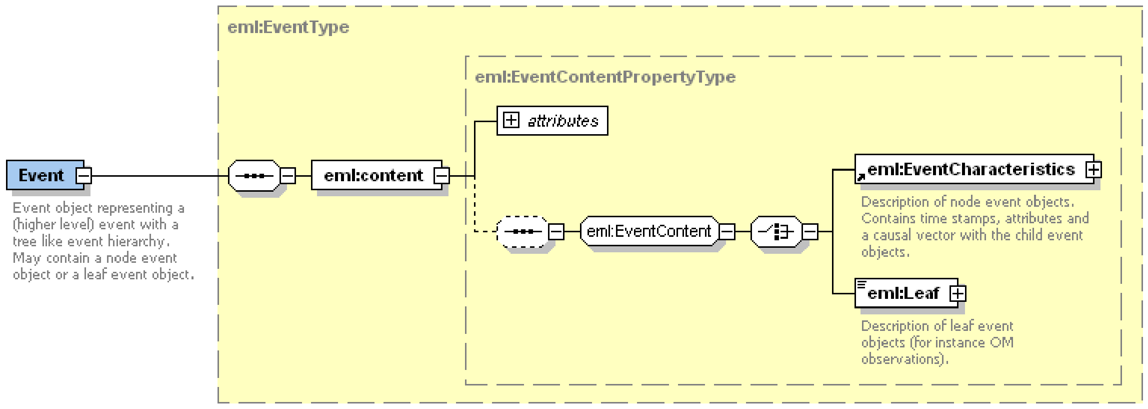

eml:EventType
eml:EventContentPropertyType
attributes
Event
eml:content
eml:EventCharacteristics
Description of node event objects. Contains time stamps, attributes and a causal vector with the child event objects.
Event object representing a (higher level) event with a tree like event hierarchy. May contain a node event object or a leaf event object.
eml:EventContent
eml:Leaf
Description of leaf event objects (for instance OM observations).

Figure 19: EML event object schema overview

The node event objects contain the EventCharacteristics with the time stamps (start and end time), the event attributes as key-value pairs and the causal vector of the event object (see Figure 20). The causal vector contains the event objects which led to the event object at hand. These event objects are again EML events.

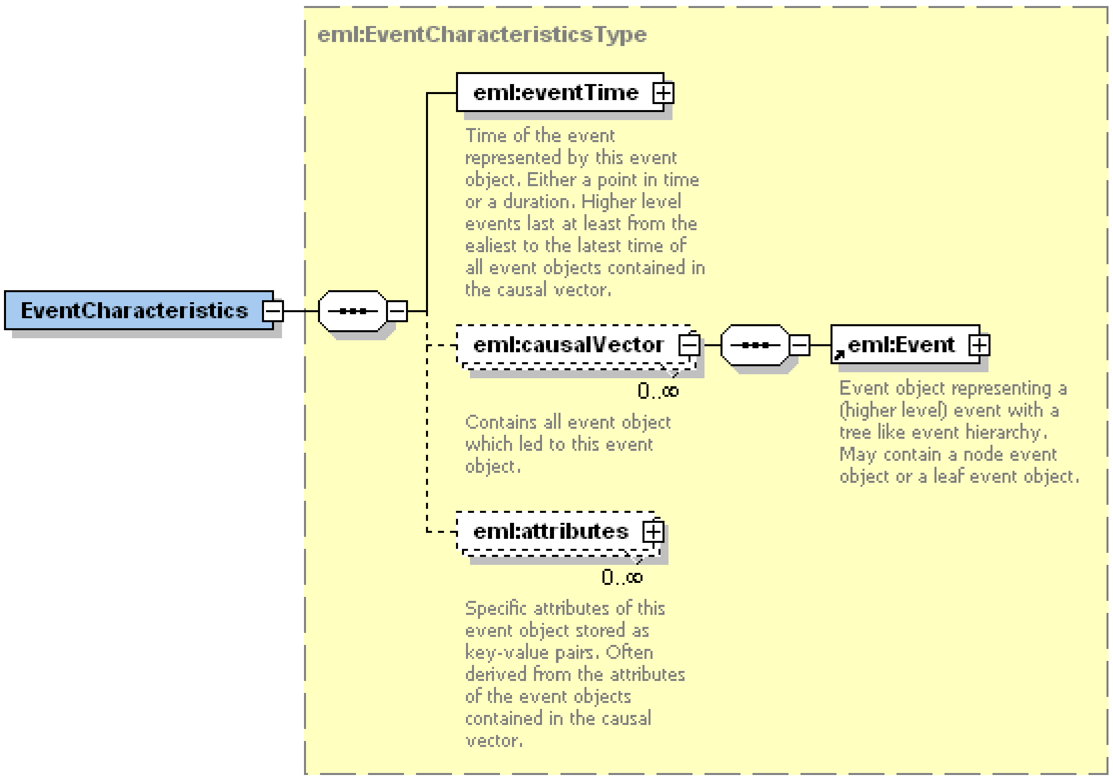

eml:EventCharacteristicsType
EventCharacteristics
eml:eventTime
Time of the event represented by this event object. Either a point in time or a duration. Higher level events last at least from the ealiest to the latest time of all event objects contained in the causal vector.
eml:causalVector
0..∞
Contains all event object which led to this event object.
eml:Event
Event object representing a (higher level) event with a tree like event hierarchy. May contain a node event object or a leaf event object.
eml:attributes
0..∞
Specific attributes of this event object stored as key-value pairs. Often derived from the attributes of the event objects contained in the causal vector.

Figure 20: EML EventCharacteristics overview

The event attributes usually contain derived higher level data such as the average of a set of values (the attribute values of the event objects contained in the causal vector). The value can be of any type, for instance a GML or a SWECommon data type (see [3]).

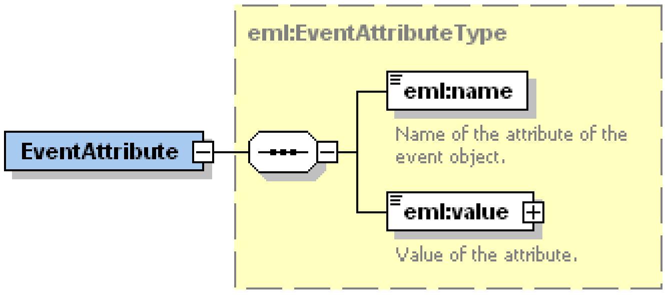


Figure 21: EML EventAttribute overview

The leaf event objects contain a leaf of the event hierarchy tree instead of the EventCharacteristics. This Leaf can be of any type. (Also EML events are allowed here, but it is strongly recommended to store EML events as node event objects.) An example for the content of a leaf event object would be an O&M observation (see [2]) as a starting point of a higher level EML event. It is recommended to use the leaf elements only for unprocessed data or for results of simple patterns (via ouputName).

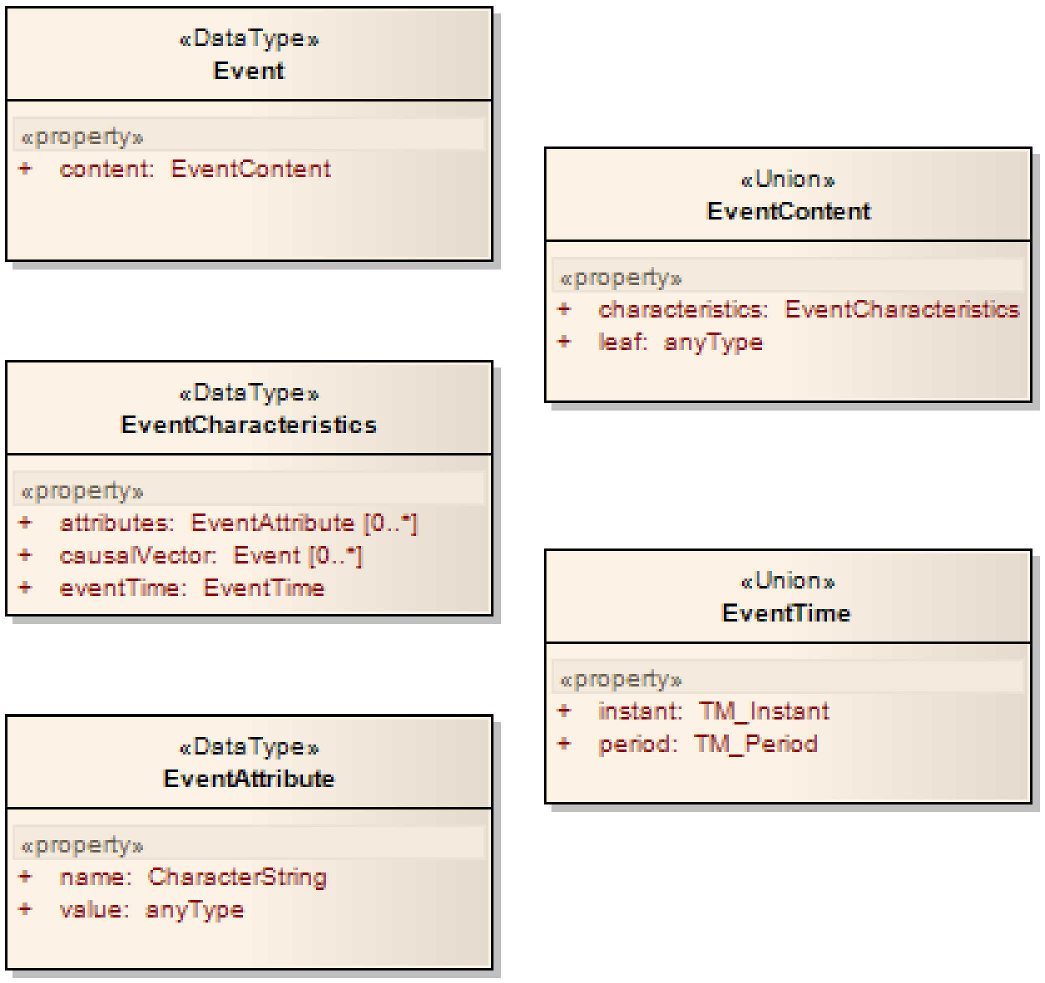


Figure 22: Conceptual model of the EML event

To make use of the EML event in an application it may be necessary to encapsulate it into other elements. For instance, in SensorML descriptions for process in- and outputs only types included in SWE Common “AnyData” are allowed (see [3]). To use the EML event in such an application one could specify a new data type inheriting the SWE Common “AbstractDataRecord” and containing an EML event (see Figure 23).

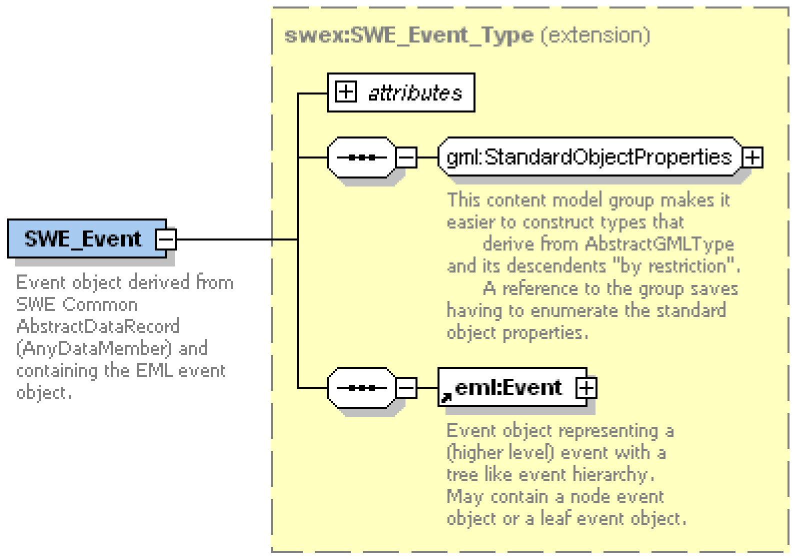

swex:SWE_Event_Type (extension)
attributes
gml:StandardObjectProperties
This content model group makes it easier to construct types that derive from AbstractGMLType and its descendents "by restriction". A reference to the group saves having to enumerate the standard object properties.
SWE_Event
Event object derived from SWE Common AbstractDataRecord (AnyDataMember) and containing the EML event object.
eml:Event
Event object representing a (higher level) event with a tree like event hierarchy. May contain a node event object or a leaf event object.

Figure 23: Example for the integration of the EML event into SensorML

Annex A
(normative)

Abstract test suite

Specific compliance tests for a pattern matcher have not yet been determined.

At the moment an application using EML must satisfy the following system characteristics to be minimally conformant with this specification.

1. An EML pattern expression encoded according to this specification must validate relative to the normative schemas defined in Annex B.

2. An EML capabilities document encoded according to this specification must validate relative to the normative schemas defined in Annex B.

3. An EML event object encoded according to this specification must validate relative to the normative schemas defined in Annex B.

4. The behavior of the pattern matcher executing the EML event patterns must comply with the behavior described in the normative sections of this specification.

5. All clauses in the normative sections of this specification that use the keywords ” required”, “shall” and “shall not” have been satisfied.

Annex B
(normative)

XML Schema Documents

B.1 eml.xsd

The eml.xsd schema contains the schemas for the EML patterns, the EML capabilities and the EML data type for event objects.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:eml="http://www.opengis.net/eml/0.0" targetNamespace="http://www.opengis.net/eml/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified">
	<!--Event Pattern Markup Language (EML) with patterns, capabilities and data type for event
objects.-->
	<xs:include schemaLocation="emlPatterns.xsd"/>
	<xs:include schemaLocation="emlCapabilities.xsd"/>
	<xs:include schemaLocation="emlEvent.xsd"/>
</xs:schema>
```

B.2 emlPatterns.xsd

The emlPatterns.xsd schema contains the definition for the encoding of EML event patterns.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:eml="http://www.opengis.net/eml/0.0" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:gml="http://www.opengis.net/gml" xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:swex="http://www.opengis.net/swex/0.0.1"
targetNamespace="http://www.opengis.net/eml/0.0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<!--= Imports ================-->
	<xs:import namespace="http://www.opengis.net/swe/1.0.1" schemaLocation="..\..
\sweCommon\1.0.1\simpleTypes.xsd"/>
	<xs:import namespace="http://www.opengis.net/ogc" schemaLocation="..\..
\filter\1.1.0\filter.xsd"/>
	<xs:import namespace="http://www.opengis.net/gml" schemaLocation="..\..
\gml\3.1.1\base\temporal.xsd"/>
	<xs:include schemaLocation="emlEvent.xsd"/>
	<!--========================-->
	<xs:element name="EML">
		<xs:annotation>
			<xs:documentation>root element</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence>
				<xs:element name="SimplePatterns">
					<xs:complexType>
						<xs:sequence>
```

```
                              <xs:element ref="eml:SimplePattern" minOccurs="0"
maxOccurs="unbounded"/>
                          </xs:sequence>
                      </xs:complexType>
                  </xs:element>
                  <xs:element name="ComplexPatterns">
                      <xs:complexType>
                          <xs:sequence>
                              <xs:element ref="eml:ComplexPattern" minOccurs="0"
maxOccurs="unbounded"/>
                          </xs:sequence>
                      </xs:complexType>
                  </xs:element>
                  <xs:element name="TimerPatterns">
                      <xs:complexType>
                          <xs:sequence>
                              <xs:element ref="eml:TimerPattern" minOccurs="0"
maxOccurs="unbounded"/>
                          </xs:sequence>
                      </xs:complexType>
                  </xs:element>
                  <xs:element name="RepetitivePatterns">
                      <xs:complexType>
                          <xs:sequence>
                              <xs:element ref="eml:RepetitivePattern" minOccurs="0"
maxOccurs="unbounded"/>
                          </xs:sequence>
                      </xs:complexType>
                  </xs:element>
              </xs:sequence>
          </xs:complexType>
      </xs:element>
      <!--======================-->
      <xs:element name="AbstractPattern" type="eml:AbstractPatternType" abstract="true">
          <xs:annotation>
              <xs:documentation>base substitution group for all patterns</xs:documentation>
          </xs:annotation>
      </xs:element>
      <xs:complexType name="AbstractPatternType">
          <xs:sequence>
              <xs:element name="SelectFunctions">
                  <xs:complexType>
                      <xs:sequence>
                          <xs:element ref="eml:SelectFunction" minOccurs="0"
maxOccurs="unbounded"/>
                      </xs:sequence>
                  </xs:complexType>
              </xs:element>
              <xs:element name="PatternDescription" type="xs:string" minOccurs="0">
                  <xs:annotation>
                      <xs:documentation>Optional textual description of the event
pattern.</xs:documentation>
                  </xs:annotation>
              </xs:element>
          </xs:sequence>
          <xs:attribute name="patternID" type="xs:string" use="required">
              <xs:annotation>
```

```
				<xs:documentation>Unique ID to refer to this pattern.</xs:documentation>
			</xs:annotation>
		</xs:attribute>
	</xs:complexType>
	<!--=======================-->
	<xs:element name="AbstractViewPattern" type="eml:AbstractViewPatternType"
abstract="true" substitutionGroup="eml:AbstractPattern"/>
	<xs:complexType name="AbstractViewPatternType">
		<xs:complexContent>
			<xs:extension base="eml:AbstractPatternType">
				<xs:sequence>
					<xs:element ref="eml:View" minOccurs="0"/>
				</xs:sequence>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<!--=======================-->
	<xs:element name="AbstractGuardedViewPattern"
type="eml:AbstractGuardedViewPatternType" abstract="true"
substitutionGroup="eml:AbstractViewPattern"/>
	<xs:complexType name="AbstractGuardedViewPatternType">
		<xs:complexContent>
			<xs:extension base="eml:AbstractViewPatternType">
				<xs:sequence>
					<xs:element ref="eml:Guard" minOccurs="0"/>
				</xs:sequence>
			</xs:extension>
		</xs:complexContent>
	</xs:complexType>
	<!--=======================-->
	<xs:element name="SimplePattern" type="eml:SimplePatternType"
substitutionGroup="eml:AbstractGuardedViewPattern">
		<xs:annotation>
			<xs:documentation>pattern to match on single events</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="SimplePatternType">
		<xs:complexContent>
			<xs:extension base="eml:AbstractGuardedViewPatternType">
				<xs:sequence>
					<xs:element name="PropertyRestrictions">
						<xs:complexType>
							<xs:sequence>
								<xs:element name="PropertyRestriction"
type="eml:EventAttributeType" minOccurs="0" maxOccurs="unbounded"/>
							</xs:sequence>
						</xs:complexType>
					</xs:element>
				</xs:sequence>
				<xs:attribute name="inputName" type="xs:string" use="required">
					<xs:annotation>
						<xs:documentation>Reference to the input for this Event Pattern. Must
be unique for the application.</xs:documentation>
					</xs:annotation>
				</xs:attribute>
			</xs:extension>
		</xs:complexContent>
```

```
    </xs:complexType>
    <!--=======================-->
    <xs:element name="ComplexPattern"
substitutionGroup="eml:AbstractGuardedViewPattern">
        <xs:annotation>
            <xs:documentation>pattern to match multiple events</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="eml:ComplexPatternType"/>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="ComplexPatternType">
        <xs:complexContent>
            <xs:extension base="eml:AbstractGuardedViewPatternType">
                <xs:sequence>
                    <xs:choice>
                        <xs:element name="StructuralOperator">
                            <xs:complexType>
                                <xs:choice>
                                    <xs:element name="CAUSE">
                                        <xs:annotation>
                                            <xs:documentation>Matches if all events that
satisfy the first pattern are cause of all events that satisfy the second
pattern.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="PARALLEL">
                                        <xs:annotation>
                                            <xs:documentation>Matches if all events that
satisfy the first pattern are not cause of all events that satisfy the second
pattern.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="BEFORE"/>
                                </xs:choice>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="Logicaloperator">
                            <xs:complexType>
                                <xs:choice>
                                    <xs:element name="AND">
                                        <xs:annotation>
                                            <xs:documentation>Matches if both patterns
match.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="AND_NOT">
                                        <xs:annotation>
                                            <xs:documentation>Matches if the first but not
the second pattern matches.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="OR">
                                        <xs:annotation>
```

```
                                                <xs:documentation>Matches if one of the two
patterns matches.</xs:documentation>
                                            </xs:annotation>
                                        </xs:element>
                                    </xs:choice>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="UserDefindeBinaryOperator"
type="eml:UserDefinedOperatorType"/>
                        </xs:choice>
                        <xs:element name="FirstPattern" type="eml:PatternReferenceType"/>
                        <xs:element name="SecondPattern" type="eml:PatternReferenceType"/>
                        <xs:element name="MaximumListeningDuration" type="xs:duration"
minOccurs="0">
                            <xs:annotation>
                                <xs:documentation>Allows to specify a maximum duration in
witch a pattern must match. The duration starts with the first match of of one of the inner patterns.
If the pattern does not match during the duration the first match is discarded and the pattern
listens for new matches.</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <!--=======================-->
        <xs:element name="TimerPattern" type="eml:TimerPatternType"
substitutionGroup="eml:AbstractViewPattern">
            <xs:annotation>
                <xs:documentation>pattern to match on system clock events</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="TimerPatternType">
            <xs:complexContent>
                <xs:extension base="eml:AbstractViewPatternType">
                    <xs:choice>
                        <xs:element name="TimerAt">
                            <xs:annotation>
                                <xs:documentation>Matches for specific times. Every included
element is combined by AND. If an element is not set, TRUE is used
instead.</xs:documentation>
                            </xs:annotation>
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Second" type="eml:MinuteSecondType"
minOccurs="0"/>
                                    <xs:element name="Minute" type="eml:MinuteSecondType"
minOccurs="0"/>
                                    <xs:element name="Hour" type="eml:HourType"
minOccurs="0"/>
                                    <xs:element name="DayOfWeek"
type="eml:DayOfWeekType" minOccurs="0"/>
                                    <xs:element name="DayOfMonth"
type="eml:DayOfMonthType" minOccurs="0"/>
                                    <xs:element name="Month" type="eml:MonthType"
minOccurs="0"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="TimerInterval" type="xs:duration">
                            <xs:annotation>
                                <xs:documentation>Matches after a given
duration.</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:simpleType name="MinuteSecondType">
            <xs:restriction base="xs:unsignedByte">
                <xs:minInclusive value="0"/>
                <xs:maxInclusive value="59"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="HourType">
            <xs:restriction base="xs:unsignedByte">
                <xs:minInclusive value="0"/>
                <xs:maxInclusive value="23"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="DayOfWeekType">
            <xs:restriction base="xs:unsignedByte">
                <xs:minInclusive value="0"/>
                <xs:maxInclusive value="6"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="DayOfMonthType">
            <xs:restriction base="xs:unsignedByte">
                <xs:minInclusive value="1"/>
                <xs:maxInclusive value="31"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="MonthType">
            <xs:restriction base="xs:unsignedByte">
                <xs:minInclusive value="1"/>
                <xs:maxInclusive value="12"/>
            </xs:restriction>
        </xs:simpleType>
        <!--======================-->
        <xs:element name="RepetitivePattern" type="eml:RepetitivePatternType"
substitutionGroup="eml:AbstractPattern">
            <xs:annotation>
                <xs:documentation>pattern to match on repetitions</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="RepetitivePatternType">
            <xs:complexContent>
                <xs:extension base="eml:AbstractPatternType">
                    <xs:sequence>
                        <xs:element ref="eml:EventCount"/>
                        <xs:element name="PatternToRepeat"
type="eml:PatternReferenceType"/>
                    </xs:sequence>
```

```
                    </xs:extension>
                </xs:complexContent>
            </xs:complexType>
            <!--========================-->
            <xs:element name="View" type="eml:ViewType">
                <xs:annotation>
                    <xs:documentation>data views like sliding and tumbling
windows</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:complexType name="ViewType">
                <xs:choice>
                    <xs:element name="LengthView">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="eml:EventCount"/>
                            </xs:sequence>
                            <xs:attribute name="isBatch" type="xs:boolean"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="TimeView">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="eml:Duration"/>
                            </xs:sequence>
                            <xs:attribute name="isBatch" type="xs:boolean"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="TimeLengthView">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="eml:EventCount"/>
                                <xs:element ref="eml:Duration"/>
                            </xs:sequence>
                            <xs:attribute name="isBatch" type="xs:boolean"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="AllView"/>
                    <xs:element name="UserDefinedView">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="ViewParameters">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="ViewParameter"
type="eml:UserParameterType" minOccurs="0" maxOccurs="unbounded"/>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                            <xs:attribute name="name" type="xs:string" use="required">
                                <xs:annotation>
                                    <xs:documentation>Name of the view.</xs:documentation>
                                </xs:annotation>
                            </xs:attribute>
                        </xs:complexType>
                    </xs:element>
```

```
        </xs:choice>
    </xs:complexType>
    <!--=========================-->
    <xs:element name="SelectFunction" type="eml:SelectFunctionType">
        <xs:annotation>
            <xs:documentation>output generation from event patterns</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="SelectFunctionType">
        <xs:choice>
            <xs:element name="SelectEvent">
                <xs:annotation>
                    <xs:documentation>Selects the whole event..</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="eventName" type="xs:string" use="required">
                        <xs:annotation>
                            <xs:documentation>Events with this name will be
selected.</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
            <xs:element name="SelectProperty">
                <xs:annotation>
                    <xs:documentation>Selects a property of an event.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="propertyName" type="xs:string" use="required">
                        <xs:annotation>
                            <xs:documentation>Name of an event property. Used to specify
the property to be slected. </xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
            <xs:element name="SelectSum">
                <xs:annotation>
                    <xs:documentation>Selects the sum of a property of all available
events.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="propertyName" type="xs:string" use="required">
                        <xs:annotation>
                            <xs:documentation>Name of an event property. Used to specify
the property to be slected. </xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
            <xs:element name="SelectAvg">
                <xs:annotation>
                    <xs:documentation>Selects the average of a property of all available
events.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="propertyName" type="xs:string" use="required">
```

```
                                    <xs:annotation>
                                          <xs:documentation>Name of an event property. Used to specify
the property to be slected. </xs:documentation>
                                    </xs:annotation>
                              </xs:attribute>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="SelectMax">
                        <xs:annotation>
                              <xs:documentation>Selects the maximun of a property of all available
events.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:attribute name="propertyName" type="xs:string" use="required">
                                    <xs:annotation>
                                          <xs:documentation>Name of an event property. Used to specify
the property to be slected. </xs:documentation>
                                    </xs:annotation>
                              </xs:attribute>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="SelectMin">
                        <xs:annotation>
                              <xs:documentation>Selects the minimum of a property of all available
events.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:attribute name="propertyName" type="xs:string" use="required">
                                    <xs:annotation>
                                          <xs:documentation>Name of an event property. Used to specify
the property to be slected. </xs:documentation>
                                    </xs:annotation>
                              </xs:attribute>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="SelectCount">
                        <xs:annotation>
                              <xs:documentation>Selects the count of the available
events.</xs:documentation>
                        </xs:annotation>
                  </xs:element>
                  <xs:element name="NotifyOnSelect">
                        <xs:annotation>
                              <xs:documentation>If an event is fired a message is sent instead of
selecting the event. Can be used for system diagnostics.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Message" type="xs:string">
                                          <xs:annotation>
                                                <xs:documentation>Message sent via the referred
output.</xs:documentation>
                                          </xs:annotation>
                                    </xs:element>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
```

```
			<xs:element name="UserDefinedSelectFunction">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="FunctionParameters">
							<xs:complexType>
								<xs:sequence>
									<xs:element name="FunctionParameter"
type="eml:UserParameterType" minOccurs="0" maxOccurs="unbounded"/>
								</xs:sequence>
							</xs:complexType>
						</xs:element>
					</xs:sequence>
					<xs:attribute name="name" type="xs:string" use="required"/>
				</xs:complexType>
			</xs:element>
		</xs:choice>
		<xs:attribute name="outputName" type="xs:string" use="optional">
			<xs:annotation>
				<xs:documentation>Refers to an output description. If set, the results will be
values for the output.</xs:documentation>
			</xs:annotation>
		</xs:attribute>
		<xs:attribute name="createCausality" type="xs:boolean">
			<xs:annotation>
				<xs:documentation>If set to true,  the causing events will be saved in the causl
vector of the result event. If not used, the default value if "false".</xs:documentation>
			</xs:annotation>
		</xs:attribute>
		<xs:attribute name="newEventName" type="xs:string" use="required">
			<xs:annotation>
				<xs:documentation>Specifies the name of the event that is newly created if the
pattern matches (pattern matched event). This event contains the selected (derived) properties or
events. The name shall be unique for the EML document.</xs:documentation>
			</xs:annotation>
		</xs:attribute>
	</xs:complexType>
	<!--======================-->
	<xs:element name="Guard" type="eml:GuardType"/>
	<xs:complexType name="GuardType">
		<xs:sequence>
			<xs:element ref="ogc:Filter"/>
		</xs:sequence>
	</xs:complexType>
	<!--======================-->
	<xs:complexType name="UserDefinedOperatorType">
		<xs:attribute name="name" type="xs:string" use="required"/>
	</xs:complexType>
	<!--======================-->
	<xs:element name="EventCount" type="xs:positiveInteger"/>
	<xs:complexType name="EventCountType">
		<xs:simpleContent>
			<xs:extension base="xs:unsignedInt"/>
		</xs:simpleContent>
	</xs:complexType>
	<!--======================-->
	<xs:element name="Duration" type="xs:duration"/>
	<!--======================-->
	<xs:complexType name="PatternReferenceType">
		<xs:sequence>
			<xs:element name="PatternReference" type="xs:string"/>
			<xs:element name="SelectFunctionNumber" type="xs:int">
				<xs:annotation>
					<xs:documentation>Numer of the selectFunction wich defines the
newEventName to repeat. Starting with 0.</xs:documentation>
				</xs:annotation>
			</xs:element>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="UserParameterType">
		<xs:sequence>
			<xs:element name="UserParameterName" type="xs:string"/>
			<xs:element name="UserParameterValue" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>
</xs:schema>
```

B.3 emlCapabilities.xsd

The emlCapabilities.xsd schema contains the definition for the encoding of the capabilities of an application using EML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:eml="http://www.opengis.net/eml/0.0" xmlns:ogc="http://www.opengis.net/ogc"
targetNamespace="http://www.opengis.net/eml/0.0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<!--========================-->
	<xs:element name="EML_Capabilities" type="eml:EML_CapabilitiesType">
		<xs:annotation>
			<xs:documentation>Capabilities of the EML implementation. The Guard Capabilities
are the spatial and scalar capabilities of the filter capabilities.</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="EML_CapabilitiesType">
		<xs:sequence>
			<xs:element ref="eml:View_Capabilities"/>
			<xs:element ref="eml:SelectFunction_Capabilities"/>
			<xs:element ref="eml:PatternOperator_Capabilities"/>
			<xs:element ref="eml:SpecializedPattern_Capabilities"/>
			<xs:element name="WildcardSupport" type="xs:boolean">
				<xs:annotation>
					<xs:documentation>If true, wildcards (*) can be used instead of event
names.</xs:documentation>
				</xs:annotation>
			</xs:element>
		</xs:sequence>
	</xs:complexType>
	<!--========================-->
	<xs:element name="View_Capabilities" type="eml:View_CapabilitiesType"/>
	<xs:complexType name="View_CapabilitiesType">
		<xs:sequence>
```

```
			<xs:element name="SupportedDataView" type="eml:SupportedViewType"
minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="SupportedViewType">
		<xs:sequence>
			<xs:element ref="eml:ParameterNames"/>
			<xs:element name="Description" type="xs:string" minOccurs="0"/>
		</xs:sequence>
		<xs:attribute name="name" type="xs:QName" use="required">
			<xs:annotation>
				<xs:documentation>Name of the supported data view
(window).</xs:documentation>
			</xs:annotation>
		</xs:attribute>
		<xs:attribute name="batch" type="xs:boolean">
			<xs:annotation>
				<xs:documentation>If set to "true", the programm supports this window as
batch window. Not set = "false".</xs:documentation>
			</xs:annotation>
		</xs:attribute>
	</xs:complexType>
	<!--=======================-->
	<xs:element name="SelectFunction_Capabilities"
type="eml:SelectFunction_CapabilitiesType"/>
	<xs:complexType name="SelectFunction_CapabilitiesType">
		<xs:sequence>
			<xs:element name="CausalityCreation" type="xs:boolean">
				<xs:annotation>
					<xs:documentation>If set to "true" the programm can set the causality
(causal vector) of new events.</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="SupportedSelectFunction"
type="eml:SupportedSelectFunctionType" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="SupportedSelectFunctionType">
		<xs:sequence>
			<xs:element ref="eml:ParameterNames"/>
			<xs:element name="Description" type="xs:string" minOccurs="0"/>
		</xs:sequence>
		<xs:attribute name="name" type="xs:QName" use="required"/>
	</xs:complexType>
	<!--=======================-->
	<xs:element name="PatternOperator_Capabilities"
type="eml:PatternOperator_CapabilitiesType"/>
	<xs:complexType name="PatternOperator_CapabilitiesType">
		<xs:sequence>
			<xs:element name="SupportedBinaryPatternOperator"
type="eml:SupportedPatternOperatorType" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="SupportedPatternOperatorType">
		<xs:sequence>
			<xs:element name="Description" type="xs:string" minOccurs="0"/>
		</xs:sequence>
		<xs:attribute name="name" type="xs:QName" use="required"/>
	</xs:complexType>
	<!--=========================-->
	<xs:element name="SpecializedPattern_Capabilities"
type="eml:SpecializedPattern_CapabilitiesType"/>
	<xs:complexType name="SpecializedPattern_CapabilitiesType">
		<xs:sequence>
			<xs:element ref="eml:TimerPattern_Capabilities"/>
			<xs:element ref="eml:Repetetive"/>
		</xs:sequence>
	</xs:complexType>
	<!--=========================-->
	<xs:element name="TimerPattern_Capabilities" type="eml:TimerPattern_CapabilitiesType"/>
	<xs:complexType name="TimerPattern_CapabilitiesType">
		<xs:sequence>
			<xs:element name="IntervalPattern" type="xs:boolean"/>
			<xs:element name="AtPattern" type="xs:boolean"/>
		</xs:sequence>
	</xs:complexType>
	<!--=========================-->
	<xs:element name="Repetetive" type="xs:boolean" default="false"/>
	<!--=========================-->
	<xs:element name="SystemDiagnostics" type="xs:boolean"/>
	<xs:element name="ParameterNames">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="ParameterName" type="xs:QName" minOccurs="0"
maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

B.4 emlEvent.xsd

The emlEvent.xsd schema contains the definition of the encoding of event objects.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml" xmlns:eml="http://www.opengis.net/eml/0.0"
targetNamespace="http://www.opengis.net/eml/0.0" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
	<xs:element name="Event" type="eml:EventType">
		<xs:annotation>
			<xs:documentation>Event object representing a (higher level) event with a tree like
event hierarchy. May contain a node event object or a leaf event object.</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="EventType">
		<xs:sequence>
			<xs:element name="content" type="eml:EventContentPropertyType">
				<xs:annotation>
					<xs:appinfo>
						<gml:targetElement>EventContent</gml:targetElement>
```

```
                    </xs:appinfo>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="EventPropertyType">
        <xs:sequence>
            <xs:element ref="eml:Event"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="EventCharacteristics" type="eml:EventCharacteristicsType"/>
    <xs:complexType name="EventCharacteristicsType">
        <xs:sequence>
            <xs:element name="eventTime" type="eml:EventTimePropertyType">
                <xs:annotation>
                    <xs:appinfo>
                        <gml:targetElement>EventTime</gml:targetElement>
                    </xs:appinfo>
                    <xs:documentation>Time of the event represented by this event object.
Either a point in time or a duration. Higher level events last at least from the ealiest to the latest
time of all event objects contained in the causal vector.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="causalVector" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Contains all event object which led to this event
object.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="eml:Event" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="attributes" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Specific attributes of this event object stored as key-
value pairs. Often derived from the attributes of the event objects contained in the causal
vector.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="eml:EventAttribute" maxOccurs="unbounded">
                            <xs:annotation>
                                <xs:documentation>Attribute of an event object as key-value
pair.</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="EventCharacteristicsPropertyType">
        <xs:sequence>
            <xs:element ref="eml:EventCharacteristics"/>
        </xs:sequence>
```

```
	</xs:complexType>
	<xs:element name="EventAttribute" type="eml:EventAttributeType"/>
	<xs:complexType name="EventAttributeType">
		<xs:sequence>
			<xs:element name="name" type="xs:token">
				<xs:annotation>
					<xs:documentation>Name of the attribute of the event 
object.</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="value" type="xs:anyType">
				<xs:annotation>
					<xs:documentation>Value of the attribute.</xs:documentation>
				</xs:annotation>
			</xs:element>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="EventAttributePropertyType">
		<xs:sequence>
			<xs:element ref="eml:EventAttribute"/>
		</xs:sequence>
	</xs:complexType>
	<xs:group name="EventContent">
		<xs:choice>
			<xs:element ref="eml:EventCharacteristics">
				<xs:annotation>
					<xs:documentation>Description of node event objects. Contains time 
stamps, attributes and a causal vector with the child event objects.</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="Leaf" type="xs:anyType">
				<xs:annotation>
					<xs:documentation>Description of leaf event objects (for instance OM 
observations).</xs:documentation>
				</xs:annotation>
			</xs:element>
		</xs:choice>
	</xs:group>
	<xs:complexType name="EventContentPropertyType">
		<xs:sequence minOccurs="0">
			<xs:group ref="eml:EventContent"/>
		</xs:sequence>
		<xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
		<xs:attribute name="unionSemantics" type="eml:EventContentUnionSemantics"/>
	</xs:complexType>
	<xs:simpleType name="EventContentUnionSemantics">
		<xs:restriction base="xs:string">
			<xs:enumeration value="characteristics"/>
			<xs:enumeration value="leaf"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:group name="EventTime">
		<xs:choice>
			<xs:element ref="gml:TimeInstant"/>
			<xs:element ref="gml:TimePeriod"/>
		</xs:choice>
	</xs:group>
	<xs:complexType name="EventTimePropertyType">
		<xs:sequence minOccurs="0">
			<xs:group ref="eml:EventTime"/>
		</xs:sequence>
		<xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
		<xs:attribute name="unionSemantics" type="eml:EventTimeUnionSemantics"/>
	</xs:complexType>
	<xs:simpleType name="EventTimeUnionSemantics">
		<xs:restriction base="xs:string">
			<xs:enumeration value="instant"/>
			<xs:enumeration value="period"/>
		</xs:restriction>
	</xs:simpleType>
</xs:schema>
```

Annex C
(informative)

Example XML documents

C.1 Introduction

This annex provides example XML documents for EML patterns, capabilities and event objects.

C.2 Pattern example

The following XML document contains various examples for EML event patterns.

The first simple pattern (patternID “In1”) selects always the newest event from an Input named “In”. It selects the whole event object for further processing and publishes it under the new event name “lastIn1”.

The second simple pattern (patternID “AvgIn) selects the average value (property name “value” of input event object “In”) over the last minute. This value is published as “avgIn”.

The third simple pattern (patternID “notify”) sends a new event object with the content “event object with value = 10 received” to the output connection “MsgOut” and publishes it as “msgIn10” for further processing.

The last simple pattern (patternID “MaxIn”) selects the maximum value of the last 25 received event objects. It uses a view in batch mode. The maximum is published as event object with name “max25”.

The complex pattern (patternID “newMax”) selects an event object if it is received directly after the pattern “MaxIn” delivered a match and if its value is greater than the maximum of the previous 25 event objects. It uses the operator “BEFORE” on the patterns “MaxIn” (select function 0) and “In1” (select function 0) to select from the desired event objects. An OGC filter expression is used as guard to compare the delivered values. If this pattern matches the matching event object is selected and published as “NewMaximum”. The causality is created for this new event object which means that all event object, that took part at the processing of this pattern are stored in the causal vector of “NewMaximum”.

The timer pattern (patternID “Sec10”) matches every ten seconds from application start. As there is no select function specified, selectEvent with parameter “*” ist used. The new event object is published under the patternID.

The repetitive pattern (patternID “ComplexCount”) matches after the complex pattern matched five times. Then a new event object with the message content “Pattern matched five times” is created and published as “5timesComplex” and also send to the output connection “MsgOut”.

```
<?xml version="1.0" encoding="UTF-8"?>
<eml:EML xsi:schemaLocation="http://www.opengis.net/eml/0.0 ../emlPatterns.xsd"
xmlns:eml="http://www.opengis.net/eml/0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc">
<!--###################
	Simple patterns
###################-->
	<eml:SimplePatterns>
		<!-- select the newest event from stream 'In'-->
		<eml:SimplePattern inputName="In" patternID="In1">
			<eml:SelectFunctions>
				<eml:SelectFunction newEventName="lastIn1">
					<eml:SelectEvent eventName="In"></eml:SelectEvent>
				</eml:SelectFunction>
			</eml:SelectFunctions>
			<eml:View>
				<eml:LengthView>
					<eml:EventCount>1</eml:EventCount>
				</eml:LengthView>
			</eml:View>
			<eml:PropertyRestrictions></eml:PropertyRestrictions>
		</eml:SimplePattern>
		<!--select the average of the last minute from stream 'In'-->
		<eml:SimplePattern inputName="In" patternID="AvgIn">
			<eml:SelectFunctions>
				<eml:SelectFunction newEventName="avgIn">
					<eml:SelectAvg propertyName="In/value"></eml:SelectAvg>
				</eml:SelectFunction>
			</eml:SelectFunctions>
			<eml:View>
				<eml:TimeView>
					<eml:Duration>PT0H1M0S</eml:Duration>
				</eml:TimeView>
			</eml:View>
			<eml:PropertyRestrictions></eml:PropertyRestrictions>
		</eml:SimplePattern>
		<eml:SimplePattern inputName="In" patternID="notify">
			<!-- notifies if an event with value 10 is received-->
			<eml:SelectFunctions>
				<eml:SelectFunction newEventName="msgIn10" outputName="MsgOut">
					<eml:NotifyOnSelect>
						<eml:Message>event object with value = 10 received</eml:Message>
					</eml:NotifyOnSelect>
				</eml:SelectFunction>
			</eml:SelectFunctions>
			<eml:View>
				<eml:LengthView>
					<eml:EventCount>1</eml:EventCount>
				</eml:LengthView>
			</eml:View>
			<eml:PropertyRestrictions></eml:PropertyRestrictions>
```

```
        </eml:SimplePattern>
        <!--selectes the maximum value of the last 25 received events in batch mode -->
        <eml:SimplePattern inputName="In" patternID="MaxIn">
            <eml:SelectFunctions>
                <eml:SelectFunction newEventName="max25">
                    <eml:SelectMax propertyName="In/value"></eml:SelectMax>
                </eml:SelectFunction>
            </eml:SelectFunctions>
            <eml:View>
                <eml:LengthView isBatch="true">
                    <eml:EventCount>25</eml:EventCount>
                </eml:LengthView>
            </eml:View>
            <eml:PropertyRestrictions></eml:PropertyRestrictions>
        </eml:SimplePattern>
    </eml:SimplePatterns>
<!--###################
    Complex patterns
###################-->
    <eml:ComplexPatterns>
        <!--select the newest input (pattern In1) when its value is gretater that the maximum of
the previous group of 25 events-->
        <eml:ComplexPattern patternID="newMax">
            <eml:SelectFunctions>
                <eml:SelectFunction newEventName="NewMaximum" createCausality="true">
                    <eml:SelectEvent eventName="lastIn1/In"></eml:SelectEvent>
                </eml:SelectFunction>
            </eml:SelectFunctions>
            <eml:Guard>
                <ogc:Filter>
                    <ogc:PropertyIsGreaterThan>
                        <ogc:PropertyName>lastIn1/In/value</ogc:PropertyName>
                        <ogc:PropertyName>max25/value</ogc:PropertyName>
                    </ogc:PropertyIsGreaterThan>
                </ogc:Filter>
            </eml:Guard>
            <eml:StructuralOperator>
                <eml:BEFORE/>
            </eml:StructuralOperator>
            <eml:FirstPattern>
                <eml:PatternReference>MaxIn</eml:PatternReference>
                <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
            </eml:FirstPattern>
            <eml:SecondPattern>
                <eml:PatternReference>In1</eml:PatternReference>
                <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
            </eml:SecondPattern>
        </eml:ComplexPattern>
    </eml:ComplexPatterns>
<!--###################
    Timer patterns
###################-->
    <eml:TimerPatterns>
    <!--10 second timer-->
        <eml:TimerPattern patternID="Sec10">
            <eml:SelectFunctions>
                <!-- select * is default -->
				</eml:SelectFunctions>
				<eml:TimerInterval>PT0H0M10S</eml:TimerInterval>
			</eml:TimerPattern>
		</eml:TimerPatterns>
<!--##################
		Repetitive patterns
##################-->
		<eml:RepetitivePatterns>
			<!-- alert everytime the complexpattern matched 5 times-->
			<eml:RepetitivePattern patternID="ComplexCount">
				<eml:SelectFunctions>
					<eml:SelectFunction newEventName="5timesComplex"
outputName="MsgOut">
						<eml:NotifyOnSelect>
							<eml:Message>Pattern matched five times</eml:Message>
						</eml:NotifyOnSelect>
					</eml:SelectFunction>
				</eml:SelectFunctions>
				<eml:EventCount>5</eml:EventCount>
				<eml:PatternToRepeat>
					<eml:PatternReference>newMax</eml:PatternReference>
					<eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
				</eml:PatternToRepeat>
			</eml:RepetitivePattern>
		</eml:RepetitivePatterns>
</eml:EML>
```

C.3 Capabilities example

The following XML document contains an example for EML capabilities.

```
<?xml version="1.0" encoding="UTF-8"?>
<eml:EML_Capabilities xsi:schemaLocation="http://www.opengis.net/eml/0.0
../emlCapabilities.xsd" xmlns:eml="http://www.opengis.net/eml/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
			<!--
				##############################
				EML capabilities
				##############################
			-->
			<!--
				View Capabilities
			-->
			<eml:View_Capabilities>
				<!-- AllView -->
				<eml:SupportedDataView name="AllView">
					<eml:ParameterNames/>
					<eml:Description>Holds all events.</eml:Description>
				</eml:SupportedDataView>
				<!-- TimeView -->
				<eml:SupportedDataView name="TimeView">
					<eml:ParameterNames>
						<eml:ParameterName>Duration</eml:ParameterName>
					</eml:ParameterNames>
					<eml:Description>Holds every event for a given duration.</eml:Description>
			</eml:SupportedDataView>
			<!-- TimeView_batch -->
			<eml:SupportedDataView name="TimeView" batch="true">
				<eml:ParameterNames>
					<eml:ParameterName>Duration</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Collects events for a given duration and tries to match
then.</eml:Description>
			</eml:SupportedDataView>
			<!-- LengthView -->
			<eml:SupportedDataView name="LengthView">
				<eml:ParameterNames>
					<eml:ParameterName>EventCount</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Holds a given number of events.</eml:Description>
			</eml:SupportedDataView>
			<!-- LengthView_batch -->
			<eml:SupportedDataView name="LengthView" batch="true">
				<eml:ParameterNames>
					<eml:ParameterName>EventCount</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Collects a given number of events and tries to match
then.</eml:Description>
			</eml:SupportedDataView>
			<!-- TimeLengthView -->
			<eml:SupportedDataView name="TimeLengthView" batch="true">
				<eml:ParameterNames>
					<eml:ParameterName>Duration</eml:ParameterName>
					<eml:ParameterName>EventCount</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>
					Collects events for a given duration or a given number whatever is reached
first. Tries to match then.
				</eml:Description>
			</eml:SupportedDataView>
		</eml:View_Capabilities>
		<!--
			Select Function Capabilities
		-->
		<eml:SelectFunction_Capabilities>
			<!-- causality creation -->
			<eml:CausalityCreation>true</eml:CausalityCreation>
			<!-- SelectEvent -->
			<eml:SupportedSelectFunction name="SelectEvent">
				<eml:ParameterNames>
					<eml:ParameterName>eventName</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Selects a whole event.</eml:Description>
			</eml:SupportedSelectFunction>
			<!-- SelectProperty -->
			<eml:SupportedSelectFunction name="SelectProperty">
				<eml:ParameterNames>
					<eml:ParameterName>propertyName</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Selects a property of an event.</eml:Description>
			</eml:SupportedSelectFunction>
			<!-- SelectSum -->
			<eml:SupportedSelectFunction name="SelectSum">
				<eml:ParameterNames>
					<eml:ParameterName>propertyName</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Selects the sum of a property of all available 
events</eml:Description>
			</eml:SupportedSelectFunction>
			<!-- SelectAvg -->
			<eml:SupportedSelectFunction name="SelectAvg">
				<eml:ParameterNames>
					<eml:ParameterName>propertyName</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Selects the average of a property of all available 
events.</eml:Description>
			</eml:SupportedSelectFunction>
			<!-- SelectMax -->
			<eml:SupportedSelectFunction name="SelectMax">
				<eml:ParameterNames>
					<eml:ParameterName>propertyName</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Selects the maximum of a property of all available 
events.</eml:Description>
			</eml:SupportedSelectFunction>
			<!-- SelectMin -->
			<eml:SupportedSelectFunction name="SelectMin">
				<eml:ParameterNames>
					<eml:ParameterName>propertyName</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Selects the minimum of a property of all available 
events.</eml:Description>
			</eml:SupportedSelectFunction>
			<!-- SelectCount -->
			<eml:SupportedSelectFunction name="SelectCount">
				<eml:ParameterNames/>
				<eml:Description>Selects the count of availble events.</eml:Description>
			</eml:SupportedSelectFunction>
			<!-- NotifyOnSelect -->
			<eml:SupportedSelectFunction name="NotifyOnSelect">
				<eml:ParameterNames>
					<eml:ParameterName>Message</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Returns (selects) a given message on each 
match.</eml:Description>
			</eml:SupportedSelectFunction>
			<!--SelectStdDev (user defined) -->
			<eml:SupportedSelectFunction name="SelectStdDev">
				<eml:ParameterNames>
					<eml:ParameterName>propertyName</eml:ParameterName>
				</eml:ParameterNames>
				<eml:Description>Selects the standard deviation of a property of all available 
events.</eml:Description>
			</eml:SupportedSelectFunction>
		</eml:SelectFunction_Capabilities>
		<!--
			Pattern Operator Capabilities
		-->
		<eml:PatternOperator_Capabilities>
```

```
                        <!-- logical -->
                        <!-- AND -->
                        <eml:SupportedBinaryPatternOperator name="AND">
                              <eml:Description>Matches if the first and the second pattern
matches.</eml:Description>
                        </eml:SupportedBinaryPatternOperator>
                        <!-- AND_NOT -->
                        <eml:SupportedBinaryPatternOperator name="AND_NOT">
                              <eml:Description>Matches if the first pattern matches, but not the
second.</eml:Description>
                        </eml:SupportedBinaryPatternOperator>
                        <!-- OR -->
                        <eml:SupportedBinaryPatternOperator name="OR">
                              <eml:Description>Matches if the first or the second pattern
matches.</eml:Description>
                        </eml:SupportedBinaryPatternOperator>
                        <!-- structural -->
                        <!-- CAUSE -->
                        <eml:SupportedBinaryPatternOperator name="CAUSE">
                              <eml:Description>Matches if the first pattern match is a causal ancestor of the
second pattern match.</eml:Description>
                        </eml:SupportedBinaryPatternOperator>
                        <!-- PARALLEL -->
                        <eml:SupportedBinaryPatternOperator name="PARALLEL">
                              <eml:Description>Matches if the first pattern match is not a causal ancestor of
the second pattern match.</eml:Description>
                        </eml:SupportedBinaryPatternOperator>
                        <!-- BEFORE -->
                        <eml:SupportedBinaryPatternOperator name="BEFORE">
                              <eml:Description>Matches if the first pattern match happened before the
second pattern match.</eml:Description>
                        </eml:SupportedBinaryPatternOperator>
                  </eml:PatternOperator_Capabilities>
                  <!--
                        Specialized Pattern Capabilities
                  -->
                  <eml:SpecializedPattern_Capabilities>
                        <!-- Timer Pattern Capabilities -->
                        <eml:TimerPattern_Capabilities>
                              <!-- interval patterns -->
                              <eml:IntervalPattern>true</eml:IntervalPattern>
                              <!-- at patterns -->
                              <eml:AtPattern>true</eml:AtPattern>
                        </eml:TimerPattern_Capabilities>
                        <!-- repetitive patterns -->
                        <eml:Repetetive/>
                  </eml:SpecializedPattern_Capabilities>
                  <!--
                        Wildcard support
                  -->
                  <eml:WildcardSupport>true</eml:WildcardSupport>
            </eml:EML_Capabilities>
```

C.4 Event object example

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<eml:Event xsi:schemaLocation="http://www.opengis.net/eml/0.0 ../emlEvent.xsd"
xmlns:eml="http://www.opengis.net/eml/0.0" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:swe="http://www.opengis.net/swe/1.0.1">
	<eml:content>
		<!--complex event-->
		<eml:EventCharacteristics>
			<eml:eventTime>
				<gml:TimePeriod>
					<gml:beginPosition>2008Y8M20DT15H0M0S</gml:beginPosition>
					<gml:endPosition>2008Y8M20DT16H30M0S</gml:endPosition>
				</gml:TimePeriod>
			</eml:eventTime>
			<eml:causalVector>
				<!--inner simple events-->
				<eml:Event>
					<eml:content>
						<eml:Leaf>
							<!--some other event object representation like an O&M
observation-->
							<!-- event at time 2008Y8M20DT15H0M0S-->
						</eml:Leaf>
					</eml:content>
				</eml:Event>
				<eml:Event>
					<eml:content>
						<eml:Leaf>
							<!--some other event object representation like an O&M
observation-->
							<!-- event at time 2008Y8M20DT16H30M0S-->
						</eml:Leaf>
					</eml:content>
				</eml:Event>
			</eml:causalVector>
			<eml:attributes>
				<eml:EventAttribute>
					<eml:name>average</eml:name>
					<eml:value>
						<swe:Quantity>
							<gml:description>average temperature</gml:description>
							<swe:uom code="Cel"/>
							<swe:value>15.4</swe:value>
						</swe:Quantity>
					</eml:value>
				</eml:EventAttribute>
			</eml:attributes>
		</eml:EventCharacteristics>
	</eml:content>
</eml:Event>
```

Bibliography

[1] Luckham, D. (2002), The Power of Events. Published by: Addison-Wesley, Boston, USA.

[2] Observations and Measurements – Part 1 – Observation schema, OGC document 07-022r1.

[3] OpenGIS Sensor Model Language (SensorML) Implementation Specification, OGC document 07-000.