INTERNATIONAL STANDARD

ISO/IEC 3721

First edition
2023-09

Information technology — Computer graphics, image processing and environmental data representation —Information model for mixed and augmented reality content — Core objects and attributes

Reference number
ISO/IEC 3721:2023(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 24, *Computer graphics, image processing and environmental data representation*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Mixed and Augmented Reality (MAR) refers to a spatially coordinated combination of media/ information components that represent, on the one hand the physical real world and its objects and on the other, those that are virtual, synthetic and computer generated. MAR, as an information medium, strives to provide rich experience based on realism, presence and augmentation.

In this document, a comprehensive set of information constructs for representing mixed and augmented reality (MAR) contents is described. This set of components extends the conventional ones used for representing virtual reality (VR) contents, as MAR environments are technically realized as virtual environments. The principles and requirements for the extension are laid out and the details of the component model including (but not limited to) those for representing *physical* real world objects, extending the virtual scene graph/structure to that for MAR (with the physical objects), how to spatially the physical objects into the MAR scene graph, associating these content components to the MAR system, and other miscellaneous constructs (e.g. event mapping, MAR events/ behaviours, video backdrops, etc.). This document is designed for the ease, generality and extendibility, and this is demonstrated with various examples and implementation results. The model will serve as a sound basis for establishing standard and interoperable file formats MAR contents in the future.

The document also provides definitions for terms as related to these MAR content informational components and their attributes.

The target audience of this document are mainly MAR system developers and contents designers interested in specifying MAR contents to be played by an MAR system or browser. The standard will provide a basis for further application standards or file formats for any virtual and mixed reality applications and content representation.

The extension will be self-contained in the sense that it is independent from the existing virtual reality information constructs, focusing only on the mixed and augmented reality aspects.

However, this document only specifies the information model, and neither promotes nor mandate to use a specific language, file format, algorithm, device, implementation method, and standard. The standard model is to be considered as the minimal basic model that can be extended for other purposed in actual implementation,

This document is based on the MAR Reference model (ISO/IEC 18039) that specifies for the contents-browser/player type reference architecture. The MAR content (in ISO/IEC 18039) is specified as the input that describes the scene and objects' behaviours, given to the browser/player which in turn parses, simulates and renders it to the display. The standard is the information model for the content.

As an extension to the virtual reality based contents or scene structure, this standard is very much related to the existing standard for VR scene representation such as ISO/IEC 19775-1 (X3D) and other related on-going standards such as the image-based object/environment representation for VR/MAR (ISO/IEC 23488) as well. There are also specific object models relevant to this standard such as those for the live actors and entities (ISO/IEC 18040 and ISO/IEC 23490) and MAR system sensor components (ISO/IEC 18038).

Information technology — Computer graphics, image processing and environmental data representation — Information model for mixed and augmented reality content — Core objects and attributes

1 Scope

This document specifies the information model for representing the mixed and augmented reality (MAR) scene/contents description, namely, information constructs for:

a) representing the virtual reality scene graph and structure such that a comprehensive range of mixed and augmented reality contents can also be represented;

b) representing physical objects in the mixed and augmented reality scene targeted for augmentation;

c) representing physical objects as augmentation to other (virtual or physical) objects in the mixed and augmented reality scene;

d) providing ways to spatially associate aforementioned physical objects with the corresponding target objects (virtual or physical) in the mixed and augmented reality scene;

e) providing other necessary functionalities and abstractions that will support the dynamic MAR scene description such as event/data mapping, and dynamic augmentation behaviours;

f) describing the association between these constructs and the MAR system which is responsible for taking and interpreting this information model and rendering/presenting it out through the MAR display device.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 18039, *Information technology — Computer graphics, image processing and environmental data representation — Mixed and augmented reality (MAR) reference model*

ISO/IEC 18040, *Information technology — Computer graphics, image processing and environmental data representation — Live actor and entity representation in mixed and augmented reality (MAR)*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in from ISO/IEC 18039 and ISO/IEC 18040 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1.1
aggregation
relation among objects/components that specifies a whole-part relationship as defined and used in the Unified Modelling Language literature

3.1.2
association
relation among objects/components (and their instances) that they can be linked to each other or combined logically or physically into an aggregation

Note 1 to entry: An aggregation is a specific form of an association as defined and used in the Unified Modelling Language literature.

3.1.3
behaviour
object or *component* that describes how certain objects and their attribute values change in time and/or in response to *events*

3.1.4
content
MAR content
MAR scene
description of mixed and augmented reality based experience to end-users by publishers or media producers

3.1.5
component
MAR component
object
MAR object
self-contained computational entity or model that has one or more input channels and/or one or more output channels

Note 1 to entry: In the context of MAR contents or system, a component has a relevant MAR functionality.

3.1.6
component model
object model
model of a collection of computational *objects* and their relationships serving a particular purpose or collective function

3.1.7
event mapping
object or *component* that maps events produced by the *MAR system* and maps/relates them to the corresponding ones in the *MAR scene/content* space

3.1.8
glass-based
type of an optical see-through device that has the form factor of optical glasses

3.1.9
inheritance
association representing a parent and child relationship

Note 1 to entry: In general, the child object, if not specified otherwise, will inherit the attribute model of the parent as defined and used in the Unified Modelling Language literature.

3.1.10
node
term referred to an actual implementation or specific format of an *object* or *component*

3.1.11
optical see-through device
apparatus or device that allows the *real/physical world* to be seen directly or through optical elements. in addition, the device allows an overlaid display of graphical (or *virtual*) elements

3.1.12
real transform group
RTG
transform group representing a group of objects situated at a *physical/real* spatial location

3.1.13
transform
abstract entity representing a relative spatial relationship (translation, rotation and scaling) with another coordinate system (or equally *transform*)

3.1.14
transform group
TG
component or object that represents a group of *objects* or *components* sharing a common physical or virtual coordinate system

3.1.15
video see-through device
apparatus or device that displays the real/physical world live through a video camera feed

3.1.16
virtual transform group
VTG
transform group representing a group of *objects* situated at a virtual spatial location

3.2 Abbreviated terms

AR	augmented reality
AV	augmented virtuality
CRT	capturer, recognizer, tracker
GNSS	global navigation satellite system
LAE	live actor and entity
MAR	mixed and augmented reality
MAR-RM	mixed and augmented reality reference model
MR	mixed reality
UI	user interface
VR	virtual reality
RTG	real transform group
TG	transform group
VTG	virtual transform group

4 Overview

MAR refers to the interactive medium that uses and merges the real (or physical) and virtual objects.[6] The two representative genres of MAR are augmented reality (AR) where virtual objects are added on to representation of the real physical world, while in augmented virtuality (AV) real world physical object representations are added to the virtual environment. The continued innovations and advances in computer vision, mobile/cloud computing and portable display devices have brought about a renewed interest in MAR, as a prominent information visualization and interaction medium. MAR not only has many application areas but also can empower users in daily activities by spatially augmenting useful information to key interaction objects. Traditionally, MAR services/contents have been developed as a single application using popular application programming interfaces (APIs). While there are moves to standardize the API[7,8],there is also a strong and natural move toward the separation of such singleton applications into contents (in some standard formats) and dedicated players/browsers, similarly to those for web documents.

X3D already offers a standard and declarative method of specifying dynamic virtual environments (ISO/IEC 19775-1) and is in fact being extended for MAR functionalities[9]. Such a service structure (and standards) can promote the proliferation of the given media and associated content form. Thus, in this document, a minimal (yet able to cover a comprehensive and reasonable, typical MAR contents) set of information constructs for representing MAR contents is established. The standard would extend the conventional constructs used for representing virtual reality contents (such as X3D), as virtual environments are used for the implementation platform for mixed reality contents as well.

First, the principles and requirements for the extension are laid out and the details of the component model including (but not limited to) those for representing real world physical objects, extending the virtual scene graph/structure to that for MAR (with the real objects), how to spatially the real objects into the MAR scene graph, associating these content components to the MAR system, and other miscellaneous constructs (e.g. event mapping, MAR events/ behaviours, video backdrops, etc.). The standard is designed for the ease, generality and extendibility, and this is demonstrated with various examples and implementation results. The model will serve as a sound basis for establishing standard and interoperable file formats MAR contents in the future.

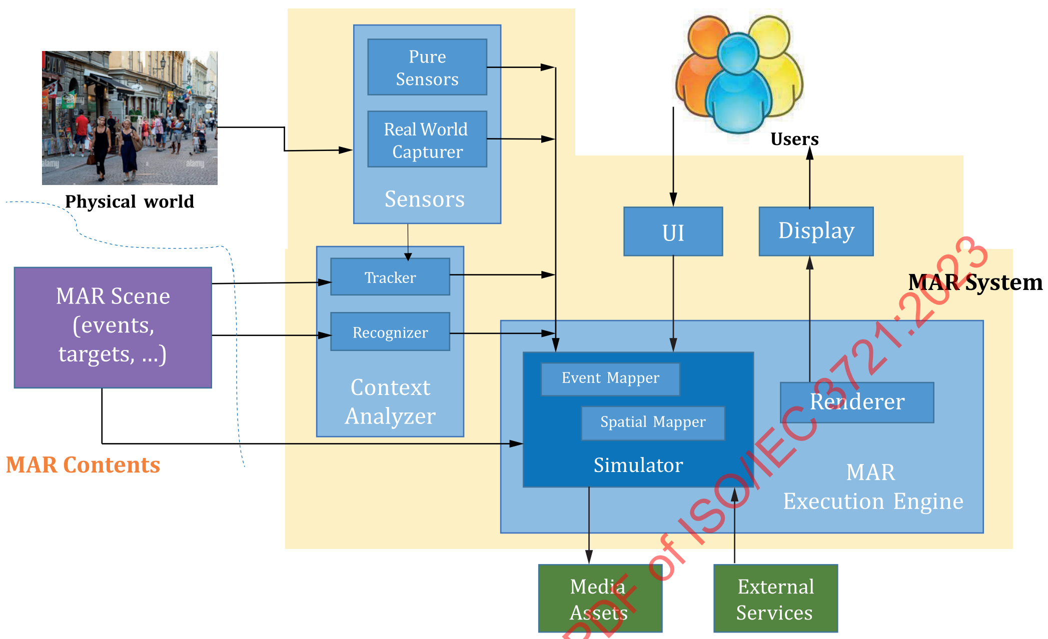
Most MAR systems have been implemented as a single application with all the contents and assets embedded in it, using programming libraries or APIs. The GPS equipped mobile and smart phones have made ways for location based augmented reality services (e.g. providing guides for commercial points of interest and tourism). Such a service necessitated the separation of contents (and its format specification) and the underlying player (to support the notion of one place-many contents). Future interoperability, reusability and proliferation of MAR contents will hinge on the comprehensible and extendable content model (and file format). For example, the file format for location based AR service called the ARML (Augmented Reality Mark-up Language) has been adopted as a standard for the Open Geospatial Consortium. ARML allows defining geographical points or landmarks of interest and associate GPS coordinates and simple augmentation contents (e.g. text, logos, and image).

The Web3D consortium has also been considering adding new components for MAR functionality to X3D, the ISO standard and declarative mark-up file format for representing virtual environments. The group has specified new and extended nodes to support e.g. video see-through based AR, such as the live video background, extended camera sensor nodes[9]. Some of its work is reflected in this standard as well. As the video augmented content is a popular form of MAR, the MPEG has proposed the ARAF (AR Application Format) that uses other MPEG standards (like MPEG-4 and MPEG-V) to specify video based and spatially augmented contents (ISO/IEC 23000-13).

Another class of approach of realizing, extending and specifying for MAR functionality and behaviour is through scripting. InstantReality[10] has developed its own extension to X3D and modules (e.g. for marker and image patch tracking allowing to express MAR contents with scripts through which the virtual objects (as expressed by the usual X3D formalism) can be associated with the position/pose tracked real world physical objects. There are several similar web-based MAR systems that can present AR contents with Javascript1) programming[11-13]. Despite these approaches, the current state with

1) JavaScript is a registered trademark of Oracle Corporation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC.

respect to MAR content representation is still limited in its comprehensiveness (e.g. only covers certain genre of MAR such as location based or video based), often implementation dependent and proprietary, and lacks sufficient abstraction and clean modularization requiring lengthy and complicated script programming.



The MAR Scene or equivalently content description (purple box in the left part of the figure) is an input to the larger MAR system to be interpreted by it and be rendered to a display for the user consumption.

Figure 1 — The generic MAR system architecture as specified in the ISO/IEC 18039 MAR reference model

5 Principles and Requirements

The MAR reference model (ISO/IEC 18039) suggests the scope for which a general MAR system and content shall encompass as shown in Figure 1. Accordingly, the MAR content model should provide reasonable generality, and be able to express both AR and AV, the two notable genres along the MAR continuum under a consistent and unified representational framework. Other requirements are as follows:

— independent of particular sensor/device model, algorithm or implementation platform (such issues would be absorbed into the browser/player implementation),

— provide virtually any digital information and media, both static and dynamic, as augmentation such as text, images, videos, animation, HTML document elements, etc.,

— provide useful abstract and declarative constructs for often used content functionalities and minimize manual scripting or programming,

— make use of existing standard constructs, where possible,

— flexible/extendible to accommodate new future requirements.

To fulfil the final requirement (and others indirectly), this standard will use the component based approach, similarly to that of the X3D. Components in X3D are collections of objects that perform or represent similar operations, displays, or functions. Each component may define multiple levels. The levels indicate the complexity of the object or object features. The component framework is ideal for introducing and modelling new capabilities that satisfy MAR requirements (regardless of using X3D or not) – new objects can be added or expanded as needed. The components form a structured environment that ensures consistent behaviour, usability and generality. The MAR reference model (ISO/IEC 18039) outlines the model architecture of a prototypical MAR system and indirectly points to the informational needs of an MAR content as an input to the system. In the next clause is described, the kind of new or expanded MAR components (equivalently functionalities) to be abstracted and represented.

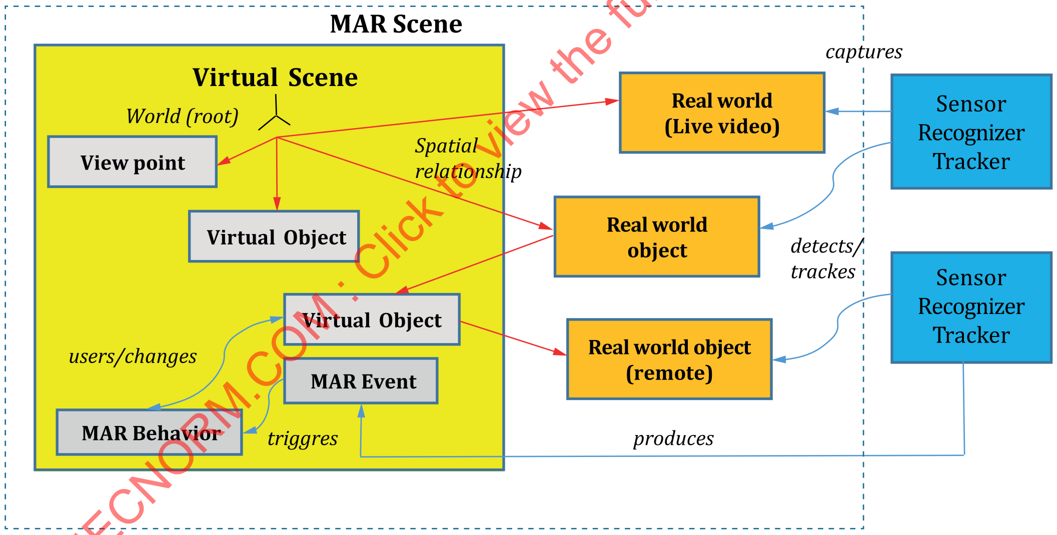
Again it is emphasized that the MAR content is the input to the system (purple box in the left part of Figure 1) which comprises the entities such as the sensor, capturer, tracker, recognizer, simulator, renderer, etc. Thus, while the content model may be associated with such system components, the system components themselves would not be part of the content description or model.

Note that the extension will be self-contained in the sense that it is independent from the existing virtual reality information constructs, focusing only on the mixed and augmented reality aspects.

6 MAR content model

6.1 Concept

MAR contents are technically realized as virtual reality contents with its scene containing special type objects that represent the real physical world/objects. In other words, one can think of a MAR scene that is technically a virtual scene with placeholders for the real world physical objects. The placeholders may be defined logically or spatially with respect to the virtual scene. These so called special type placeholder objects may need additional information of how it might be spatially registered into the implementation virtual scene. It is important to note that such a representation is (and should be) amenable to representing both AR and AV contents in a unified fashion.



Real world objects (in yellow) are associated with the virtual environment scene graph (in orange), and in turn the real objects are sensed, tracked, recognized and captured by the MAR system (in blue).

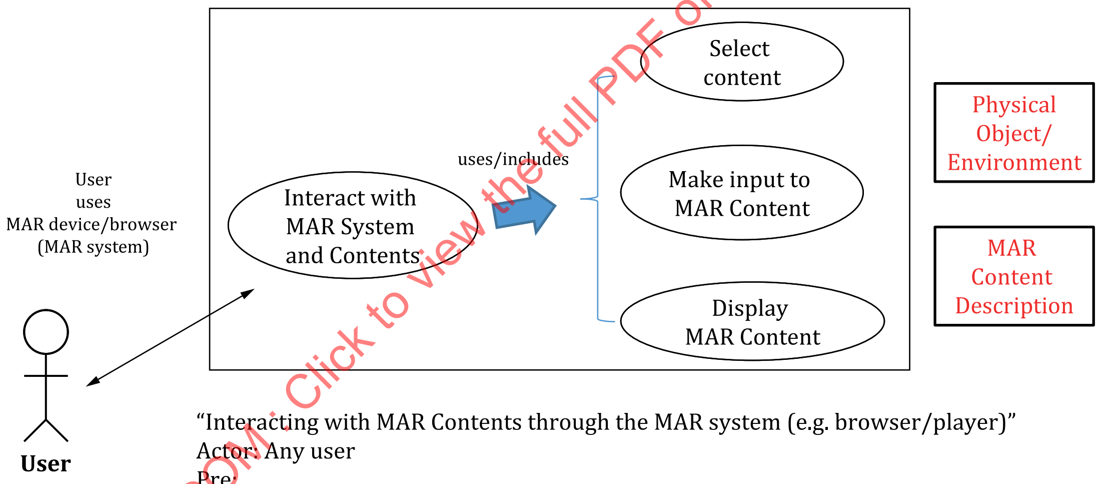
Figure 2 — The concept of extending virtual reality scene structure for MAR contents

Figure 2 illustrates the basic concept in which virtual objects (or the root of the virtual scene) provide a place for which certain real world physical object (remote or local) would map to in the MAR scene. This way, the real world physical objects are seamlessly represented within the virtual scene structure. For example, a virtual object can be spatially defined with respect to a real world physical object (or its placeholder). Note that the virtual scene may not necessarily be 3D (i.e. could be augmentation onto the 2D content), and real world physical objects can be individually recognized and tracked, or captured as a whole (e.g. live video) and implanted into the virtual scene. This is accomplished and specified by association with the MAR System components (e.g. blue boxes in Figure 2). The whole content structure is input to the MAR Simulation Engine (another MAR System component) to be interpreted and ultimately rendered and displayed to the user. The red arrows in the figure indicate spatial relationships among the captured/tracked real world physical object with respect to the virtual scene/objects. Such a scheme is easy to define by simply extending the current scene structure for virtual environments.

In the following clauses, we describe the component based information model using the Unified Modelling Language (UML) like[15] notation, starting with the Use Case Diagram. The use of UML-like conventional is not meant to be followed strictly, but only to level sufficient for to the purpose of this document. Then the class-object diagram is described for the most important and essential ones. Several examples of how the class-object specification can be used to represent a variety of MAR content type are shown. Finally, the detailed component object specifications are given.

6.2 Use case diagram

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. Figure 3 shows a use case of a user interacting with the MAR system (e.g. a browser or player) that plays a user selected MAR content. The user selects a particular content and gives input, if necessary, to drive the content as rendered and displayed by the MAR system.



"Interacting with MAR Contents through the MAR system (e.g. browser/player)"
Actor: Any user
Pre:
- User has MAR device that is equipped with MAR browser
- MAR content description is given/selected by the user
Post:
- User sees through the device/browser augmented physical environment according to the MAR content description

Figure 3 — A use case of a user interacting with the MAR system (e.g. a browser or player) that plays a user selected MAR content

6.3 MAR content and its scene structure

Just like the virtual reality environment or contents, the scene graph or scene tree structure would be the most natural and suited representation with which the MAR content is represented as a hierarchical and spatial organization of various types of objects. While the objects are in principle organized by their spatial relationship ultimately with respect to the assumed root coordinate system, logical association and aggregation are possible too.

Thus, in this regard all objects, as part of the MAR scene graph, are subclasses of an abstract MARSNode object, specialized for whatever purposes they may have.

Aggregation relation is formed and used among objects to represent their collective spatial positioning and physical containment among the various objects in the scene. In particular, aggregation relation will be used to group various objects into a particular TG which also contains the common spatial information shared among the constituent objects (i.e. transformation – rotation, translation, scaling) with respect to another TG (or by default with respect to the assumed world root coordinate system). The TG aggregation also serves to represent the part-of relation in the scene hierarchy.

Association relations may exist among various objects that can refer and relate to each other for different purposes. The association may be one directional or two directional. For example, associations between content objects and system objects are typically one directional in the sense that the content elements seek/retrieve raw data as generated and mapped from the MAR system (i.e. sensors, capturer, tracker and recognizer).

6.4 Major MAR system objects that are associated with the MAR content objects[6]

MAR content is an input to the MAR System which interprets the content/scene description, simulates it and displays it to the user. Among others, the MAR system uses the sensors and other associated modules such as the tracker, recognizer and real world capturer (see Figure 1) to understand the real physical environment and extract various information in real time. Such information is used to associate the real world physical objects to the virtual and form the mixed reality scene. Sensors may be used simply to realize interactivity of the content as well. Therefore, while the MAR system objects are not part of the content itself, they have associations to the counterpart content objects. For example, the tracker may be associated to the spatial_mapper objects (in the content) that uses the tracking information to position the tracked real world physical object into the MAR scene.

In the specification of this standard, the detailed informational constructs for the non-content MAR System are excluded such as those for the sensor, real world capturer, tracker and recognizer. We assume that the constructs exist (in certain system implementation or according to some other standard). We also assume that the event/data types as produced by these MAR System objects and consumed by the MAR Content components (MARSNode) are already defined. The definition and functional description of the main MAR System objects that the content description are associated with are well described and given in the MAR reference model (ISO/IEC 18039).

6.4.1 Sensor

A sensor is a hardware (and optionally) software component able to measure specific physical properties. In the context of MAR, a sensor is used to detect, recognize and track the target physical object to be augmented. In this case, it is called a pure sensor.

A sensor can measure different physical properties, and interpret and convert these observations into digital signals. The input and output of the sensors are:

— input: real world signals;

— output: sensor observations with or without additional metadata (position, time, etc.).

In particular, the sensor output may feed into the content components. The sensor component is a physical device characterized by a set of capabilities and parameters. A prototypical object model is shown in Table 1. Many subclasses of the sensor can exist in the specialized but inherited form. A refined standard is being pursued through ISO/IEC 18038[15].

Table 1 — A prototypical object specification for Sensor object in the MAR System

Sensor		
Data type	**Attribute/Method name**	**Explanation**
string	id	unique identifier for reference
int	type	integer code indicating the particular sensor type

6.4.2 Real World Capturer::Sensor

Another use of the Sensor is to capture and stream to the content objects and the execution engine, as the data representation of the physical world or objects for composing a MAR scene. In such a case, it is called a Real World Capturer. A typical example is the video camera that captures the real world as a video to be used as a background in an augmented reality scene. Another example is augmented virtuality, where a person is filmed in the real world and the corresponding video is embedded into a virtual world.

Thus, one subclass of Sensor is the Real World Capturer whose output is an audio, video or haptics stream to be embedded in the MAR scene or analysed by specific hardware or software/content objects. A prototypical object might look like Table 2:

Table 2 — A prototypical object specification for Real World Capturer in the MAR System

Capturer		
Data type	**Attribute/Method name**	**Explanation**
Any*	rawData	sensed data
bool	enabled	Boolean value indicating whether to enable this Capturer
Capturer*	Capturer()	constructor
void	filter()	function filtering or post-processing the sensed and captured data
Any*	getData()	access function for the captured data

6.4.3 Tracker::Sensor

Another two subclasses of the Sensor are the Tracker and the Recognizer. The two are also collectively called the Context Analyser as well. The Context refers to the condition for which the augmentation should occur upon its target object. The Tracker and Recognizer provides information regarding this context.

The Tracker is a hardware or software object that analyses signals from the real world and detects and measure changes of the properties of the target signals (e.g. pose, orientation, volume).

The input and output of the tracker are:

— input: raw or processed signals representing the physical world and target object specification data (reference target to be recognized);

— output: instantaneous values of the characteristics (pose, orientation, volume, etc.) of the recognized target signals and event data.

A prototypical component might look like Table 3:

Table 3 — A prototypical object specification for Tracker object in the MAR System

Tracker		
Data type	**Attribute/Method name**	**Explanation**
Any*	rawData	sensed data (depends on the sensor type)
bool	enabled	boolean value indicating whether to enable this Tracker
Tracker*	Tracker()	constructor
void	filter()	function filtering or post-processing the sensed and tracked data
Any*	getData()	access function for the captured data

6.4.4 Recognizer::Sensor

The Recognizer is a hardware or software object that analyses signals from the real world and produces MAR events and data (Table 4) by comparing with a local or remote target signal (i.e. target for augmentation).

Recognition can only be based on prior captured target signals. Both the recognizer and tracker can be configured with a set of target signals provided by or stored in an outside resource (e.g. third party database server) in a consistent manner with the scene definition, or by the MAR scene/content description itself.

The input and output of the recognizer are:

— input: raw or processed signals representing the physical world (provided by sensors) and target object specification data (reference target to be recognized);

— output: at least one event acknowledging the recognition.

Table 4 — A prototypical object specification for Recognizer object in the MAR System

Recognizer		
Data type	**Attribute/Method name**	**Explanation**
Any*	target	target object template information to be recognized
Any*	rawData	recognized event (depends on the Recognizer type)
bool	enabled	boolean value indicating whether to enable this Recognizer
Tracker*	Tracker()	constructor
void	filter()	function filtering or post-processing the sensed and recognized data
Any*	getData()	access function for the recognized data

7 MAR content classes

7.1 Overall class/object structure

Figures 4, 5, 6 show the overall class diagram as the gross information model for MAR contents. Figure 4 diagram shows the inheritance relationship among the classes as part of the MAR scene structure nodes (see 7.1). The Figures 5 and 6 show the association among the classes, defining the MAR scene structure itself. Explanations of each class description (this clause) follow and the detailed individual class specifications are given in Clause 8. To be clear, objects are instantiations of the classes (or subclasses). Some classes will be abstract and have subclasses (or lower level classes) from which objects can be instantiated for actual deployment in usage.

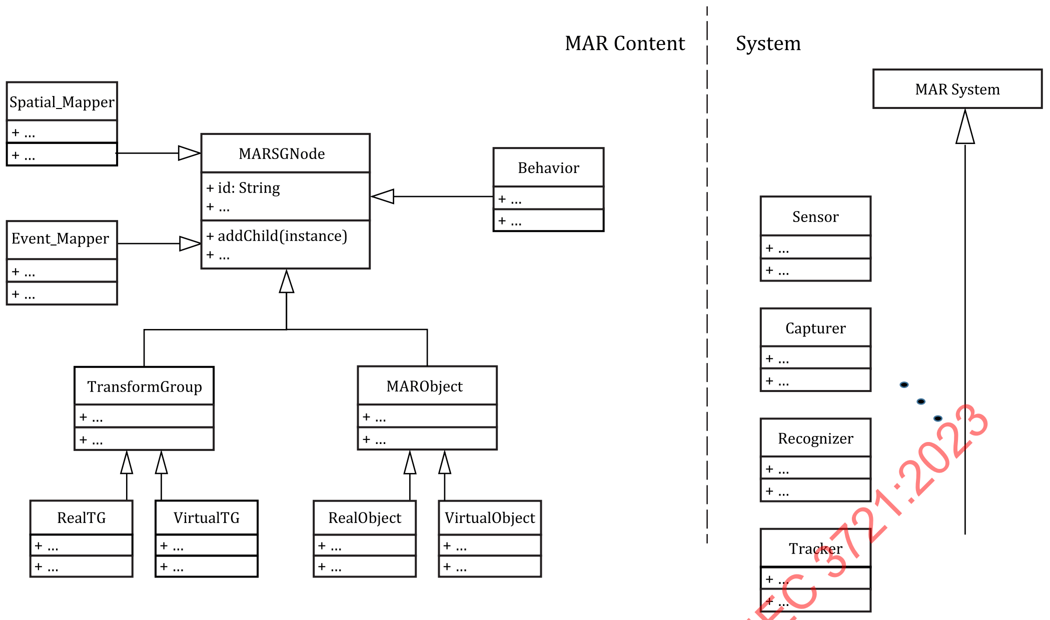
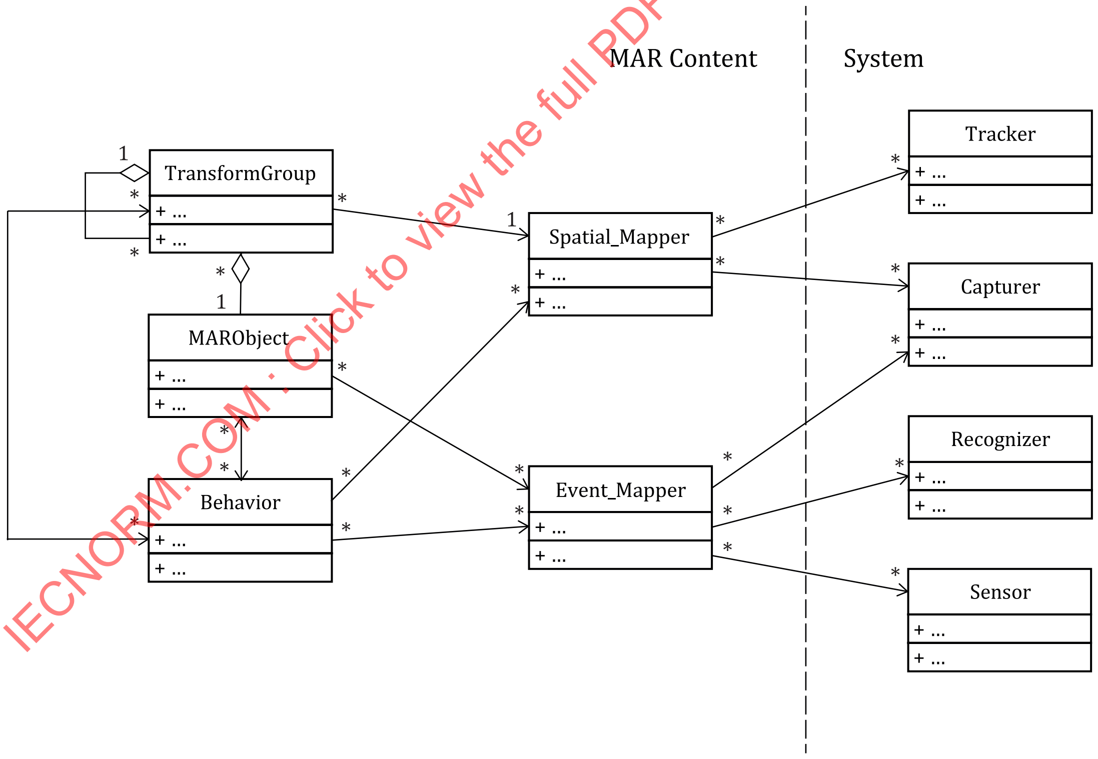
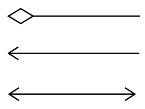


Figure 4 — Inheritance relationship (using the arrow head links) among classes in the MAR scene structure



Key



aggregation

association

Figure 5 — Association and aggregation relationship among classes in the MAR scene structure (simplified).

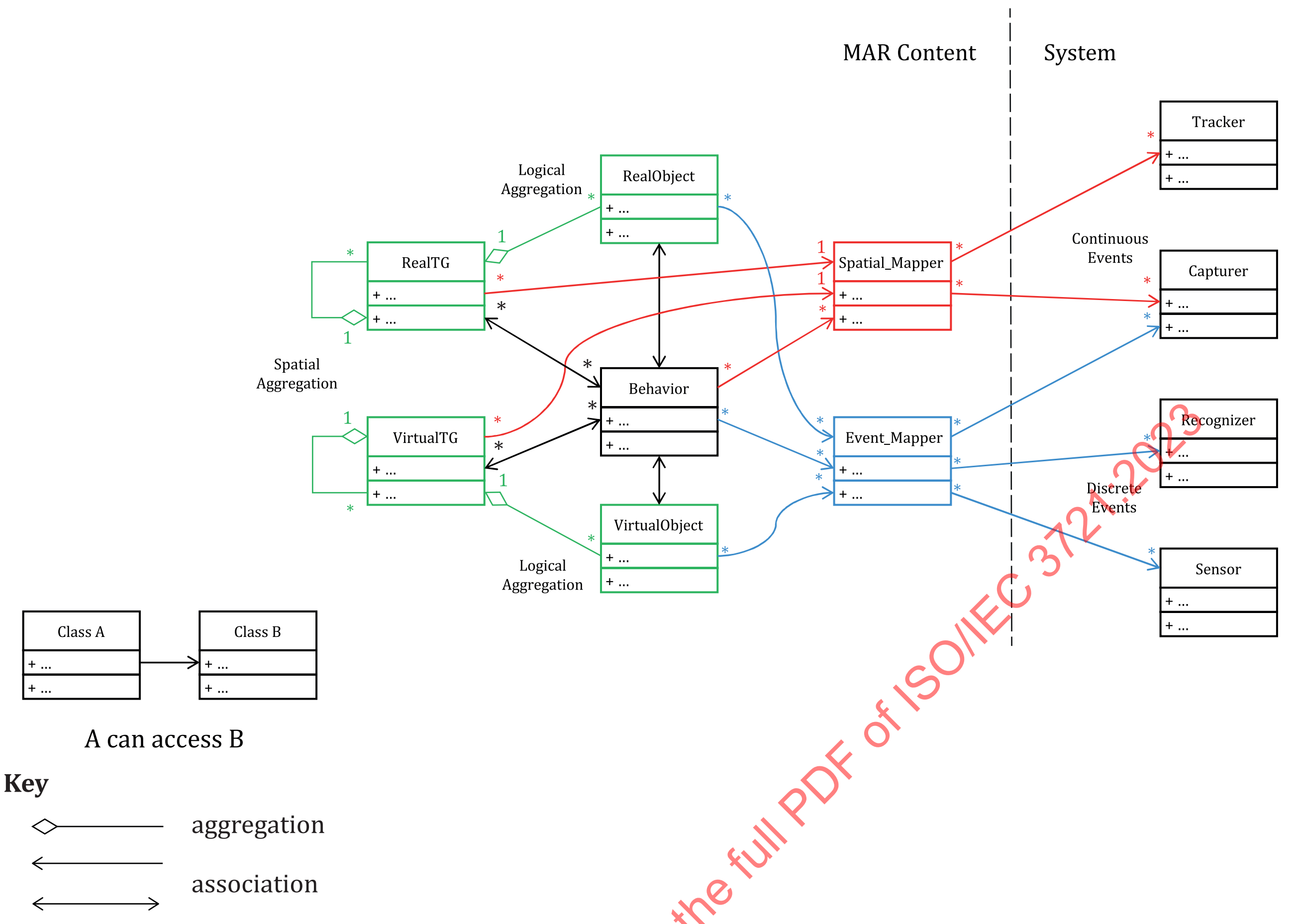


Figure 6 — Association and aggregation relationship among classes in the MAR scene structure (detailed).

7.2 MARSNode

As indicated before, all MAR content components are subclasses of the abstract MARSNode. MARSNode represents the superclass that can take a place in the hierarchical (tree) representation of the scene. In Table 5 is the detailed class specification model with the explanation of the key attributes/methods. All subsequent class/objects derive from this superclass. In this context, class, class object, object are used interchangeably without ambiguity.

Table 5 — A prototypical class specification for MARSNode

MARSNode			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	string	id	unique identifier for reference
private	MARSNode[]	parent	parent nodes (usually, there is only one parent)
private	MARSNode[]	childrenNodes	list of or array of one or more children nodes, also of the MARSNode (or its subclass) type
private	Cube	bounding-box	bounding box specification of for the object this node represents in the MAR scene (optional).
public	MARSNode	MARSCNode()	MARSNode constructor
public	void	init()	abstract initializing method for the MARSNode class

Table 5 *(continued)*

MARSNode			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
public	string	getId()	return the string id of this node
public	void	setId(string id)	set the id of this node
public	void	addChild(MARSCNode child)	add a child to this node of MARSNode type (or its subclass)
public	void	removeChild(MARSCNode child)	remove a child to this node of MARSNode type (or its subclass), if it exists.
public	void	removeAllChild()	remove all children nodes, if any
public	MARSNode[]	getChildren()	return the list/array of children nodes
public	Cube	getBoundingBox()	recompute and update the bounding box for this node considering all the sub-objects to this node and update the attribute bounding-box
public	MARSNode[]	getParent()	return the list/array of parent nodes

7.3 TransformGroup::MARSNode

This class specifies a coordinate system and relative spatial relationship with respect to a reference parent coordinate system (also represented as another TG). This means, a TG will usually have an aggregation relation to a single parent, and children constituent objects who share the common parent coordinate system. Presumably there exists an assumed root world coordinate system. Thus a TG without an explicit parent specification has this assumed root as its parent. The coordinate system information amounts to the 3D translation, 3D rotation and scaling factors in the three principal directions.

TG, as a basic construct for specifying a spatial position/pose within the MAR scene graph/structure, can also be used to spatially designate a real world physical object in relation to the rest of the MAR scene (or as a placeholder for real objects), see Table 6. To make the distinction between a placeholder for virtual and real world physical object(s), two subclasses are defined, namely, RTG and VTG. Objects within the same space, physical or virtual, are related through aggregation. However, between heterogeneous spaces with separate and different TGs, we establish their the spatial registration explicit by using an association class called Spatial_Mapper (see 7.3 for details).

The Spatial_Mapper association class between RTG and VTG is a mere but explicit indication of a real world physical object augmented with a virtual object or vice versa. The Spatial_Mapper class will also designate one to be the parent the other to be a child to resolve for the consistent spatial representation (see 7.3). Again the same rule applies in that a TG node can only have one parent, either through the parent attribute or through the registration association class.

Note that such a registration may exist not only between RTG and VTG, but also among RTGs or VTGs if they are from heterogeneous spaces. For example, a spatial registration between a physical space in Los Angeles and another physical space in New York may be specified. Similarly, a spatial registration between a virtual space created by application A and another virtual space created by application B may be specified using the same construct.

Finally, the notion of the TG may be in actuality implemented as two separate class constructs. For example, in X3D[8], separate Transform node and Group node exist, each responsible for representing the spatial relationship and logical grouping of sub-objects respectively.

Table 6 — A prototypical class specification for TransformGroup::MARSNode

TransformGroup			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	TransformGroup	parent	single parent coordinate system represented by a TG node. If nil, the parent is the assumed root coordinate system. There should be only one parent for a given TG.
private	Matrix4f	transformMatrix	relative spatial relationship represented in the 4x4 transformation matrix relative to the parent TG node.
private	Registration[]	registration	a list/array of registration this node is associated with
public	TransformGroup[]	TransformGroup()	constructor method for this node
public	void	initIdentityMatrix()	set transformMatrix to identity, i.e. [1, 0, 0, 0; 0, 1, 0, 0; 0, 0, 1, 0; 0, 0, 0, 1] (zero rotation, zero translation)
public	void	translate(Vec3f translation)	set relative translation values and update transformMatrix
public	void	rotate(Vec3f rotation)	set relative rotation values and update transformMatrix
public	void	scale(Vec3f scaling)	set relative scale values and update transformMatrix
public	Matrix4f	getTransformationMatrix()	get transformation matrix of this node
public	Registration[]	getRegistration	get a list/array of registration this node is associated with
public	TransformGroup[]	getPlaceholderFor()	get a list/array of TG nodes that this nodes serves as a parent placeholder (by searching through the Registration nodes this node is associated with)
public	TransformGroup[]	getPlaceholder()	get a list/array of the placeholder TransformGroup node for this node (by searching through the Registration nodes this node is associated with). This attribute should return nil if there exist a parent already or vice versa.

7.3.1 VirtualTG::TransformGroup::MARSNode

This is a subclass (abbreviated as VTG) of the TG specialized for virtual object(s), see Table 7.

Table 7 — A prototypical class specification for VirtualTG::TransformGroup::MARSNode

VirtualTG			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
public	int	type	Set to 0 to indicate virtual TG
public	TransformGroup	VirtualTG()	constructor

7.3.2 RealTG::TransformGroup::MARSNode

This is a subclass (abbreviated as RTG) of the TG specialized for real world physical object(s), see Table 8.

Table 8 — A prototypical class specification for RealTG::TransformGroup::MARSNode

RealTG			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
public	int	type	Set to 0 to indicate virtual TG
public	TransformGroup	RealTG()	constructor

7.4 Spatial_Mapper::MARSNode

The Spatial_Mapper is an association class among TG classes that explicitly specifies how heterogeneous spaces or objects are spatially related within the MAR scene structure, see Table 9. Always, one object is designated to be the target object to be augmented, called the placeholder, and the other (one or more) is the augmentation object, called the augmentation. The spatial relationship is specified by the attributes of the Spatial_Mapper class, using the 4x4 transformation in the associated augmentation TGs. This class corresponds to the Spatial Mapper (same identifier) as specified in the system architecture of the MAR Reference Model[6]. In order to map the physical sensor space into the MAR scene or specify spatial relationship among TG's, the explicit mapping information shall be supplied by the content or system developer.

Two objects or environments are said to be heterogeneous when the following condition is met: (1) one is virtual and the other is real/physical (or vice versa), or (2) both are virtual but defined and constructed with two spatially unrelated coordinate references, or (3) both are real/physical but the spatial relationship between the two are unknown. The situation (1) is the most typical case, for which the Spatial_Mapper class provides the explicit spatial registration and denotes an augmentation of the virtual object/environment upon the real or vice versa.

Therefore, such a spatial mapping may exist not only between RTG and VTG for augmentation purpose, but also among RTGs or VTGs if they are heterogeneous spaces. For example, a spatial mapping between a physical space in Los Angeles and another physical space in New York may be specified (assuming the exact spatial relationship between the two are practically unknown or not given). Similarly, a spatial mapping between a virtual space created by application X and another virtual space created by application Y may be specified (thus the exact spatial relation is not known or given). The Spatial_Mapper class is needed to specify the needed spatial relation between them to establish a fused mixed reality environment.

Spatial_Mapper class, if necessary, may specify the method of registration, if not the information is manually specified (e.g. by the 4x4 transformation matrix).

Finally, the Spatial_Mapper class may also associate to MAR system objects (that supply spatial information such as the Capturer, and Tracker, see 6.3) for supplying the actual spatial transformation information to the associated TG objects. This in effect also indirectly specifies the coordinate systems of the MAR system objects in relation to the MAR scene structure. It is assumed that in this case the relative spatial relationship (transformation matrix) is supplied in the associated MAR system object constructs.

Table 9 — A prototypical class specification for Spatial_Mapper::MARSNode

Spatial_Mapper			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	string	placeholder	id of the TG object in the MAR content which is being augmented MAR content. It serves as the parent spatial coordinate system for the augmentation TG objects specified in the augmentation attribute (see next row).
private	string[]	augmentation	list/array of ids of children TG objects in the MAR scene that augments the placeholder TG object. The relative spatial information is provided in the TGs themselves

Table 9 *(continued)*

Spatial_Mapper			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	System	refSystem	id of the associated MAR System component.
private	Any	sensedRawData	record the raw data from the MAR system components (Sensor-Capturer, Sensor-Tracker)
public	Registration	Registration()	constructor
protected	void	init()	initialization
public	string	getPlaceholder	return the placeholder TG object for this node
public	void	setPlaceholder(string tg)	set the placeholder to given TG id for this node
public	string[]	getAugmentation	return the list/array of augmentation TG object(s) ids for this node
public	void	setAugmentatino(string[])	set the augmentation to the given list/array of TG ids for this node
public	System	getSystem()	return the associated MAR System component, if any
public	void	setSystem(System* x)	associate a MAR System component to this node
private	void	updateDataFromSystem()	channel and update the information from the MAR System component to the associated TG object. SensedRawData are automatically updated.
public	Any	getSensedRawData()	return the values of senseRawData, if any

7.5 Event_Mapper::MARSNode

Like any interactive systems, MAR content behaviour and its simulation are driven by various events and data as produced by the MAR system components through sensor based environment monitoring and user interaction. For example, the Real World Capturer, Recognizer and Tracker are some of the MAR system components that are the sensor based modules that produce the events and data stream needed to drive the MAR content behaviour (see 6.3). Event_Mapper class corresponds to the Event Mapper (same name) as specified in the system architecture of the MAR Reference Model, see Table 10[6]. Provision of the details of the mapping is the responsibility of the content or system developer.

The MAR system components are not aware of the specifics of the content. Whatever they capture, recognize and track are understood only within their scopes. For instance, a location as recognized by a GNSS in latitude = 34.05 and longitude = −118.24 coordinates would be mapped to a particular content identifier, such as “My Home Town” which is used and understood in the content in a more intuitive manner.

In order to relate events/data to a specific content, a event mapping shall be defined so that the item captured, recognized and tracked will be understood within the scope of the given specific content, by the Event_Mapper construct. In addition, the mapping may apply simple processing to the event and data for miscellaneous purposes such as unit conversion, data filtering and smoothing, orientation change, etc.

We assume that the generic data and event types already exist. Examples of discrete events might include those for object existence, object being in a particular location or pose, user interaction (touch, gesture, click, etc.), contextual data (e.g. time, identity, place) and other user defined types. Examples of continuous and stream of data might include those for general sensor values (e.g. time), images/video frames, spatial tracking information (in various dimensions).

Therefore, the Event_Mapper will have the events/data from the MAR system components ready for consumption by the MAR content. Various MAR content components such as the Behaviour (see 7.6), MARObject (see 7.5), TG nodes (see 7.2), Spatial_Mapper (see 7.3) can associate to the Event_Mapper to obtain needed event and data values.

Table 10 — A prototypical class specification for Event_Mapper::MARSNode

Event_Mapper			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	string[]	source	refer to one (or more) source of the event or data (usually a MAR system component)
private	string[]	receiver	refer to one or more receiver of the event from the source(s).
private	System	refSystem	id of the associated MAR System component.
private	Any	sensedRawData	record the raw data from the MAR system components
public	Mapping	Mapping()	constructor
protected	void	init()	initialization
public	string[]	getSource()	return list/array of one (or more) source of the event or data (usually a MAR system component)
public	void	setSource(System* s)	associate a given MAR system component to this node as source
public	string[]	getReceiver()	return list/array of one or more receiver of the event from the source(s).
public	Void	setReceiver(MARSGNode* n)	associate a given MAR content component to this node as recipient of the data from the source(s)
public	System	getSystem()	return the associated MAR System component, if any
public	void	setSystem(System* x)	associate a MAR System component to this node
private	void	updateDataFromSystem()	channel and update the information from the MAR System component to the associated TG object. sensedRawData are automatically updated.
public	Any	getSensedRawData()	return the values of senseRawData, if any
public	void	triggerReceipient()	Trigger the recipient object(s) behaviour manually

7.6 MARObject::MARSNode

The MARObject class represents and specifies the actual concrete objects in the scene (see Table 11). In VR, all objects are assumed to be virtual. In MAR, we make an explicit distinction between the real and virtual objects. Detailed information model for the virtual objects need not be specified in this document as there are many existing standards and approaches. Typical virtual objects include 2D/3D models, text, sound, animations, scripts, bounding box, light source, view point, etc. However, few new virtual object types can be of help for more fluid MAR content specification. One example is a dedicated node or construct for Live camera node for supporting video see-through MAR display. Another is a dedicated construct for Live actor and Entity (LAE) (for this, refer to for its detailed information model as an extension to this standard in References [7,11]) as captured and imported from specialized LAE Capturer such as the chroma-keying system.

However for representation of the real physical objects, especially those that are typically used in the MAR context, require new standard information models. While there are practically infinite types of physical and real objects in the world, those that are typically used in MAR (due to the technological limitation) in practice, include the GNSS based location, markers (square coded imagery with black boundaries or features easily recognizable by image processing systems), feature-rich image patches (features refer to primitive visual elements such as points, lines, corners, colour, etc.). Note that however, the technology is moving fast such that almost any real and physical objects can be recognized and be used for MAR. Standards such as SEDRIS[14] is an excellent candidate as a way to represent real environment objects in a more general fashion.

Real objects and virtual objects are distinguished (aside from its type attribute, see 7.2) also by its TG parent. All real and virtual objects will eventually be grouped under the RTG and VTG respectively (see 7.2).

Table 11 — A prototypical class specification for MARObject::MARSNode

MARObject			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
public	int	type	set to 0 indicating it is virtual, 1 if real
private	TG	parentTG	parent coordinate system
private	TG[]	childrenTG	llist/array of children coordinate system if any
private	MARSystem[]	systemComponents	list/array of associated sensor, capturer, recognizer, tracker for this object
private	Registration	registration	list/array of associated registration for this object
private	Mapping	mapping	list/array of associated mapping for this object
public	ObjectNode*	ObjectNode()	constructor
protected	void	init()	initialization
public	void	getParentTG()	access the parent coordinate system
public	MARSystem[]	getMARSystemComponents	return list/array of associated sensor, capturer, recognizer, tracker for this object
public	Registration[]	getRegstiration	return list/array of associated registration for this object
public	Mapping[]	getMapping	return list/array of associated mapping for this object

7.6.1 VirtualObject::MARObject::MARSNode

VirtualObject is a subclass (abbreviated as VO) of the MARObject class for representing virtual and synthetic computational objects (see Table 12). There would be numerous subclasses and types such VirtualObject as the text, image, 3D model, live background video, animation, bounding box, light source, viewpoint/camera, etc. Such virtual object components are already standardized in various ways[8]. We merely list out those that are necessary for MAR content functionalities that are not supported by the conventional virtual object representations such as the background video object. We also make a note that separate efforts are under way for specifying detailed virtual and real object models (such as LAEMapper, LAEObject) and associated MAR system components (such as LAECapturer, LAERecognizer) for supporting the live actor and entities (e.g. avatars) which is one of very important objects in many MAR contents[7,11].

Table 12 — A prototypical class specification for VirtualObject::MARObject::MARSNode

VirtualObject			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
public	int	category	type of the virtual object (text, image, ...)
public	VirtualObject	VirtualObject(int category)	constructor
protected	void	init(int category)	initialization

7.6.1.1 BGVideo::VirtualObject::MARObject::MARSNode

Background video is much used in MAR contents, e.g. as the backdrop for representing the real world physical environment in AR or for representing singular real world physical objects (captured as video objects) in AV, see Table 13. Note that this object can either be captured/used live, or otherwise off-line.

Table 13 — A prototypical class specification for BGVideo::VirtualObject::MARObject::MARSNode

BGVideo			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	bool	type	0 for live and 1 for off-line pre-recorded
private	string	source	file name of the pre-recorded video or moving texture object (if type = 1)
private	Mapper	Mapper	reference to the Mapper for the captured data – has indirect reference to the Capturer for this video object (if type = 0)
public	Video	BGVideo()	constructor
protected	void	init()	initialization
public	bool	getType	return type of this node (live or off-line)
public	string	getSource	return source of off-line video if the type matches
public	Mapper	getMapper	return mapper that handles the live capture video if the type matches
public	Capturer	getCapturer	return the actual sensor/capturer object if the type matches

7.6.2 RealObject::MARObject::MARSNode

RealObject is a subclass (abbreviated as RO) of the MARObject for representing real world physical objects, see Table 14. These are used as targets of augmentation in AR or as augmentation objects into virtual environments in AV. While there can be practically infinite types of real world physical objects, we create subclasses for those that are typically used in MAR contents such as the marker, textured image patches, 3D objects with model descriptions, geo-locations, and labelled image segments (in subsequent subclauses of 7.6.2). Note that the following subclasses of RealObject are by no means exhaustive. New subclasses can be defined as necessary.

Table 14 — A prototypical class specification for RealObject::MARObject::MARSNode

RealObject			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
public	int	category	specific category of the real object (marker, image patch, 3D object, GPS, ...)
public	RealObject	RealObject(int category)	constructor
protected	void	init(int category)	initialization
private	Any*	features	features needed to be recognized by the associated recognizer

7.6.2.1 Marker::RealObject::MARObject::MARSNode (see Table 15)

Table 15 — A prototypical class specification for Marker::RealObject::MARObject::MARSNode

Marker			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	int	type	type of marker (e.g. QR code, proprietary, etc.)
private	string	filepath	path to the marker image file
private	image	image	image data of the marker (if not provided as a file)
private	Vec2f	size	size of marker in pixels
public	Marker	Marker()	constructor
protected	void	init()	initialization
public	string	getFile	return marker file path/name
public	Image	getImage	return raw image data
public	Vec2f	getSize	Return size of marker in pixels

7.6.2.2 GNSSLocation::RealObject::ObjectNode::MARSGNode (see Table 16)

Table 16 — A prototypical class specification for GNSSLocation::RealObject::MARObject::MARSNode

GNSSLocation			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	Vec2f	coordinate	GNSS coordinate in 2 component vector of longitude and latitude
private	string	geoLabel	geographic label for this location
private	float	accuracy	accuracy of this GNSS location in meters
public	GNSSlocation	GNSSlocation()	constructor
protected	void	init()	initialization
public	Vec2f	getLocation	Return GNSS location
public	string	getGeoLabel	return geographic label for this location
public	float	getAccuracy	return accuracy of this GNSS location in meters

7.6.2.3 TexturedImage::RealObject::MARObject::MARSNode (see Table 17)

Table 17 — A prototypical class specification for TexturedImage::RealObject::MARObject::MARSNode

TexturedImage			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	int	type	type of feature descriptor used
private	string	filepath	path to the rectified textured image file
private	string	featurefilepath	path to the feature descriptor information of the textured image file
private	image	image	image data of the textured image (if not provided as a file)
private	Any	featureDescriptor	feature descriptor information in predefined format
private	Vec2f	size	size of marker in pixels

Table 17 *(continued)*

TexturedImage			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
public	TexturedImage	TexturedImage()	constructor
protected	void	init()	initialization
public	int	getType	return type of feature descriptor used
public	string	getFile	return marker file path/name
public	Image	getImage	return raw image data
public	string	getFeaturefilepath	return path to the feature descriptor information of the textured image file
private	Vec2f	getSize	return size of marker in pixels

7.6.2.4 3DObject::RealObject::MARObject::MARSNode (see Table 18)

Table 18 — A prototypical class specification for 3DObject::RealObject::MARObject::MARSNode

3DObject			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	int	type	type of feature descriptor used
private	string[]	imagesetfilepath	path to the multi-view rectified image files folder
private	string	featurefilepath	path to the feature descriptor information of the textured image file
private	string	modellfilepath	path to the model information of the textured image file
private	Image[]	imageset	multi-view image data of the 3D object
private	Any	featureDescriptor	feature descriptor information in predefined format
private	Any	modelInformation	model data information in predefined format
private	Vec3f	size	3D size of object in pixels
public	3DObject	3Dobject()	constructor
protected	void	init()	initialization
public	int	getType	return type of feature descriptor used
public	string	getImageFilepath	return multi-view image set file folder path/name
public	string	getFeaturefilepath	return path to the feature descriptor information of the 3D model
public	string	modellfilepath	return path to the model information of the textured image file
public	Vec3f	getSize	return size of 3D model in pixels

7.6.2.5 RFIDTag::RealObject::MARObject::MARSNode (see Table 19)

Table 19 — A prototypical class specification for RFIDTag::RealObject::MARObject::MARSNode

RFIDTag			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	string	label	unique RFID tag label
public	Rfid	Rfid()	constructor
protected	void	init()	initialization
public	string	getLabel()	return RFID tag label

7.6.2.6 LabelledImageSegment::RealObject::MARObject::MARSNode (see Table 20)

Table 20 — A prototypical class specification for LabelledImageSegment::RealObject::MARObject::MARSNode

LabelledImageSegment			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	string	label	unique object label returned by the AI recognizer
public	ImageSegment	imageSegment()	constructor
protected	void	init()	initialization
public	string	getLabel()	return RFID tag label

7.7 Behaviour::MARSNode

Behaviour class specify the dynamics of the objects in the scene, e.g. time based or event based behaviour. Therefore, such behaviours can be expressed simply by relying on the usual scripts. Behaviour nodes, in addition to housing such flexible scripts, serve to abstract some typically used MAR augmentation behaviours (i.e. how virtual augmentation objects associated and spatially registered to a real physical object behave reacting to input and external events). The main purpose of such an abstraction is the ease of use, simplicity and thereby quick authoring. Behaviour nodes are driven by external events/data produced from the MAR system components (such as the sensor, capturer, tracker and recognizer, see 6.3) and accessible indirectly through the Mapping node (see 7.4), and simulated by the MAR simulation engine (another important part of MAR system). The behaviour node will have associations with the content objects which are needed to drive and simulate the behaviour and those that are affected by the simulation as well. See Table 21.

Examples of often used augmentation behaviour are simple activating/deactivating of the augmentation to be visible, simple animation of them, highlighting effects, changing of transparency and colour.

Table 21 — A prototypical class specification for Behaviour::MARSNode

Behaviour			
Access type	**Data type**	**Attribute/Method name**	**Explanation**
private	int	type	specifies the specific type of the behaviour
private	bool	isTriggered	Enabling or disabling the triggering of this Behaviour node (0 for disabling it, and 1 for enabling it)
private	Mapping[]	eventSource	list/array of source Mapping node(s) that delivers and filters the event/data to this node from the MAR System components