Towards an XML based standard for Timetabling Problems: TTML

Ender Özcan
Yeditepe Üniversitesi
eozcan@cse.yeditepe.edu.tr

Abstract: There is a variety of approaches developed by researchers to solve different instances of timetabling problems. During these studies different data formats are used to represent a timetabling problem instance and its solution, causing difficulties in the evaluation and comparison of approaches and sharing data. In this paper, a model for timetabling problems and a new XML data format for them based on MathML is proposed.

Keywords: timetabling, standard data format, scheduling, XML, MathML

1. INTRODUCTION

Timetabling problems consist in feasible assignment of time-slots to a set of events, subject to a set of constraints. The timetabling problem is an NP complete problem [13]. There are a growing number of solutions to different types of timetabling problems having different types of constraints [1-12, 15, 18-22]. Since there is no common standard on specifying a timetabling problem instance and its solution proposed by a researcher, most of the results cannot be compared and benchmarking becomes almost impossible. Proposal for a common data format for timetabling is initiated by Andrew Cumming at ICPTAT'95. Studies in the area yield a language named SSTL [7, 16]. SSTL has not become a common format as expected, possibly because; it is not that easy to convert existing data to SSTL. Furthermore, most of the research in timetabling is due to some practical need, making the researchers concentrate on solving the problem at their hand, ignoring the data format.

Causmaecker *et. al.* [2] argues that timetabling research community can benefit from Semantic Web, focusing the timetabling ontology, rather than one of the layers of the architecture that requires definition of an Extensible Markup Language (XML). XML lets users to create their own set of tags, enabling them to specify the structure of their documents. Furthermore, XML can be used to define a set of grammar rules to define markup languages. It is an efficient way of representing data on the web as a basis for machine to machine communication. XML documents can be considered to be a globally linked database. There are already defined XML based languages. For example, MathML provides means to use mathematical expressions in the web; Scalable Vector Graphics (SVG) is a language for describing two-dimensional graphics in XML. All the details about technologies related to XML can be found in W3C site [23].

Timetabling problems can be formulized using set theory as described in Section 3 in detail, where a constraint is a function operating on the sets. Hence, MathML provides a basis for the representation of timetabling components. For example, MathML allows users to define completely new content symbols that represent a function or a type or another content markup element. This important feature can be used to standardize some timetabling constraints, providing flexibility for users to define their own constraints as well.

In this paper, Timetabling Markup Language (TTML), an XML based data format for timetabling problems is presented utilizing MathML content markup.

2. TTML: TIMETABLING MARKUP LANGUAGE

It is vital to clearly define and represent the elements of a timetabling problem using TTML. The same requirements explained in previous works will be considered during the process [7, 16].

This section is an overview of TTML tags for content markup to generate a well-formed document. For a world wide accepted format for representing timetabling problems, a working group should come together under W3C from researchers and vendors. TTML will be developed further whether this action is taken or not. Instead of creating a new approach, TTML extends MathML, intensifying the importance of modelling. The elements of TTML are built around MathML. The aim is to address the underlying issues, and come up with possible solutions during the modelling. Note that the conversion between different XML documents with similar contents is easy and this conversion does not require a valid document. Hence, XML Schema is left as a further study. All bold TTML elements are optional elements, “|”

denotes or and "`[]`" denotes one or more occurrence of the element enclosed.

2.1 MathML

MathML is an XML based standard for describing mathematical expressions [23]. Presentation markup defines a particular rendering for an expression, while content markup in the MathML provides a precise encoding of the essential mathematical structure of an expression. Some of the content markup elements include relations, calculus and vector calculus, theory of sets, sequences and series, elementary classical functions and statistics. Note that `declare` element is a MathML constructor for associating default attribute values and values with mathematical objects. In TTML, `declare` is used to associate a name with the defined sets. Attributes `desc` and `name` are proposed for `declare` element in TTML, denoting a short description of the declared item and a unique name associated with it, respectively. Unless it is mentioned otherwise the order of TTML elements are strict.

2.2 Modelling Timetabling Problem Instances

An XML document requires one unique root element. The root element is chosen to be `time-tabling` for a timetabling problem instance. Our first aim should be enabling data exchange; hence a TTML document must include input data and the constraints for the problem instance. Additionally, for the research community, in order to make comparisons, test results obtained from applying an algorithm to the input data should be attached. Further attachments might be required, such as output formats for the solution. For example, course section meeting schedules can be generated as a solution to a timetabling problem, but both schedules of all teachers and students can be required as an output. Hence, a TTML document might declare multiple output formats for the same solution. Main and first level of child elements of a TTML document are illustrated in *Figure 1(a)*. A TTML document can include an *output* format or *test results*, optionally. Element `time-tabling` can have attributes such as, last update, problem type (e.g., *university course timetabling*, *highschool timetabling*, *exam timetabling*, *employee shift timetabling*), etc.

3. MODELLING INPUT DATA

Timetabling problems are constraint optimization problems that can be represented using (V, L, C), forming input data, where $V = \{v_1, v_2, \ldots, v_i, \ldots, v_P\}$ is a set of *variables*, $L = \{ d_1, d_2, \ldots, d_i, \ldots, d_P \}$,is a nonempty set of *domains* of variables, defining the set of possible values for each variable in V and C is a set of *constraints*, where each constraint is defined for some subsets of the variables, specifying the allowable combinations of values for it. This 3-tuple forms the input data for a timetabling problem.

In a timetabling problem, a list or a higher dimensional array of attributes (properties) belonging to a set might be required. Let *attributed set* indicate a set where all the members have attributes. For example, the number of students registered to each course, the distances between classrooms, capacity of the classrooms can be considered as attributes of the related sets. Then the set of courses and the set of classrooms are attributed sets. Note that attribute values might be used while defining the constraints. Considering timetabling problems, we can limit the domain of the attribute values to $\mathbb{R}$, $\mathbb{Z}$, and $\mathbb{S}$.

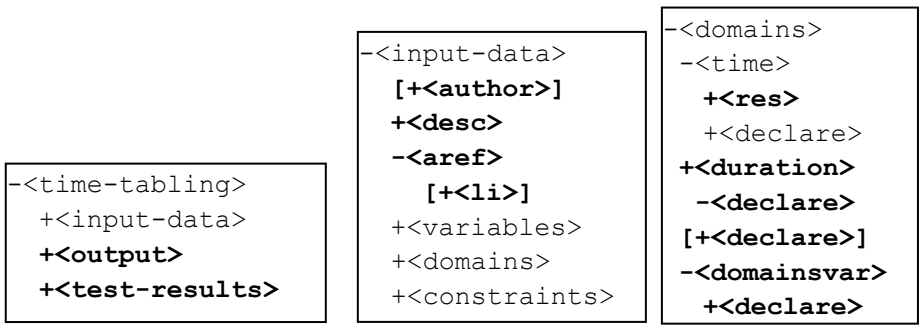

Figure 1. Main and lower level of child elements of a TTML document

Main and first level of child elements of input data are illustrated in *Figure 1(b)*. Elements `author`, `desc` and `aref` describe the author of the timetabling input data, a brief description of the problem and associated references (which might be more than one), respectively. Element `variables` contains a declaration of a single set of variables, identifying each member (*Figure 2(a)*).

3.1 Attributed Sets

Two approaches can be applied to support attributed sets, so that attribute values could be entered as input data. Assuming that higher dimensions can

be mapped into a single dimension, a `vector` can be associated with each set member as shown in *Figure 2(b)*. If there are more than one set of attributes, then it is not that straightforward how to distinguish between them, since they will all be together in the attribute vector. So, TTML shall support the second approach, allowing declaration of single or higher dimensions of attributes associated with the set using `attrset` as shown in *Figure 2(c)*.

Element `attrset` contains two or more declarations. First declaration is a set, and the rest of the declarations (at least one) are the related attributes of it. Attribute declarations must contain `vector` or `matrix` elements where each attribute value will be accessible via `selector` function in MathML (*Figure 2(c)*). All set declarations can be replaced by attributed set declarations in TTML. If the element `attrset` is used to represent a set, then the first declaration is although a set, the order of the elements in the set becomes important, for this reason for the rest of the attribute declarations, that order will be used to identify an attribute value. For example, 24 is the number of registered students. Note that an attributed set might contain more than one set of attributes. It is assumed that the corresponding attribute value of a set member is associated by keeping the same order, wherever the member is defined. For example, assuming CSE462 is the first member of the attributed set then *NoOfStudents* attribute value of it will be the first entry of the corresponding declaration, which is 24 as shown in *Figure 2*. Define `attrval` element, accepting the attributed set member and attribute set name as input and returning the attribute value of an attribute set member.

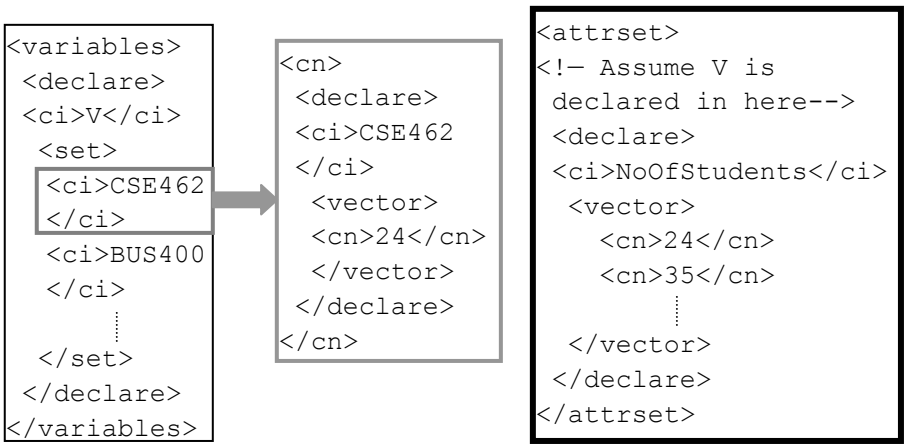

Figure 2. Possible approaches to support attributed sets as input data in TTML

4. MODELLING DOMAINS OF VARIABLES

A candidate solution V' of a timetabling problem is defined by an assignment of values from the domain to the variables:

$V' = \{v_1 = v_1', ..., v_i = v_i', ..., v_P = v_P'\}$, where $v_i' \in d_i$ and

$d_i \subseteq D_1$ x D_2 x … x D_l x … x D_n, $1 \leq n$

While defining the constraints this assignment function might be needed. For this reason, a TTML element `assignvar` is defined as a unary function requiring a single argument which must be a member of the set of variables, returning the assigned value. For example, `assignvar`(v_i) returns v_i'. A domain consists of either time intervals (*time set*) or Cartesian product of several sets, one of them being the time set (D_1=T). If a domain is a Cartesian product of multiple sets then `domainsvar` element should be used for domain declaration in TTML. In such a case, the assignment might be an n-tuple, represented by a `vector` in TTML, allowing access of any *dimension* via `selector` function. For example, assuming a set of courses as a set of variables, `assignvar`(v_i) might return (4,A200), indicating an assignment of the i^{th} course to the 4^{th} time interval in the timetable which will meet in the classroom A200. Selecting the 1^{st} element in the vector returns 4, 2^{nd} element returns A200.

It is possible that in some timetabling problems, durations might be also in the domain of a variable. TTML shall support declaration of duration set using `duration` element. Each member of duration set must be of type duration as explained in the following section. All of the related sets must be *declared* in a TTML document as domains of variables as shown in *Figure 1(c)*.

In timetabling problems, a timetable is either discrete or continuous. In TTML, a combination of both is also supported for generality. *Time intervals* in a timetable might have equal or unequal length, or be periodic or non-periodic. In some problems date, in some others date and time might be required. TTML shall support all.

4.1 Modelling Time Interval and Duration

A time interval can be represented using a starting time and a *duration*. MathML does not contain any type definition related to time or duration, but it allows user-defined types. Similar definitions for `dateTime` and `duration` types in XML schema are proposed to describe a time interval in TTML. In order to get rid of the confusion and be able to use a total order on time, Coordinated Universal Time (UTC) is chosen using the syntax CCYY-MM-DDThh:mm:ss. Duration type syntax is P*n*Y*n*M*n*DT*n*H*n*M*n*S, indicating the number (*n*) of years (Y), months (M), and so on. Any

substring generated using the syntaxes defined above will be valid, assuming that the string includes at least one time item. Duration set as a domain of a variable will be composed of members that are of duration type. This set shall be bound to a name using a declaration as shown in *Figure 1(c)*, if used. TTML shall support three functions; `tistart`, `tiduration` and `tiend` returning the starting time, duration and end of a time interval, requiring a single parameter.

4.2 Modelling Timetables

User should be able to define its own formatting string, emphasizing the time elements relevant to the problem and then the timetable. In TTML, `res` element will be used to state the format of the time used in the timetable definition. For example, the quantity `10-10T10:00 <sep/> P1H` represents a time interval at the 10th day of October with duration 1 hour, based on the formatting string `<res>MM-DDThh:mm</res>`. A timetable is, ultimately, a set of time intervals. TTML shall support this most general approach, enabling shortcuts. An attribute, named as `interval` is added to the `set` element to identify, whether the time intervals are *continuous* or *discrete*. For example, the time set in *Figure 3(a)*, identifies a discrete timetable with 4 time intervals, where in the first day, first interval starts at 10 with 50 minute duration, second one starts at 11 with 50 minute duration, and in the second day, first interval starts at 10 with 50 minute duration, second one starts at 11 with 50 minute duration. A timetable can be assumed to be a two dimensional structure, as the name suggests. We can consider that, this structure contains a number of columns. Each column element is ordered within itself and each column is ordered as well, providing a total order on time. Three functions are proposed for defining a timetable as a domain of variables: `spread`, `spreadcolumn`, and `tmatrix`. Usage of these functions is illustrated in *Figure 3(b)*, *(c)*, and *(d)*. The *spreadcolumn* function repeats a given set of time intervals for a given number of times by interleaving a given duration in between them and returns a time set (*Figure 3(b)*). The *spread* function repeats a given time interval for a given number of times, forming a column, and then applies *spreadcolumn*, using a given interleave and repetition (*Figure 3(c)*).

In some discrete timetabling problems, instead of time intervals, indices indicating a timetable slot can be used. Function *tmatrix*, generates a discrete timetable of a given number of rows and columns, in which each timetable slot is identified by its row and column index and time line proceeds in column major order on the matrix generated (*Figure 3(c)*). Both *spreadcolumn* and *spread* own *interval* attribute indicating whether the timetable is *discrete* or *continuous*. Furthermore, in discrete case, constraints

will refer to timetable slots using start times for the corresponding time interval, by default. It has been observed that time slots in a discrete timetable might also be referred using two indices; their row and column index, or using a single index, while defining the constraints.

```
<time>
 <declare>
 <ci>T</ci>
  <set
    interval="discrete">
   <cn>1T10<sep/>P50M</cn>
   <cn>1T11<sep/>P50M</cn>
   <cn>2T10<sep/>P50M</cn>
   <cn>2T11<sep/>P50M</cn>
  </set>
 <declare>
</time>
```

(a)

```
<apply>
  <spreadcolumn    interval=
"discrete"/><ci><set>
    <cn>1T10<sep/>P50M</cn>
    <cn>1T11<sep/>P50M</cn>
   </set> </ci>
  <!-- interleave -->
  <cn>P1D</cn>
  <!-- repetition -->
    <cn>2</cn>
</apply>
```

(b)

```
<apply>
  <!—column major order-->
  <tmatrix/>
  <cn>2</cn><!--row-->
   <cn>2 </cn><!--column-->
<apply>
```

(d)

```
<apply>
 <spread
   interval="discrete"/>
  <cn>1T10<sep/>P50M</cn>
  <cn>P1H</cn>
  <cn>2</cn>
  <cn>P1D</cn>
  <cn>2</cn> </apply>
```

(c)

Figure 3. Defining a timetable in TTML

For example, *Figure 3(a)*, *(b)*, *(c)* describes the timetable illustrated in *Figure 4*. Ignoring the dashed lines, *Figure 3(d)* identifies the very same table. There are three ways to refer to the marked time slot in the timetable: 2T10, (1,2), 3 or 2. Single indices 3 and 2 are produced by a column major and row major scan on the timetable, respectively. TTML shall support row-column and column major order single indexing for referrals during constraint declarations other than the default. For this reason, for all table defining functions having discrete intervals, an additional attribute, named as `itype` is proposed, indicating the type of the indexing mechanism to be used for timetable slots. The supported values are `default`, `row-column`, and `column-major`. Depending on the *itype*, TTML shall allow user to identify the start index to be (0,0) or (1,1) for *row-column*, or 0

or 1 for *column-major*. `start` attribute belonging to table defining functions will get value either 0 or 1.

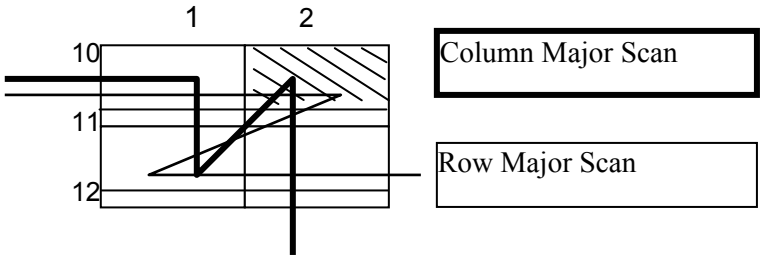

Figure 4. Timetable described in *Figure 3* and indexing mechanisms using a single index.

5. MODELLING CONSTRAINTS

Constraints are classified as *hard* and *soft* for timetabling problems. Hard constraints are the most common constraints. Soft constraints are the preferences that are strongly desired. In general, six different constraint types can be identified for TTPs: *edge constraints*, *exclusions*, *presets*, *ordering constraints*, *event-spread constraint* and *attribute constraints* (includes *capacity constraints*). The details about these constraints can be found in [14].

The problem of determining the minimum number of time slots needed subject to some basic constraints (edge constraints, exclusions, presets), is a graph colouring problem, studied by many researchers [17, 22]. Constraints are functions to be applied on variables or subsets of variables or their related attributes. Since MathML supports user defined functions, *constraints* in TTML are proposed to be *declaration* of *functions* grouped as *hard* / *soft*.

Example: Assume that we have two sets of courses; ES and CS and it is required that no pair of variables should be scheduled at the same time, where each pair is an element of the Cartesian product of ES and CS.

Pairing up all the events that should not overlap and use it as an input data would not be practical, yet a feature that should be supported in TTML. Instead, while defining the constraint function, the sets in question can be used directly and computation of the Cartesian product of sets can be supported by TTML, possibly as in *Figure 5*. Being a function, each constraint requires parameters in TTML. Hence, TTML should allow users to define subsets of variables via *classifiers*, representing logical groupings in a hierarchical way. By this way, user will be able to use the same constraint function for different sets defined in the same TTML document.

Define a *classifier* to be a set which is either a subset of variables, named as *base classifier* or a set of classifiers. Notice that classifiers can form a hierarchy, just like rooted trees. For this reason a *similar* terminology will be used. A *parent classifier* is a classifier having non base classifiers as members. Each member of a parent classifier is called *child classifier*. By default, variables set form a base classifier that should not be redeclared. Revisiting the example, ES and CS classifiers can be defined as base classifiers, and then the constraint function in *Figure 5* would be supported.

```
<apply>
  <forall/> <bvar> <ci> x </ci> <ci> y </ci> </bvar>
  <condition>
    <apply>  <and/>
     <apply> <in/><ci> x </ci><ci> ES </ci> </apply>
     <apply> <in/><ci> y </ci><ci> CS </ci> </apply>
    <apply/>
  </condition>
  <apply> <neq/>
  <ci><apply> <selector/><ci>
    <apply><assignvar/> <ci> x </ci></apply> <ci/>
    <cn>1<cn/></apply></ci>
  <ci><apply> <selector/><ci>
    <apply><assignvar/> <ci> y </ci></apply><ci/>
    <cn>1<cn/></apply></ci>
  </apply>
</apply>
```

Figure 5. A constraint function imposing that no two events one from ES and the other from CS sets should overlap assuming a discrete timetable

In TTML, before the constraint functions are defined, classifiers that will be used in the constraint functions must be declared. Element `set` is used to declare child and base classifiers in a recursive manner. Element `rootcl` is used to declare root classifiers only. Each set is bind to a name using `declare` element. A parent classifier might contain a classifier that is already defined. TTML should avoid redeclarations of the *same* classifiers. Considering all above concerns, `constraints` element is designed as illustrated in *Figure 6*.

Additionally, a function is needed to convert a parent classifier into a subset of variables. For example, assume that we have two base classifiers, one identifying courses with laboratories (ESL), the other identifying courses without labs (ESN) with ES code and ES is a parent classifier such that; ES={ESL, ESN}. Assuming the same for courses in CS code;

CS={CSL, CSN}. Then the constraint function in *Figure 5* cannot be applied on ES and CS. Union of all the members of base classifiers of ES and CS should be generated. Define *self projection* of a parent classifier to be a base classifier, generated by applying union on each member classifier recursively down to the base classifiers. Hence applying self projection on ES and CS would return the expected arguments for the constraint function in *Figure 5*. Define *child projection* of a parent classifier to be a set of base classifiers, generated by applying self projection on each member classifier recursively down to the base classifiers. As an example applying child projection on a parent classifier ALL, defined as ALL={ES, CS}, would return a two member set of self projections of ES and CS. TTML shall support self projection using `self-project` element and child projection using `child-project` element, requiring a single argument that is a parent classifier.

```
-<constraints>
  -<classifiers>
    [+<rootcl>]
  +<hard> |
  +<soft> |
  +<hard>
  +<soft>
```

```
-<rootcl>
 +<declare>
   +<ci>
   -<set>
   [ (+<ci> |
      +<ci>
      +<declare>)
   ]
```

```
-<hard | soft>
[-<coi>
  -<apply>
     +<function> |
   -<declare>
     +<ci>
     +<apply>]
```

Figure 6. Main and the lower level child elements of constraints, where *`function`* element represents a specific constraint function.

Constraints part includes a set of constraint items, indicated by `coi` element, where each item can be defined as a function with or without binding the returned result to an identifier using *declare* element. The reason that the user is allowed to bind the result to an identifier is to support sequential filtering mechanism. The output of applying a constraint can be fed into another constraint as an input. The output of a constraint is discussed in the following section.

5.1 TTML Constraint Functions

Leaving all the constraints to be defined by the user, might lead some problems during data sharing. For example, two different ways can be used to define a semantically same function. For this reason, some basic standard functions should be identified covering at least most of the constraint types. This would also reduce the effort to write all well known constraint functions in MathML.

ID	Functions	Explanation	Semantic
O0	<notsame/> • v_i • v_k	v_i and v_k are not same	$\texttt{assignvar}^{\texttt{dim}}(v_i) \neq \texttt{assignvar}^{\texttt{dim}}(v_k)$
O1	<nooverlap/> • v_i • v_k	No overlap between v_i and v_k	$\texttt{tiend}(\texttt{assignvar}^{\texttt{t}}(v_i)) \leq \texttt{tistart}(\texttt{assignvar}^{\texttt{t}}(v_k))$
O2	<preset/> • v_i • S	Include the set S as the domain of v_i	$\texttt{assignvar}^{\texttt{t}}(v_i) \in S$
O3	<exclude/> • v_i • S	Exclude the set S from the domain of v_i	$\texttt{assignvar}^{\texttt{t}}(v_i) \notin S$
O4	<ordering comp="|>|<|=" /> • v_i • v_k	v_i is after (smaller) | before (larger) | same as (equal to) v_k	$\texttt{assignvar}^{\texttt{t}}(v_i)$ (>|<|=) $\texttt{assignvar}^{\texttt{t}}(v_k)$
O5	<eventspr comp="<|>|=|≤|≥" /> • v_i • v_k • d	The difference between v_i and v_k must be less than| greater than | equal to | greater than or equal to | less than or equal to d	$\texttt{tiend}(\texttt{assignvar}^{\texttt{t}}(v_i)) + d$ (>| <|=| ≤|≥) $\texttt{tistart}(\texttt{assignvar}^{\texttt{t}}(v_k))$
O6	<fullspr per="*duration*" all= "yes|no" comp=">|<|=|avr|≤|≥" /> • V_i • d	The total number of assignments of each variable in set V_i per *duration* throughout the whole timetable has to be greater than | less than | equal to | on average | less than or equal to | greater than or equal to d (if the interval contains any assignment (all=no))	
O7	<freespr per="*duration*" block="on|off" all="yes|no" comp=">|<|=|avr|≤|≥" /> • V_i	The total number of empty slots between each variable assignment (assuming consecutive assignments as single block of assignment, if block=on) in set V_i per *duration* throughout the whole timetable has to be greater than | less than | equal to| on average | less than or equal to | greater than or equal to d (if the	

	• *d*	interval contains any assignment (all=no))	
O8	<attrcomp comp=">|<|=|≤|≥" /> • v_i • *a* • *p* • *b* • *r*	Compares the selected attribute *p* value of the variable v_i along a defined dimension *a* and selected attribute *r* value of the assignment along a defined dimension *b*	attrval(assignvar^{a}(v_i), *p*) (>|<|=|≤|≥) attrval(assignvar^{b}(v_i), *r*)
O9	<resnoclash/> • v_i • v_k	If the assignments of the selected dimension (domain) are same for a pair of variables, then there must be no overlap between the time assignment of v_i and v_k,	If assignvar^{dim}(v_i)) ==assignvar^{dim}(v_k)) then tiend(assignvar^{t}(v_i)) ≤ tistart(assignvar^{t}(v_k))
O10	<chksum per="*duration*" tt="common|seperate" comp=">|<|=|avr|≤|≥″ /> • V_i • *d* • *r*	Forms a data structure where each entry spans time slots of the timetable *duration* long. If *r* is *default* and *tt* is *common*, then the function scans assignment of all the elements in V_i and using the timetable mappings of each entry, it increments the related field in the data structure, After the scan is complete, quantity at each field is compared with *d*, using the selected criterion. If *r* is a selected attribute, then the quantity in a field is incremented by the corresponding attribute value of an element in V_i. Setting *tt* to *separate* creates a data structure for each member classifier in V_i.	

Figure 7. Functions assuming that assignvar^{t}(v_i)<assignvar^{t} (v_k), where v_i and v_k are variables and assuming that *t* represents the time dimension of the assignment

It is assumed that constraint functions are control functions checking some conditions and with the elements that do not satisfy a constraint is up to the problem solver. *Figure 7* displays the standard constraint functions supported by TTML. *O0-O5 functions return the set of variables (or pairs of*

variables) that does not satisfy the related constraint. *O5 returns all the pair of variables that does not satisfy the event spread constraint along with the real gap between two events*. O6 and O7 functions are used to define how a group of events should be distributed over the timetable. O6 function is used for determining the distribution of filled slots, while O7 is used for determining the distribution of empty slots between filled slots due to a set of events. For example, a user might impose a constraint of a workload for a student per day. Note that the workload might be required to be distributed to whole week, or this expected workload might be required excluding the days when a student does not have any course. Also, in a student course schedule, minimum empty slots might be required between course meeting blocks. *O6 and O7 return a positive real value as compared to d*. O8 function is for comparing attribute values. For example, the number of students taking a course should not exceed the capacity of a classroom. The number of student is an attribute of a variable, while capacity is an attribute of a classroom. *O8 function is supported for such constraint declarations, returning the variables that do not satisfy the related constraint*. O9 function checks whether two assigned values along a dimension are same or not. If they have same value, then checks for time overlap. This function is for scheduling resources, other than the ones variables represent, without a clash. For example, the constraint imposing that the courses should not be scheduled to the same classrooms at the same time can be defined using O9 (*Figure 8 (c)*). *O9 function returns all the pairs of variables that do not satisfy the constraint*. *O10 function returns an array having an equal size with the timetable divided by the duration, where each entry is an aggregation of a selected quantity at a group of timetable slots determined by the duration*. An entry of the array is a pair. One of the pairs is the absolute difference between the total sum and the entered value and the other is a Boolean flag indicating the comparison result. For example, in a final exam timetabling problem, a schedule disallowing 1500 students to be seated at the same time might be required. O10 is a useful construct to define such constraints.

For no overlap, we know that comparison is made with a pair of time interval items, so no dimension selection is needed, even if the timetabling problem involves a Cartesian product of multiple sets as domains of variables. But in such a case, for other functions, dimension should be selected using `dim` attribute, where corresponding value should be either an index or a name, indicating a domain set, otherwise the *n*-tuple resulting from `assignvar` should be used. Except O1, O6, O7, O8 functions, all the rest have `dim` attribute in TTML. O9 cannot have *time* as a value of its *dim* attribute and O8 function accepts dimension as an input. O1, O6 and O7 can

be used to define only time related constraints on a variable set, where as the rest can be used to define constraints on a selected dimension (domain).

There are 3 more input cases for O1-O5, other than single events:

1. A binary function accepting a single set:
2. A binary function accepting two sets
3. A unary function accepting a single set

For case 1, 2 and 3, self projections of the input sets will be taken, and then the related function will be applied on the resulting base classifier. For example, *nooverlap* function can accept a base classifier as input, indicating that no pair in the set should overlap, or it could accept a parent classifier which will be reduced to a base classifier by self projection. Case 1 can be enriched by several more interpretations. Single set parameter might be a parent classifier and the user would like to apply the binary function on any pair in each child projection. For binary and unary functions accepting sets as their parameters, attribute `projection` is proposed with values "`single|self | child`", indicating a base classifier, self projection of a parent classifier or a child projection of a parent classifier, respectively. Note that O6, O7 and O10 do not accept a single variable as their parameter. Using this feature, the function definition in *Figure 5* reduces to the function call in the constraint declaration in *Figure 8(a)*.

```
<coi>
 <declare>
  <ci>S1</ci>
  <apply>
  <nooverlap
projection ="single"/>
    <ci>ES</ci>
    <ci>CS</ci>
  </apply>
  </declare>
</coi>
```

(a)

```
<coi>
 <apply>
  <notsame
   dim="room"/>
   <ci>S1
   </ci>
 </apply>
</coi>
```

(b)

```
<coi>
 <apply>
  <resnoclash
   dim="room"/>
    <ci>ALL
    </ci>
 </apply>
</coi>
```

(c)

Figure 8. Declaration of a constraint as a function *(a)*, a filtering example *(b)*, the same affect of (b) requiring no filtering, where `ALL=ES U CS` *(c)*

A filtering example is illustrated in *Figure 8(b)*. It is more appropriate to use standard functions, although the same affect can be obtained using filtering as shown in *Figure 8(a)-(c)*.

6. MODELLING OUTPUT AND TEST RESULTS

Modelling output is necessary for a general tool to be able to generate expected visual timetables for a given timetabling problem instance. For example, the schedules of each instructor or/and the schedule of students belonging to the same set can be asked to be produced. In the schedule, user might prefer to see the classroom assignments as well. For this reason, output element should allow displaying *listed* items, where each item is a different view of variable assignments. Each *li* is assigned `info`, an attribute indicating the assigned values of dimensions that will be printed out into the timetable slot. More than one dimension can be referred in *info*, separated by commas. Furthermore, a user might expect to see the classroom schedules. For this reason, *info* can get `variables` as an attribute value. Then selecting a classroom in *li* and *variables* an *info* attribute value, will yield a timetable output of the selected classroom schedule.

Each item can be a single variable, a set of variables (a base classifier) or sets of set of variables (a parent classifier). If it is a set of variables, user might require an output for each variable (`each`), or a single output for all variables (`all`). If it is a parent classifier, user might require an output for each variable in the set of self projection (`each`), or a single output for all variables in the set of self projection (`all`), or an output for each child projection (`child`). For this reason, another attribute is supported to be used with *li*, which is `gentype`, requiring one of the values `each`, `all` or `child`.

Test results are for researchers, containing the best assignment of variables, author of the test results, references, a short description about the problem solver (algorithm), the best result obtained and the evaluation function. Each assignment of a variable is marked by `asi` element. The order of each assignment is assumed to be in the order of how variables are defined. Since each assignment might consist of several values, depending on the domains of variables, each domain item is included inside the element `di`. Main and the lower level child elements of output, test result and the best assignment of variables are shown in *Figure 9(a)*, *(b)* and *(c)*, respectively.

```
-<output>
   [+<li>]
```

```
+<test-results>
   [-<result>]
     [+<author>]
     +<pref>
     +<desc>
     +<best>
     +<penalty>
     +<eval>
```

```
+<best>
   [-<asi>]
     [+<di>]
```

(a) *(b)* *(c)*

Figure 9. Main and the lower level child elements of *(a)* output and *(b)* test results *(c)* best assignment of variables

In most of the timetabling applications, *penalizing* an unsatisfied constraint is traditional. For supporting related evaluation functions TTML allows optional declaration of a penalty value using `penalty` element for each defined constraint.

7. CONCLUSIONS

TTML can model all real-world timetabling problems based on MathML. In some situations, user might be required to use some non-standard variable, domain and constraint declarations. TTML is not a widely accepted standard, but using TTML with standard constraint functions, most of the university course timetabling, highschool timetabling, final exam timetabling and some of the shift timetabling problems can be modelled. TTML requires standardization of more constraint functions to cover more of the employee timetabling problem instances.

TTML is designed to include even the test results for comparison. For the time being, test results consist of the best results compiled from different tests. This definition can be modified to include more information for each test on the runs, such as statistics of number of evaluations, statistics of timings, or properties of the machine on which the experiments are performed, etc. A full TTML document can be retrieved from a web site by an expert application for timetabling. This application can perform experiments on the given data subject to given constraints, and then compare its results with the best results obtained previously. Furthermore, the application can update the TTML document using its best results by modifying the test results part of the retrieved TTML document. TTML requires more work in modelling the evaluation function, additional to determining more standard constraint functions.

TTML provides all the advantages and strengths of XML. Applications can be carried to the Internet, becoming services. A *TTML processor* can be designed having three major components: a parser, problem solver and a solution interpreter. A *multipurpose* TTML processor is the ultimate goal that solves different types of timetabling problems. Using TTML data sharing will be easy and fast. Additionally, TTML provides means to include different parts of other TTML documents in order to make use of previously defined components and their features (variables, constraints, etc.), using Xlink and Xpath technologies ([23]), where the same functionality is provided by SSTL using an object oriented methodology. The requirements

for a standard data format can be summarized as universality (assuming a closed world), completeness and convertibility. The latter requirement is satisfied by TTML, just by being an XML standard. TTML, powered by MathML is a strong candidate for satisfying all these requirements.

Furthermore, recent studies concentrate on case based reasoning approaches, which can benefit from the use of TTML. [4] defines similarity measures to support such approaches. In some problems, it is important to determine the strongly connected components (possibly the largest one) in a graph, mapping timetabling problem into a graph colouring problem. Finding maximal clique is an NP complete problem. If TTML is used and the variables are grouped into classifiers, then the problem of determining the strongly connected components reduces to locating classifier sets as parameters of a related constraint.

A TTML validator is not implemented, since there are many general validators available over the Internet. *Figure 10* includes a well-formed TTML document. The first TTML application, named CONFETI is implemented as a Java applet, providing a user interface to convert final exam timetabling text data into a TTML document. CONFETI will be used to build an instance repository providing TTML documents for final exam timetabling using existing data. Initially, Carter's benchmark data sets are, converted to TTML, based on the constraints defined in [6], successfully. The second application will be available soon; a full TTML processor based on a memetic algorithm for final exam timetabling, named FES (Final Exam Scheduler). The results will be reported soon. The latest developments in TTML and instance repositories will be available at `http://cse.yeditepe.edu.tr/~eozcan/research/TTML`.

```
<?xml version="1.0"?>
<time-tabling>
    <input-data      type="University-Course-Timetabling"
    lastUpdate="2003-01-22T13:20:00.000-05:00">
    <author>Ender Ozcan</author>
    <desc>An example TTML document</desc>
    <variables>
      <attrset>
        <declare> <ci>V</ci>
        <vector>
          <ci duration="2"> CSE211.01</ci>
          <ci duration="2"> CSE211.02</ci>
          <ci> CSE311.01</ci>
          <ci> CSE462.01</ci>
```

```
      </vector>
      </declare>
      <declare> <ci>noOfStudents</ci>
      <vector>
        <ci> 34</ci>
        <ci> 27</ci>
        <ci> 20</ci>
        <ci> 25</ci>
      </vector>
      </declare>
    </attrset>
  </variables>
  <domains>
    <time>
      <declare> <ci>T</ci>
      <apply>
        <tmatrix itype="row-column" start="1">
          <cn> 9</cn> <cn> 5</cn>
        </tmatrix>
      </apply>
      </declare>
    </time>
    <attrset>
     <declare> <ci>classrooms</ci>
       <vector>
        <ci> A100</ci>
        <ci> B101</ci>
        <ci> B103</ci>
        <ci> A201</ci>
       </vector>
       </declare>
      <declare> <ci>capacity</ci>
       <vector>
        <ci> 50</ci>
        <ci> 50</ci>
        <ci> 50</ci>
        <ci> 30</ci>
       </vector>
      </declare>
    </attrset>
    <domainsvar>
      <declare><ci>R</ci>
             <apply><cartesianproduct/>
```

```
                    <ci> T </ci>
                    <ci> classrooms </ci>
              </apply>
      </declare>
    </domainsvar>
  </domains>
  <constraints>
    <classifiers>
      <rootcl> <declare> <ci>lecturers</ci>
          <set> <ci> <declare> <ci> Ender Ozcan </ci>
           <set>
            <ci> CSE211.01</ci>
            <ci> CSE311.01</ci>
           </set>
          </declare> </ci>
          <ci> <declare> <ci> Ferda Dogan </ci>
           <set>
            <ci> CSE211.02</ci>
            <ci> CSE462.01</ci>
           </set>
          </declare> </ci>
      </set> </declare> </rootcl>
      <rootcl> <declare> <ci>curriculum-terms</ci>
          <set> <ci> <declare> <ci> Term#2 </ci>
           <set>
            <ci> CSE211.01</ci>
            <ci> CSE211.02</ci>
            <ci> CSE311.01</ci>
           </set>
          </declare> </ci>
          <ci> <declare> <ci> Term#3 </ci>
           <set>
            <ci> CSE462.01</ci>
           </set>
          </declare>  </ci>
      </set> </declare> </rootcl>
    </classifiers>
    <hard>
      <coi>
        <apply>
          <nooverlap projection="child"/>
            <ci>curriculum-terms</ci>
        </apply>
```

```
    </coi>
    <coi>
      <apply>
        <nooverlap projection="child"/>
          <ci>lecturers</ci>
      </apply>
    </coi> <coi>
      <apply>
        <preset dim="classroom"/>
          <ci>CSE211.01</ci>
          <cn>A100</cn>
      </apply>
    </coi>
    <coi>
      <apply>
        <exclude dim="T"/>
          <ci>CSE311.01</ci>
          <cn><vector> <cn>1<sep/>1</cn>
              <cn>2<sep/>2</cn></vector>
          </cn>
      </apply>
    </coi>
    <coi>
     <apply>
      <resnoclash projection="single"
       dim="classrooms"/>
        <ci>V</ci>
     </apply>
    </coi>
  </hard>
  <soft>
    <coi>
    <apply>
      <attrcomp projection="single" comp="&lt;="/>
        <ci>V</ci>
        <ci>T</ci>
        <ci>noOfStudents</ci>
        <ci>classrooms</ci>
        <ci>capacity</ci>
     </apply>
    </coi>
  </soft>
```

```
    </constraints>
  </input-data>
  <output>
     <li projection="child">lecturers</li>
  </output>
  <test-results>
     <result>
        <author>Ender Ozcan</author>
        <desc>  Problem is solved by TEDI, following is
                the best solution obtained in 50 runs
        </desc>
        <best>
          <asi> <di>4<sep/>1</di> <di>A100</di> </asi>
          <asi> <di>4<sep/>2</di> <di>B101</di> </asi>
          <asi> <di>4<sep/>3</di> <di>A201</di> </asi>
          <asi> <di>4<sep/>4</di> <di>B103</di> </asi>
        </best>
     </result>
  </test-results>
</time-tabling>
```

Figure 10. An example TTML definition of a timetabling problem instance

8. REFERENCES

1. D. Abramson, H. Dang and M. Krisnamoorthy, *Simulated Annealing Cooling Schedules for the School Timetabling Problem*, Asia-Pacific Journal of Op. Res., 16: 1-22, 1999.
2. A. Alkan, E. Ozcan, *Memetic Algorithms for Timetabling*, Proc. of 2003 IEEE Congress on Evolutionary Computation, pp. 1796-1802, December, 2003.
3. P. D. Causmaecker, P. Demeester, Y. Lu and G. Vanden, *Using Web Standards for Timetabling*, PATAT'02, pp.238-258, 2002.
4. E. K. Burke, A. J. Eckersley, B. McCollum, S. Petrovic and R. Qu, *Similarity Measures for Exam Timetabling Problems*, Proc. of the 1st Multidisciplinary Inter. Conf. on Scheduling: Theory and Applications, pp. 120-135, 2003.
5. E. K. Burke, D. Elliman, and R. Weare, *A Genetic Algorithm Based Timetabling System*, Proc. of the 2nd East-West Int. Conf. on Comp. Tech. in Education, pp. 35-40, 1994.
6. E. K. Burke, J.P. Newall, R.F. Weare, *A Memetic Algorithm for University Exam Timetabling*, Lecture Notes in Computer Science, 1153:241-250, Springer, 1996.
7. E. K. Burke, P. A. Pepper and J. H. Kingston, A *Standard Data Format for Timetabling Instances*, Springer Lecture Notes in Computer Science, 1408:213-222, 1997.
8. A. Colorni, M. Dorigo, and V. Maniezzo, *A genetic algorithm to solve the timetable problem*. Tech. rep. 90-060 revised, Politecnico di Milano, Italy, 1992.
9. D.Corne, P. Ross, H.L. Fang, *Evolutionary Timetabling: Practice, Prospects and Work in Progress*, Proceedings of the UK Planning and Scheduling SIG Workshop, 1994.

10. JP Cladeira, AC Rosa, *School Timetabling using Genetic Search*, PATAT'97, pp. 115-122, 1997.
11. F.P.M. Dignum, W.P.M. Nuijten, L.M.A. Janssen, *Solving a Time Tabling Problem by Constraint Satisfaction*, Tech. report, Eindhoven University of Technology, 1995.
12. W. Erben, J. Keppler, *A Genetic Algorithm Solving a Weekly Course-Timetabling Problem*, Proc. of the First Int. Conf. on the Practice and Theory of Automated Timetabling (ICPTAT), pp. 21-32, Napier University, Edinburgh, 1995.
13. S. Even, A. Itai, and A. Shamir, *On the Complexity of Timetable and Multicommodity Flow Problems*, SIAM J. Comput., 5(4):691-703, December 1976.
14. H.L. Fang, *Genetic Algorithms in Timetabling and Scheduling*, PhD thesis, 1994.
15. A. Hertz, *Finding a feasible course schedule using a tabu search*, Discrete Applied Mathematics, 35, 255-270, 1992.
16. J.H. Kingston, *Modeling timetabling problems with STTL*, Springer Lecture Notes in Computer Science, 2079:309, 2001.
17. F.T. Leighton, *A graph coloring algorithm for large scheduling problems*, Journal of Reasearch of the National Bureau of Standards, 84:489-506, 1979.
18. A. Monfroglio, *Timetabling Through a Deductive Database: A Case Study*, Data & Knowledge Engineering, 3:1-27, 1988.
19. E. Ozcan, A. Alkan, *Timetabling using a Steady State Genetic Algorithm*, PATAT'02, pp.104-107, 2002.
20. A. Schaerf, *Tabu Search Techniques for Large High-School Timetabling Problems*, Proc. of the Fourteenth National Conference on AI, pp. 363-368, August, 1996.
21. G. Schmidt, and T. Strohlein, *Time table construction-an annotated bibliography*, The Computer Journal, 23(4):307-316, 1979.
22. D. De Werra, *An introduction to timetabling*, European Journal of Operations Research, 19:151-162, 1985.
23. World Wide Web Consortium web site, http://www.w3c.org, 2004.