Opublikowane w: W seria Frontieres in Artificial Intelligence and Applications, ed. H. Jaakkola, H. Kangassalo, E. Kawaguchi, tom: Information Modelling and Knowledge Bases XII, 01.2001, IOS Press, Holandia.

Modelling Basic Multimedia Notions in UML

Ludwik KUŹNIARZ *)
Departmentof Software Engineering and Computer Science, University of Karskrona/Ronneby, Soft Center, SE-372 25 Ronneby, Sweden
e-mail : Luwdik.Kuzniarz@ipd.hk-r.se
Maciej PIASECKI
Computer Science Department, Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27, PL 50-370 Wrocław, Poland
e-mail : Maciej.Piasecki@ci.pwr.wroc.pl

Abstract. The paper presents an attempt to use UML to model essential structural and dynamic aspects of multimedia data used in multimedia presentations. The proposed model defines a set of few fundamental basic concepts: *logical data units*, *synchronisation point*, *stream, beam, movie* and *user interaction tools*. The concepts are expressed as UML constructs. Both sequential and object perspectives of possible view on multimedia presentation are covered in one model. Moreover, the model allows description of both physical and logical levels of organisation of multimedia presentation. Construction of data hierarchy highlighting different classifications of multimedia data and taking into account their basic static properties is introduced. The model unifies different formal perspectives on the multimedia presentation. Finally, mechanisms of synchronisation of multimedia presentation on intra and inter level, as well as, on the level of user interaction within the presentation are also modelled in UML.

1.Introduction

Among the variety of data present in software systems multimedia data show particular complexity. Their complexity is caused by the fact that both structural as well as temporal relation between their components should be expressed. Specification of typical data in software systems is limited only to the description of their structure but describing multimedia data one has also to express their behaviour. Multimedia presentation is a collection of multimedia data together with time constrains imposed on them. The constraints have the form of both of sequential and parallel occurrence of different events in time. What is more, user interaction tools are also often included into the description of the structure of multimedia presentation. The interaction tools comprise software and hardware elements, which enable the user to have influence on the presentation. This introduces a new level of time constraints, which are changing with subsequent presentations and so are difficult to be precisely expressed and formalised.

*) On leave from Computer Science Department, Wrocław University of Technology

The need of preserving strict time constraints during presentation forces the usage of formal methods for specification of the structure and behaviour of the presentation. Most of the formalisms used for this purpose focus on the description of the time constraints, seeing the structure in simplified way. Moreover, most of the existing formalisms perceives multimedia data only on one of the two levels of the abstraction [1,8]: as simple sequences of some units (streams) or as media objects – 'black boxes' – enclosing some complete part of data and performing some higher level operations (e.g. [3,12]). In consequence it is not possible to express all intrinsic temporal constraints of the multimedia data using only one of these formalisms. Changing a perspective one has also to change formalism. However, the existing formalisms are usually not well suited to be used in parallel, and they are not based on a common set of abstract notions. Moreover, it seems that every media object can be perceived in the same time in some contexts as a sequence and as a complete unit.

Object Oriented Modelling Languages (OOML), introduced in software engineering for specification of software systems, allow to describe both complicated structure and behaviour of software systems and therefore seem to be an appropriate tool for specification of multimedia presentation, while maintaining consistency of the description of the entire system. Moreover, using them for the specification of multimedia data can yield a more structured description still being enough formal.

The paper contains an attempt of applying OOML for description of the structure of the multimedia presentations. The main goal is to build a multilevel unifying model jointly describing in one structure the multimedia presentation on all levels of abstraction: starting with the low level of temporal sequences and finishing with highly abstract level of multimedia presentation scenarios. Unified Modelling Language (UML) [2,13] was chosen here as the specification language. The main reasons were that the language is widely accepted international standard, is becoming widely used and has a considerable expressive power. The presented attempt seems to go in parallel to the attempts aimed to extend UML for specific areas of applications e.g. specification of real-time system requirements [4].

The paper tries to formulate some basic concepts as the base of the description, defining the concepts in UML. The paper is also an attempt to capture and model the most essential parts of the structure and behaviour of multimedia presentation. Presented models are supposed to be a sort of a design pattern, which constitutes a basis for creation of a specific multimedia presentation specification using inheritance and refinement.

The desired goal of the paper is realised as follows. First, few basic concepts are chosen to be a base for the model. They are defined formally as abstract classes. Next, the class diagrams describing properties of multimedia data present in different classifications are built upon them. Then the mechanisms allowing the constructions of complex multimedia data starting from the defined elementary data as well as the means to express data synchronisation are analysed and proposed. This is followed by construction of a model for data used in multimedia presentation expressing both static and dynamic presentation requirements.

2. Media Data Structures

Before proceeding to the analysis and modelling *multimedia data* structures, some clarification of the understanding of the basic notions is needed. *Medium* is understood as the means of information representation, the way of conveying information, and the access to its different forms. *Media data* is a digital representation of an artefact in the area of a certain medium. Properties of media data can be expressed by the notion of *media type* defining data representation and operations, which can be performed on the data.

Diversity of both the properties of data as well as their sources is the reason for the existence of many different independent ways and criteria of their classification. Applying several independent criteria at a time a multidimensional classification can be obtained. The result of application of two most common criteria is presented on Table 1.

Tab. 1 Classification of multimedia data.

	non-temporal (static)	temporal (dynamic)
registered	*image* (digital, rasterised, e.g. TIFF, JPG), *text* (plain)	*motion picture*, *audio*, *speech* (recorded samples)
synthesised	*graphics* (vector e.g. PostScript), *digital ink*	*animation*, *music* (e.g. MIDI), *speech transcription*

Taking into account the source of data *registered* vs. *captured* classification is obtained [1,8]. Registered data are created by the process of quantisation of an artefact created within a certain natural media. In consequence, in the simplest case, the data have plain, one level internal structure –binary coded sequence of samples. Another important criterion is time dependence, leading to *non-temporal* (*static*) and *temporal* (*dynamic*) data [8] (in [1] the namings *time dependent* vs. *time-independent* are used). Temporal data use time as one of their components. They consist of a sequence of elements and time relations between them. In most cases these time relations have the form of simple sequence in time (the ex-

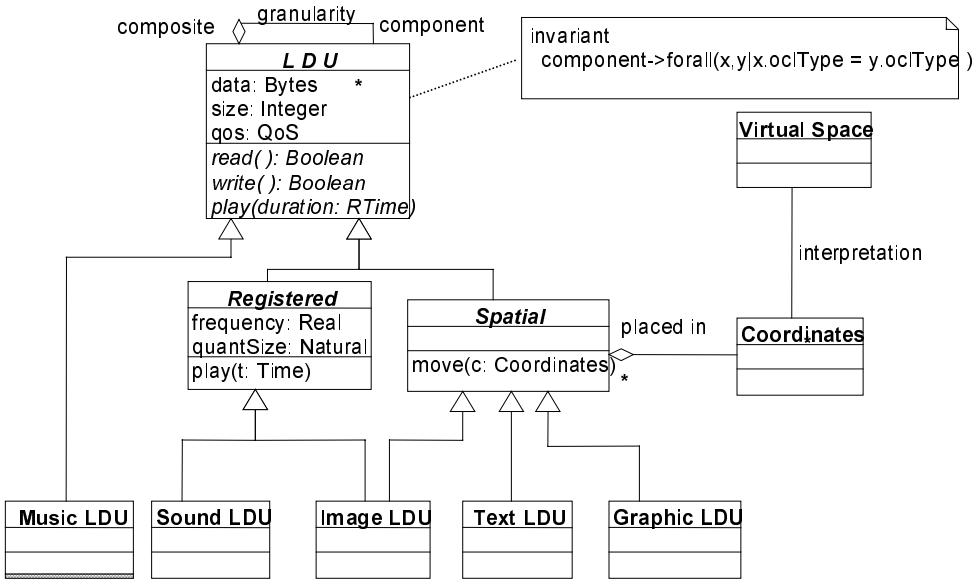

Fig.1. Multimedia data classification

ceptions are some form of animation). Non-temporal data cannot be separated into smaller units with respect to time. This is true on a certain level of abstraction because if we represent data as a sequence of bytes there can always be an imposed time sequence, for instance during their transmission. In the context of multimedia presentation every data is time related – every single datum has to start its presentation at a given point of time, as well as, to finish its presentation after a given period of time. These imply that non-temporal property of some data types is relative.

Another important classification is *simple* vs. *compound*. Simple data consist of units from one interpretation model (such as digital sound for instance), while in compound data units interpreted in a different way can be found (for instance in video, where there are both sound samples as well as frames of motion picture).

Additional criteria based on more detailed properties of data taking into account origins of data and details of the interpretation of their binary form can be formed. Often they do

not have a general character apart of *spatial* vs. *non-spatial* distinction, which seems to be useful for all multimedia data – as the screen constitutes the basic area of presentation.

Most of definitions of multimedia data types assume integrated data processing environment with at least one temporal data type [1,8]. So, in the paper, the temporal data type was chosen as the most representative multimedia data type.

"Time dependent media objects usually consists of a sequence of information units.

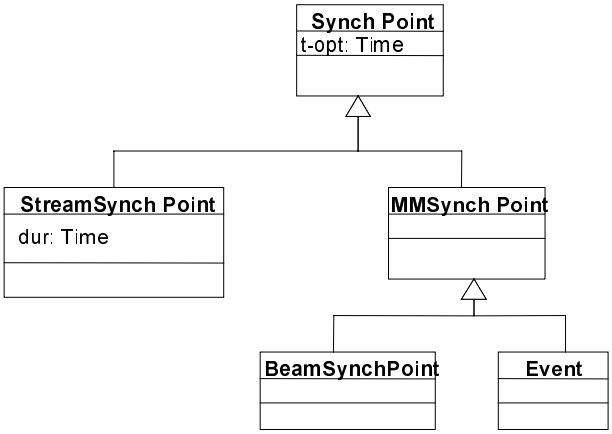

Fig.2. Hierarchy of synchronisation points

Such units are known as *Logical Data Units* (LDUs)." [1]

Investigating algebraic properties of temporal data types led to introduction of formal notion of *stream* [9]. Stream is defined as a sequence of LDUs synchronised in time.

The notion of stream and LDU is a basis for construction of UML model of multimedia data types presented on Fig.1 and Fig.3. Simple temporal data types were used as the most representative for multimedia data types. They:

- are build of LDUs of one type, and
- have incorporated time constraints, imposed by the notion of stream.

Other, mentioned above, multimedia types can be derived from those basic types, as follows:

- non-temporal simple types will be modelled as streams built of exactly one LDU,
- compound types will be modelled as appropriate composition of a number of streams (for instance video is a composition of audio and motion picture)

It is suggested to model multimedia data types using two layers:

- LDU structure layer,
- stream layer.

LDU structure is shown on Fig.1. Abstract class LDU defines a common root for the whole hierarchy. Its contains abstract operations which are to be specified by specialised classes. Only the fact that each data unit contains a piece of information stored in a binary form together with its size is modelled at the top abstract level. The interpretation of the data is made by specialised classes. The *play* operation takes a duration argument – the duration is expressed in real time (in contrast to the time used in stream, Fig. 3, and beam, Fig. 6, where an abstract time type is used). The operations *read* and *write* are introduced in their simplified forms. Omitting some technical details allows formulating a kind of design pattern for modelling of LDU structure, as well as, enables to clarify the picture of the entire structure.

The *Registered* class introduces additional attributes connected with quantisation process: *frequency* of quantisation and *quantSize* – the size of a sample. The string of bytes stored in LDU can contain several samples (e.g. there are two samples in each LDU of a stereo audio and many samples in each LDU (frame) of a motion picture). Registered /

syntetized characteristics are modelled by properties of LDUs and constraints imposed on stream constructions, what will be shown later. The class *Spatial* together with its *Coordinates* in some space (the class *VirtualSpace* is very loosely coupled with LDU of the subclass *Spatial*) introduces LDUs having certain location in the space. In this way spatial / non-spatial dichotomy is realised.

Mutual independence of the classification criteria is visible in multiple inheritance applied to the class *ImageLDU*.

Classes from the lover level of the hierarchy represent (implementation details are hidden) concrete user classes. As it was already mentioned, they mainly introduce interpretation of the data attribute. For instance the class *Graphics LDU* introduces interpretations on data in the form of a formal model.

One more element of the diagram presented on Fig.1 needs explanation. The self aggregation placed in the class LDU expresses the ability to analyse the structure of the stream on different levels of granulity. For instance, in the case of audio we can take into account a single sample, as well as a block or frame [6,14] (e.g. in the case of compressed audio), which consists of many samples.

The second basic element of the multimedia data structure – the stream – is represented on Fig.3. It consists of an ordered sequence of LDUs. The order of LDUs expresses the basic physical, sequential organisation of 'pure' media data. All LDUs in a stream must be of the same type. This fact is expressed by the attached invariant expressed in Object Constraint Language (OCL) [13]. A *synchronisation point* [9] is connected to every LDU. The point is similar to a *reference point* defined in [1]. The objects of the class *StreamSynchPoint* express time constraints imposed on every LDU in the sequence, with attributes: *t-opt* – optimal time for starting the playing and *dur* – optimal duration of the playing. The first of these attributes (*t-opt*) models the low-level editorial decision made on the raw media data stored in the stream[1]. The physical order of LDUs can be rearranged by the sequence of objects of the *StreamSynchPoint* class.

Both attributes of *StreamSynchPoint* class are expressed in abstract stream's time units, which are interpreted by an attached object of *VirtualTimeAxis* class (Fig. 3), where *time_base* attribute defines the number of abstract stream's (beam's etc.) units per second. The class implicitly assumes the existence of an 'ideal' clock, common to the whole multimedia presentation and delivers the operations of: translating the abstract time to real time, according to the time base (*calcRealTime*) and translating the time units between the two different time axes (*transTime*).

Static stream (the basis for modelling of the non-temporal data types) is a stream which consists of exactly one LDU and one *StreamSynchPoint* defining duration time and starting point of the given LDU in relation to the upper layer unit, i.e. *beam*. Objects of the *StreamSynchPoint* class are also a mean of mapping the inter-stream synchronisation scheme on the low level sequential organisation of multimedia data (in MPEG-1 [6] intra-stream synchronisation points correspond to *Decoding Time Stamps* and inter-stream synchronisation points correspond to *Presentation Time Stamps*).

Dynamic stream is a basis for all temporal multimedia types. The temporal registered type is modelled by the *Regular* class. The class requires that all its component LDUs are the subclasses of the class Registered and that the presentation time as well as the duration time are *frequency* dependent.

Stream synchronisation points (*StreamSynchPoints*) play double role:

---

[1] The change in the order of LDUs during presentation can be also caused by purely technical reasons e.g. in MPEG-1, where, because of compression, frames are stored in a slightly different sequence than the sequence in which they are presented [5].

- determine the low level temporal sequential organisation of multimedia data, and
- are the elements which allow to bound the temporal stream and the entire presentation by temporal constraints.

They are also used to specify synchronisation on the lower level and are basis for specification of inter-stream synchronisation [9].

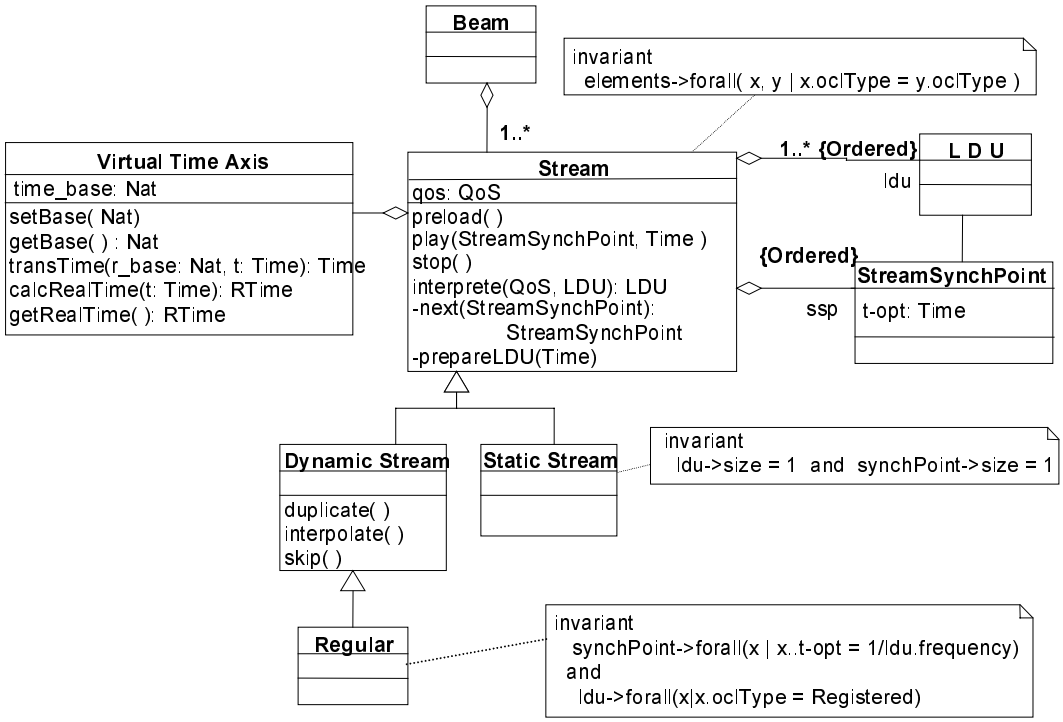

Fig. 3. Multimedia stream.

3. Synchronisation in Multimedia

As in the case of the definition of a multimedia system there is a number of different definitions as well as understandings of synchronisation. It can be understood as [1]:

1. *Content Relation* – „defining a dependency of media objects on some data" (e.g. two objects are both views of the same data or some data of some media object are in the same time a part of a different object);
2. *Spatial Relation* – (another name *layout relationships*) „defining the space used for the presentation of a media object on an output device at a certain point of time";
3. *Temporal Relation* – „defining the temporal dependencies between media objects".

Because the notion of synchronisation is closely related to time, we will leave it in the paper to the last of the enumerated inter-object relations and we will express the remaining relations in the form of independently described associations.

Content relation is a relation of rather technical nature and will not be considered in the model (it can however be easily modelled using UML associations and aggregation).

Spatial relation will be present as an association between streams constructed of the objects from the subclass *Spatial*.

Time constraints named as synchronisation can be classified as:

- *intra-stream synchronisation* [9] (*intra-object synchronisation*, *single media stream synch.* [1]) – describing temporal relations between LDUs in a given stream,

- *inter-stream synchronisation* [9] (*inter-object synchronisation*, *multiple media stream synchronisation* [1]) – describing relations between streams (in [1] it is restricted to the relations between temporal data types, while here this notion is extended),
- *time-dependent/time-independent media synchronisation* [1] – this type of synchronisation will be called *event synchronisation*, because some of the original relations matching the definition has been enclosed here to the second type (i.e. non-temporal multimedia data are also equipped with mechanisms of inter-stream synchronisation).

The difference between the last two is caused only by considering the fact whether the streams bounded by time relations form compound type (modelled as beam), or whether the relation between them is imposed by the *presentation scenario* (understood as multimedia data specification, user interaction tools and algorithm of presentation). In the first case we have inter-stream synchronisation, in the second case we have event synchronisation. Inter-stream synchronisation is usually 'repeatable' – in both streams many mutually related time points expressing synchronisation relation are defined.

Inter-stream synchronisation can be classified as follow [1]:

- *lip synchronisation* –the name and the model come from video (movie with sound), where sound samples and picture frames synchronise in regular intervals of time,
- *pointer synchronisation* – the name comes from co-operative work environment (CSCW), where presentation is a sequence of slides connected with the motion of a pointer device and verbal comments.

In the second case two streams are present: motion picture (or animation) and audio stream, which are synchronised in a repeated but irregular way. Both streams constitute an entity and synchronisation is here an important gluing factor.

Lip synchronisation is possible only between two regular streams.

Many methods of synchronisation specification exist (a review can be find in [1] and [8]). Most of them are limited to certain aspect or applications. Blakowski and Steimetz [1] define *Synchronisation Reference Model* (SRM) which defines abstraction level in specification of multimedia systems and requirements for synchronisation. Basic assumptions of the model will be presented as we will refer our model to them. Fig.4 shows a schema of the model. It consists of four specification levels starting from the lower technical level to the upper abstract level. Each level defines a set of services and imposes certain level of abstraction used in description of multimedia data and operations.

1. *Media Layer*
   - Multimedia data are seen as a single media stream.
   - Operations are restricted to the operations on streams.
   - Specification of synchronisation is restricted to intra-synchronisation.
   - Time constraints between LDUs from different streams are possible to be expressed only by physical interleaving (interleaving LDUs from different streams).
2. *Stream Layer*
   - Based on media streams and groups of media streams.
   - Requires mechanisms to defined both intra as well as inter stream synchronisation.
   - Requires operations on groups of streams.
   - Does not require mechanisms for definitions of synchronisation between temporal and non-temporal data, but special point in the streams can be defined by the use of non-temporal data treated as events.

- Mechanisms to convey values of the Quality of Service (QoS) are required, (QoS is an abstract measure of the quality of multimedia presentation, the value of QoS expresses the minimal value of the quality of presentation).

3. *Object Layer*
   - Definition of presentation for data of all types is required.
   - The difference between temporal and non-temporal data should be hidden.

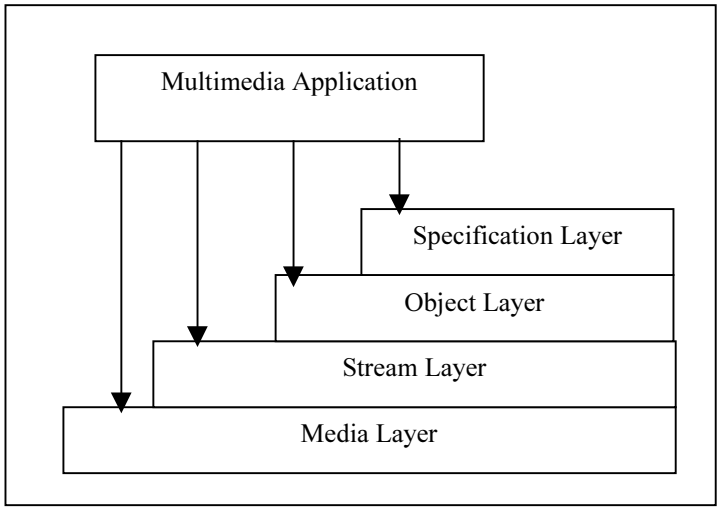

Fig. 4. Four Layer reference model

   - All mechanisms of specification which include QoS should be defined.

4. *Specification Layer*
   - It is an open layer without a specific interface.
   - Languages for synchronisation specification should be supplied.
   - Translation of abstract QoS values (provided by upper layer) to the appropriate properties of the synchronisation mechanisms should be possible.

QoS has an influence on all aspects of multimedia presentation (for instance it ensures proper colour and picture resolution, maximal frequency of audio). In the case of synchronisation QoS decides on the allowed error margin:

- for single regular stream (intra stream synchronisation) it is the maximal value of *jitter* (difference in the interval between subsequent samples);
- for many streams (inter stream synchronisation) it is a *skew* – interval between the first and the last synchronisation stream.

In the presented model the requirement of preserving a given value of QoS will be restricted to synchronisation.

4. Modelling Multimedia Presentation

Multimedia presentation is not only a collection of multimedia data. It contains also spatial and temporal restrictions imposed on the data and those restrictions are not imposed by the data themselves and cannot be derived from them. Those additional constraining relations express desired presentation scenario created by the author of the presentation. For their description, a special level of synchronisation was introduced and named *event synchronisation*.

What is more, additional time constraints can have dynamic character, caused by interaction between the presentation and its user. By introducing additional objects to the pres-

entation by means of user interaction tools, the user may generate events in any unpredictable moment of presentation. Those points of user interaction cannot be expressed in the specification of synchronisation included in the description of multimedia data, i.e. by mechanisms of inter-stream synchronisation delivered by the beam. However, occurrences of such events can be associated with the inter-stream synchronisation level.

In the presented model the *active presentation* paradigm is assumed. It means that dependence of the presentation on its environment is hidden. The object from the presentation can itself perform their operations (e.g. play) without contacting their environment. This paradigm is particularly useful in modelling portable presentations but it requires a closed list of multimedia types. The MHEG [5] standard for multimedia presentation is based on similar assumption. In the presented approach, contrary to MHEG, synchronisation specification mechanisms are defined on all levels of SRM, while MHEG requires two top levels to do this.

Attempts of using UML for modelling certain aspects of user interface were presented at [10], where only static aspects of the interaction were considered. In the model presented here the emphasis is put to more technical aspects of the dynamic relations between multimedia data.

5. UML Model of Multimedia Presentation

Elements of the model presented so far described the level of data structures. The level of description was close to the technical realisation. In this paragraph new classes, which constitute building blocks for multimedia presentation, will be introduced in the increasing order of complexity. Those will be:

- *Beam* – (Fig. 6) which allows to construct complex multimedia types and provides mechanisms for specification of inter-stream synchronisation;
- *Multimedia Movie* (*MMMovie* class) – (Fig. 9) a generalised, interactive multimedia 'motion picture', representing a part of the presentation and providing a virtual screen (*VirtualSpace*) and mechanisms for specification of event synchronisation;
- *User Interaction Tools* (*UI Tools* class) – (Fig. 9) objects that enable interaction between the user and the presentation; their internal structure and behaviour will not be considered here, apart of the assumption that they contain mechanism that can react on user actions and influence the state of presentation by appropriate event synchronisation (it is only assumed that user actions, done on a object of *UITool* class, change the value of *state* attribute (Fig. 9) and trigger the sending of *changed* message (Fig. 10)).

Objects of the class *Stream* are not mentioned above in the list as they are not direct components of the presentation. They must be first 'wrapped up' into objects of *Beam* class. They only play the role of units of low level organisation of multimedia data. The same, each component of the multimedia presentation is perceived at the same time from two perspectives:

- as an atomic object – beam perspective, and
- as a temporal sequence of LDU – stream perspective.

5.1 Stream

Despite the fact that the *Stream* class does not directly influence the structure of presentation, it provides means for specification of intra-stream synchronisation, which are the base for definition of the inter-stream synchronisation.

The synchronisation mechanisms on all levels are based on the class hierarchy for the *SynchPoin* presented on Fig. 2, where the basic notion of reference point on the time axis is

defined. Data stored in the objects of this class are always interpreted relating to the time of the object which aggregates them. The objects of the class *StreamSynchPoint*, as it was already said, define a sequence of the LDUs in time. The values of the *optimal time* (*t-opt*) and *duration* of playing (*dur*) constitute specification of the intra-stream synchronisation – what is expressed on the diagram on Fig. 5.

After initialisation of the stream, during *preloading* state executed in response for a *load* signal from beam (Fig. 5), stream is waiting (*waiting* state) for asynchronous calls from the beam aggregating the given stream.

Each *play* message carries the information derived from *BeamSynchPoint* objects (Fig. 6):

- *sp* – a *StreamSynchPoint* object – pointing to the first LDU of a sub-sequence to be played,
- *play_dur* – a duration of the given sub-sequence to be played (*play_dur* is expressed in beam's time and must be translated to the stream's time).

In the *playing* compound state, subsequent LDUs are loaded and played for the time defined by *dur* attribute of the *StreamSynchPoint* objects corresponding to them. The abstract time *dur* is converted to the real time of the environment by the *calcRealTime* method of the *VirtualTimeAxis* object.

The real time of realisation of the *play* operation is remotely controlled by the *Beam* ob-

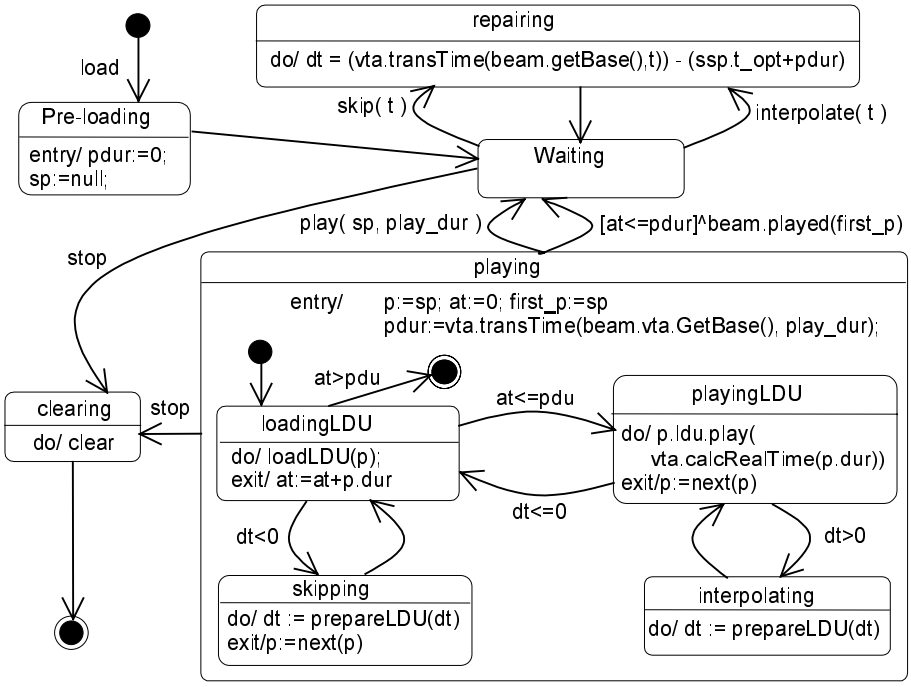

Fig. 5. Basic behaviour of stream.

ject aggregating the stream. The information about the possible errors of synchronisation (time attribute in *interpolate* and *skip* calls) can be sent to stream by the beam after playing of sub-sequence of LDUs. This process will be examined in details in the next subsection., The value of *dt*, time correction, is computed in the *repairing* state on the base of information sent by the beam. The value is negative when the playing of the stream is too fast.

The value of *dt* different than zero means an error in synchronisation, which has to be corrected either by speeding up or slowing down the presentation of the stream. This is done by the operations *skip*, *interpolate* and *duplicate* corresponding to standard operations for modification of the presentation speed [1], where:

- *skip* – causes skipping of the following LDU and, in consequence, speeding up the presentation;
- *duplicate* – duplicates playing the current LDU and in consequence causes slowing down the presentation (this operation is not allowed in some media – for instance in audio it would cause a considerable decrease of the quality of presentation);
- *interpolate* – generates some additional LDU to be played, interpolated from the closest LDU.

The *interpolate* operation is a generalisation of a *duplicate* operation and that is why only the *interpolate* has been included in the model. Both introduced operations, besides the simple elimination or duplication of LDUs, must change a number of other LDUs (*preparing* operation) which are proceeding and following the group of LDUs on which the operation is performed.

The process of loading of LDUs and the process of *preparing* new LDUs can be affected by *interpretQoS* operation. The operation can reduce the size of LDUs according to the requested level of *Quality of Service*, which is stored in *qos* attribute.

Quitting the *playing* state triggers sending *played* signal to the *Beam*. The signal informs the beam that the inter-stream synchronisation point was successful achieved (see next subsection and Fig. 8).

The mechanisms of composition and synchronisation included in stream fully correspond to the Media Layer in SRM model. However, they are also the base for construction of higher levels of synchronisation.

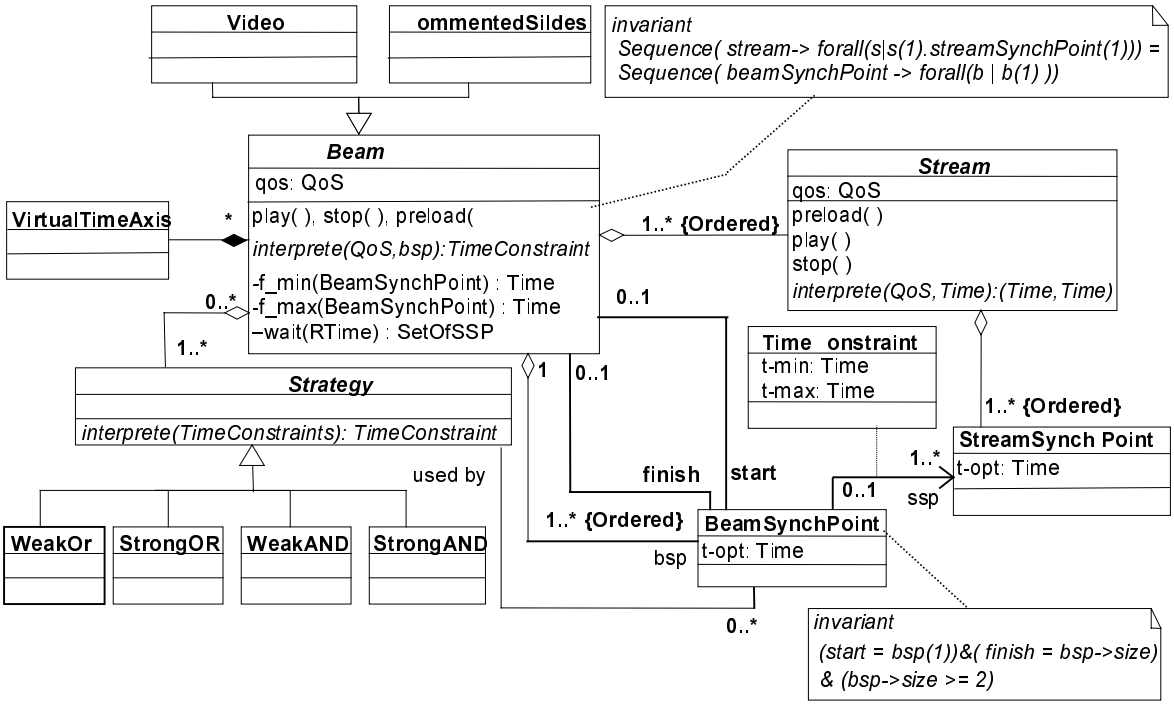

Fig.6. Structure of Beam.

5.2. Beam

Beam class represents an ordered collection of streams together with the specification of inter-stream synchronisation between the streams. Beam allows modelling of both simple as well as compound multimedia data. Simple multimedia type is modelled by a beam consisting of only one stream. 'Wrapping up' of a single stream is necessary because of inter-stream synchronisation mechanisms and different tasks performed by both classes:

- organisation of multimedia data on the low level – in the case of the stream,
- definition of those properties of the multimedia object which allow for synchronisation (and co-operation) with other multimedia objects.

The mechanisms of inter-synchronisation are based on objects of the class *BeamSynch-Point*. Every such object is connected with a number of *StreamSynchPoint* objects [9]. *BeamSynchPoint* objects specify points in time (*t-opt*, interpreted by time axis of the *Beam*) for synchronisation of a number of streams. The first point in every *StreamSynchPoint* must be tied to the appropriate *BeamSynchPoint* point (see invariant on Fig. 6). In that way, the moment of starting of each stream in a given beam is specified. However, there is no need for them to start at the same time. The collection of *BeamSynchPoint* objects is ordered and

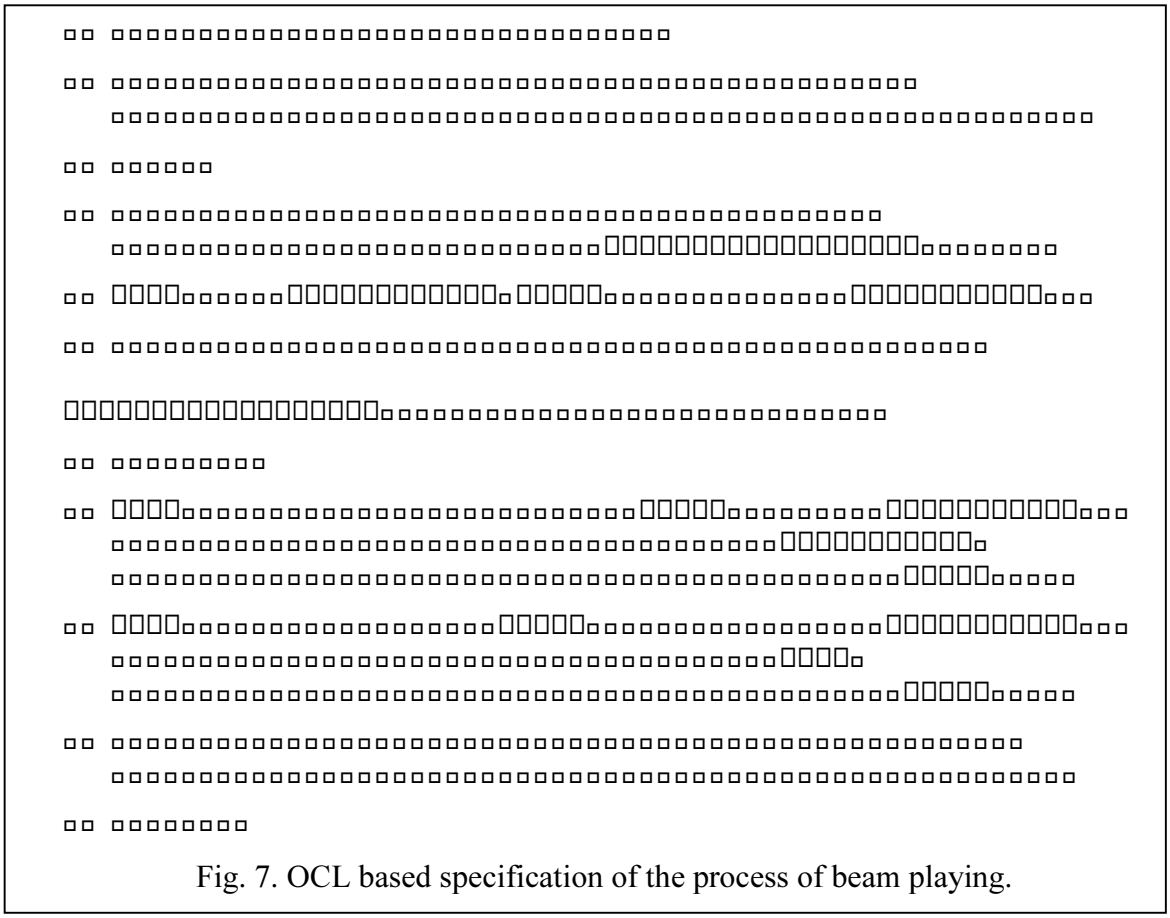

Fig. 7. OCL based specification of the process of beam playing.

defines the process of beam presentation. Thus, each *BeamSynchPoint* object (together with its neighbour objects of the same class) defines three temporal constrains:

- *a starting point* for playing of some sub-sequences of LDUs of the given streams (the streams associated with the *StreamSynchPoint* objects connected to the given *Beam-SynchPoint* object; each *StreamSynchPoint* object identifies some point in a certain stream);
- *parts of the streams* to be played, where duration is equal to t_opt_2 (i.e. t_opt of the next *BeamSynchPoint* object) minus t_opt_1 (i.e. t_opt of the given *BeamSynchPoint* object); the duration is sent to each stream object with play message (Fig. 5), and next is translated to stream's units and interpreted according to the appropriate starting point; the duration expresses the expected time – the final time of playing can be different;
- *an optimal time* in which playing of all parts of streams triggered by the preceding *BeamSynchPoint* object should be finished.

There are two distinguished *BeamSynchPoint* objects: the *start* object(association on Fig. 6), which defines the starting point of the beam, and the *finish* object, which defines the end of the presentations of the beam i.e. it does not trigger playing of any part of any stream. *BeamSynchPoint* objects allow reorganising the order in which LDUs from the given streams are played and can express both: regular types of synchronisation (i.e. lips and pointer synchronisation) and the editorial decisions made on the composition of the

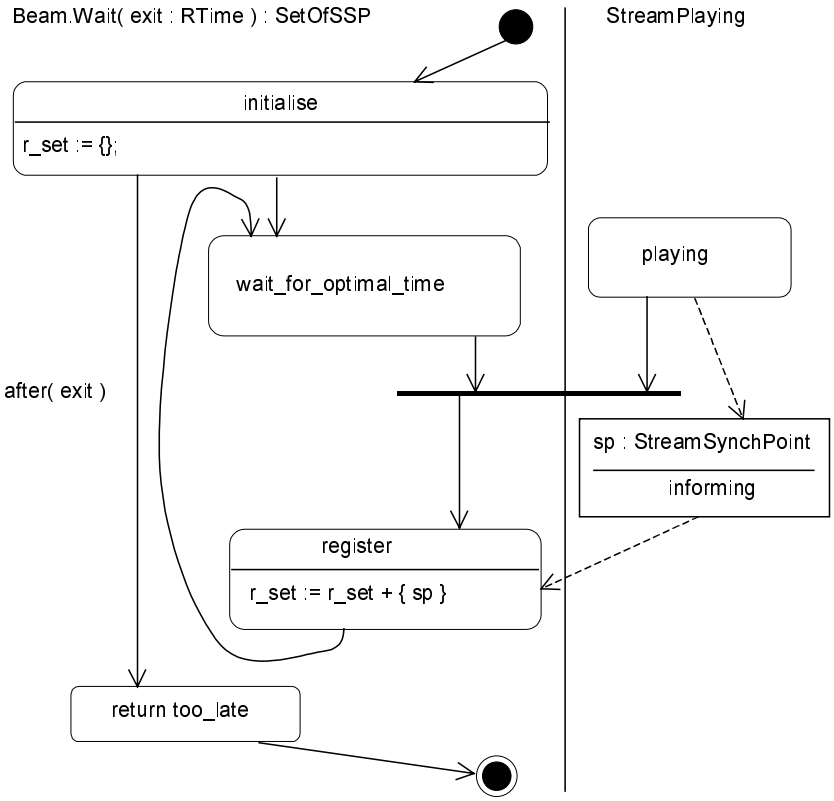

Fig.8. Beam behaviour: the algorithm of *Wait* operation

compound medium (e.g. video with subtitles). Apart from specifying inter-stream specification, objects of the class *BeamSynchPoint* play also a role of a source for events available during presentation.

Additional time constraints are expressed by the *TimeConstraint* objects, which specify constraints imposed on possible errors in playing data by the streams (each stream confirms finishing of playing by sending a *played* message Fig. 5). The final interval for the given *BeamSynchPoint* objects, in which the beam should receive messages from the streams, is calculated by the *interpret* operation defined in the *Strategy* class. The method is based on the extended Timed Petri Nets approaches proposed in [3], and generalised in [12] (i.e. a strategy operation from a set of pairs computes interval used as the final time constrains for the given point). In addition to this, the time constrains of synchronisation are controlled (weaken or sharpen) by the time interval obtained from interpretation of the attribute (*Quality of Service*) by the *interpret* method. The interpretation of *qos* depends on the types of the streams joined in the given *BeamSynchPoint* object [1], and can vary from point to point. The paper [1] presents results of experiments on the eligible values for QoS. *Interpret* is an abstract method and should be adjusted according to the type of streams forming the given beam. The value of the skew is evaluated in two steps – by the strategy and by the interpretation of QoS. In case of conflict, sharper conditions are chosen.

The process of beam playing, together with operations specific for inter-stream synchronisation, is modelled on Fig. 7 and Fig. 8.

The specification shown on Fig. 8 presents an algorithm expressed using basic OCL [13] operations and can be directly converted to an activity diagram, treating the numbered points as specifications of activities.

In the first step beam activates all streams sending to them *load* signal (see also Fig. 5). Next, *start* - the distinguished *BeamSynchPoint* object - points to the parts of streams which should be played first. The duration of playing is counted as a difference between the *t-opt* attribute of *start* and the *t-opt* of the next *BeamSynchPoint* object in the order of all *BeamSynchPoint* objects. The existence of at least two *BeamSynchPoint* objects (i.e. *start* and *finish*) is ensured by the invariant attached to *BeamSynchPoint* in Fig. 6.

Then, each *BeamSynchPoint* object, located in the sequence between *start* and *finish*, controls inter-stream synchronisation of playing of the streams which was started by the preceding point (internal *wait* operation is modelled by a separate activity diagram Fig. 8). And next, it starts playing a next group of parts of the streams at determined positions within them. The first call of *wait* operation (Fig. 8) collects *played* signals coming from the streams before the time returned by the internal operation *f-min* for the given *BeamSynchPoint* object. The time returned by *f_min* is the lower bound for synchronisation at the given *BeamSynchPoint* object, computed first by the application of the *interpret* operation from *Strategy* object and next the *interpret* operation of the beam (as was said earlier the stricter condition is chosen). The set of *StreamSynchPoints* returned by the first call of *wait* identifies the streams which finished playing the requested part too early. In order to repair the broken synchronisation between streams, an operation *interpolate* ('slowing down' op-

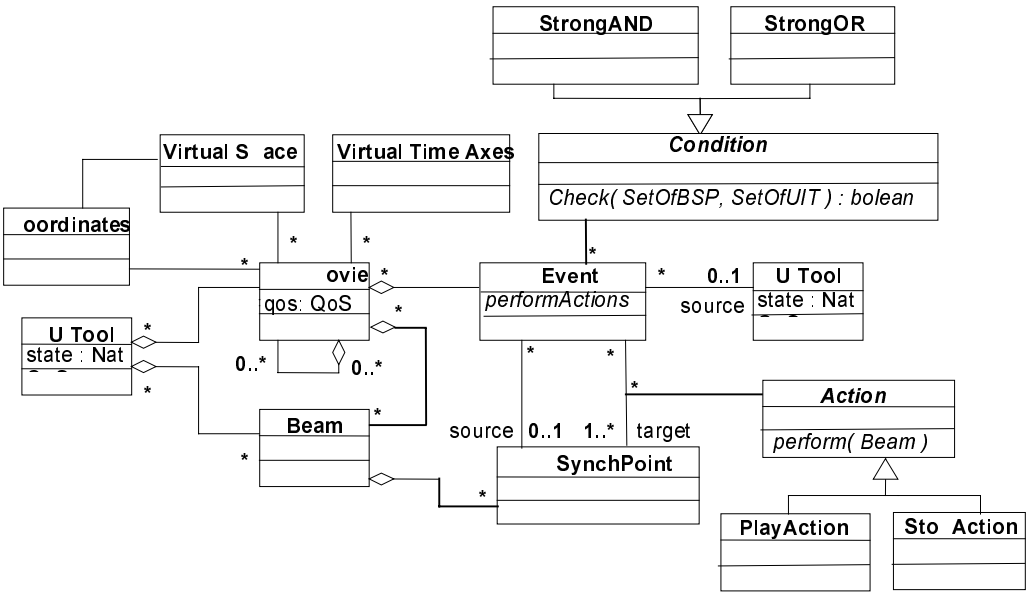

Fig.9. Multimedia Movie

eration) is called for each stream in the set. In a similar way, for the streams returned as the result of the second call of *wait*, *skip* operation is called. The last task to do for each *BeamSynchPoint* object, except *finish* object, is to start playing the next group of parts of the streams.

Finally, after waiting for the time specified in *finish BeamSynchPoint* object, all streams are stopped.

The mechanisms of inter-stream synchronisation presented in the paper, models 'hard manner' [11] of synchronisation where the values of time conditions are explicitly specified. However, beam is assumed to be a representation of compound media types with built in time constrains. The more 'flexible manner' [11] of synchronisation is modelled on the higher level of the whole multimedia presentation (i.e. *movie* level) by event synchronisation.

5.3. Multimedia Movie

Multimedia *movie* is an element of the highest level, modelling complete multimedia presentation. *MMMovie* class provides the mechanisms for event synchronisation which bind *events* coming from different *Beam* objects, as well as from specific user interface tools (*UI Tool* class). Event is understood here as achievement of some inter-stream synchronisation point within a beam, and as a change in state of the user tool caused by the user.

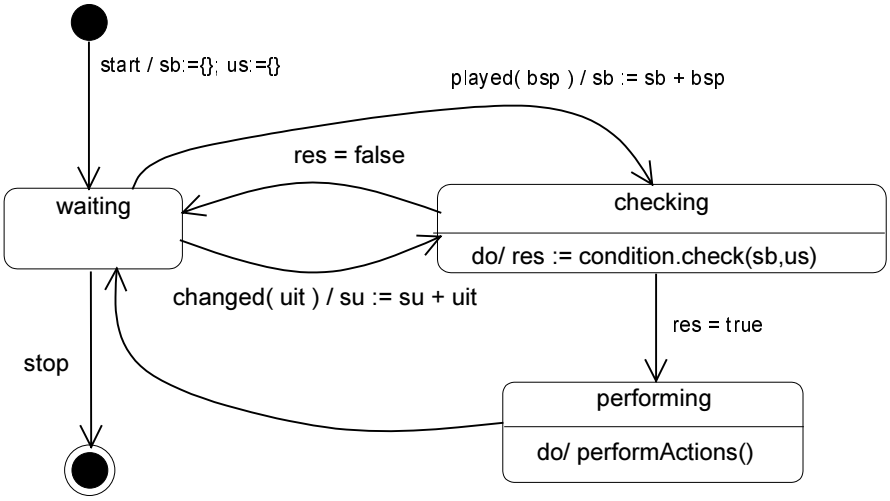

Fig.10. Behaviour of Event

Specification of event synchronisation is expressed by *Event* objects, which define:

- a set of event sources (*source* association) and target objects (*targets* association),
- logical condition defined by the object of *Condition* class, and
- action for each target object defined by the object of *Action* class (attribute of the association).

Both *Condition* class and *Action* class are abstract classes and must be specialised by appropriate derived classes.

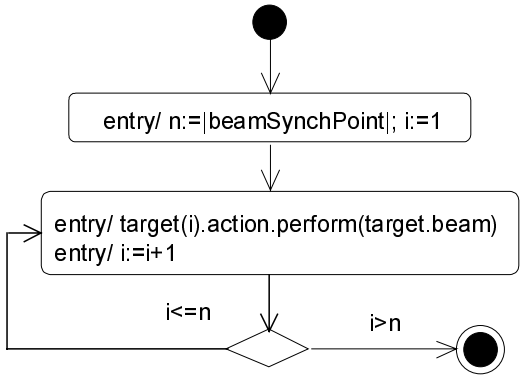

Fig.11. Algorithm of *performActions* operation

The mechanism of event synchronisation defined by *Event* class is modelled on activity diagrams specifying the behaviour of *Event* objects: Fig. 10 and Fig 11.

Occurrences of events, which are manifested as signals from appropriate source objects, are collected in two sets: *sb* – a set of *BeamSynchPoint* objects, modelling events happening in beams, and *su* - a set of *UITool* objects, modelling events happening in interaction tools.

Each signal coming to *Event* object is added to the appropriate set and *check* operation of the *Condition* object is called with both sets as arguments. The condition can be as simple as *strong or* [3] - checking for occurrence of at least one signal, *strong and* [3] - checking for occurrence of all one signal or much more complicated e.g. depending on exact state of some *UI Tool* objects.

Depending on the result of *check* operation *performActions* is called (modelled on the activity diagram in Fig. 11). This operation in turn calls a *perform* operation of an appropriate object of *Action* derived class for each *source*. The necessary operation of beam can be called through different versions of *perform* operation in classes derived from *Action* class.

The mechanism of event synchronisation, as modelled here, represents a 'flexible manner' of synchronisation [11]. There is no need to specify exact time conditions (but such future extension of the model is also possible: the *Event* class, similarly to the *BeamSynchPoint* class, is derived from *MMSynchPoint* class). However, the event synchronisation is strongly and explicitly connected by *BeamSynchPoint* objects to lower levels of synchronisation and it is built on top of them. In that way, the model presented in the paper covers all levels of SRM model [1].

6. Conclusion

In the paper an attempt to formulate a conceptual basis for the formal description of the multimedia presentation was undertaken. UML, chosen as the specification language, is not as formal as algebraic specifications or Petri nets, but is offering greater expressive power allowing to create more structured and multilevel description of both multimedia data and multimedia presentation. To present the idea of the usage of UML for this purpose, a minimal set of notions such as logical data units, synchronisation point, stream, beam and movie, has been chosen and on that base, a model explaining several static and dynamic aspects of multimedia presentation was constructed.

The model reveals origins of different classifications of multimedia data being in use. An interesting property of the model, not present in other model (e.g. MHEG [5]), is that it allows perceiving multimedia data from different perspective at the same time.

The ideas used in the constructions of the model can also be considered as domain specific extension of the UML to include multimedia-oriented constructs using standard UML extension mechanisms, e.g. specialised, formally defined stereotypes. The model of multimedia presentation can be used as a starting point for defining a set of fully developed design patterns (in the style of [7]) to be used in applications using multimedia data. Finally models presented can be a base for construction of an authoring system and can be a starting point for development of object-oriented graphical language of specification of multimedia presentations.

References

[1] G. Blakowski and R. Steinmetz, A Media Synchronisation Survey: Reference Model, Specification and Case Studies, *IEEE Journal on Selected Areas in Communications*, vol.**14**, no. **1**, (Jan. 1996).

[2] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modelling Language Reference Manual, Addison Wesley, 1999.

[3] M. Diaz, C.A. Senac, Time Stream Petri Nets, A Model for Timed Multimedia Information. In: Proceedings of International Conference on Application and Theory of Petri Nets, Zaragoza, June 1994.

[4] B. P. Douglass, Real Time UML, Addison-Wesley, 1999

[5] M. Echiffre *et al*, MHEG-5 Aims, Concepts and Implementation Issues, *IEEE Multimedia*, (Jan-Feb, 1998)

[6] B. Furth and R. Westwater, Video Presentation and Compression. In: F. Borko (ed.) Handbook of Multimedia Computing, CRC Press, 1999, pp. 171-204.
[7] E. Gamma *et al*., Design Patterns: Elements Of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[8] S. J. Gibbs and D. C. Tsichritzis, Multimedia Programming, Addison-Wesley, 1995
[9] L. Kuźniarz and M. Piasecki, An Abstract Model for Temporal Composition of Multimedia Data, In: Proceedings of the 23th Euromicro Conference, Budapest 1997
[10] L. Kuźniarz and M. Piasecki, Towards Complex Object Oriented Analysis and Design, In: H.J. Bullinger and J. Ziegler (editors) Human-Computer Interaction, Lawrence Erlbaum Associates Pub., London, 1999, pp. 1257-1261.
[11] B. Prabhakaran, Multimedia Synchronization, In: F. Borko (ed.) Handbook of Multimedia Computing, CRC Press, 1999, pp. 525-544.
[12] Z. Spławski, Synchronisation Mechanisms for Multimedia Streams and Their Specification in Timed LOTOS, In: Proceedings of the 23th Euromicro Conference, Budapest 1997.
[13] Unified Modeling Language Specification: UML Semantics v 1.3., Object Management Group. [http://www.omg.org/techprocess/meetings/schedule/UML_RTF.html], 1999.
[14] R. Westwater, Digital Audio Presentation and Compression, In: F. Borko (ed.) Handbook of Multimedia Computing, CRC Press, 1999, pp. 135-147.