Finding interactive 3D objects by their interaction properties

Jacek Chmielewski

Published online: 4 June 2012

© The Author(s) 2012. This article is published with open access at Springerlink.com

Abstract The problem of metadata for interactive 3D objects has appeared very recently. It is significant due to the growing demand for interactive applications of 3D technologies. Time required to prepare such applications depends strongly on availability of reusable interactive 3D objects. Easy access to such objects can be increased by search solutions that cover not only object geometry and semantics but also object interactions. However, existing metadata standards provide tools only for general, technical and semantic descriptions of objects. The missing elements of object description solution are metadata of object interactions and an optimized query language. In our previous research we have developed an extensible solution for describing interactions of 3D objects. In this paper, we present the second missing element, a specification of a special query language that enables efficient usage of the interaction metadata.

Keywords Metadata · Interactive 3D objects · Virtual reality · Query language

1 Introduction

3D technologies are used in many domains, from professional computer aided design systems, simulations, and telemedicine, to search [44], education [42], personal and entertainment multimedia solutions [38] such as virtual museums [39, 43], mobile devices [29], interactive TV [16], networked home appliances, on-line multi-user 3D games [31], and interactive Internet applications. Apart from 3D content, 3D solutions are also used for building user interfaces for operating systems [15, 28] and applications [22, 30]. Such 3D user interfaces and content objects are not only static 3D figures. Creation of new 3D interfaces and applications [37, 40] requires the use of interactive 3D objects. An interactive object recognizes a number of external events and reacts according to predefined algorithms. A complete set of such algorithms composes an object's behavior. The behavior of an interactive 3D object may be programmed externally in an application or stored together

J. Chmielewski (✉)
Poznań University of Economics, Poznań, Poland
e-mail: chmielewski@kti.ue.poznan.pl

with the object [36]. The second approach allows an easy exchange and reuse of interactive 3D objects used as application components, without the need to modify the application code. As a result, time and effort required to build an interactive 3D scene or to compose a 3D world decrease.

The benefits of interactive 3D objects increase with the growth of shared, searchable libraries of reusable 3D objects. However, with the increasing number of reusable interactive 3D objects, collections of such objects may become so immense that finding a right object will be very time consuming. A solution to this problem is to create tools that will facilitate finding 3D objects matching requirements of a particular application, especially interaction requirements. Some tools already exist. For example, a prototype of a search engine that uses geometric characteristics of 3D objects has been created at Princeton University [19] and is available on the Internet [1]. However, to achieve highly relevant search results a search engine needs to address all aspects of an interactive 3D object – its geometry [4], semantics [21] and behavior. While geometry information can be extracted from the object, it's difficult and sometimes impossible to gather semantics and especially behavior automatically. Therefore, there is a need for a metadata solution and a corresponding querying language that will allow describing additional information about the object and building search engines capable of using such information. The semantic part of object description is addressed by a number of existing metadata standards and in our previous research we have developed a solution for building the behavior metadata – the Multimedia Interaction Model. In this paper, we propose the second element, a new query language for finding interactive 3D objects, which enables easy creation of queries that address not only general or technical metadata but also interaction metadata of 3D objects.

This paper is organized as follows. Section 2 provides background information on metadata of interactive 3D objects. The query language proposed in this paper is based on results of our previous research on interaction metadata, so Section 3 presents our approach to interaction metadata. Section 4 contains a description of the new Interaction Metadata Query sub-Language. In Section 5, an example of interaction metadata and results of using the proposed query language are presented. Section 6 concludes the paper.

2 Metadata of interactive 3D objects

Interactive 3D objects belong to the more general class of multimedia objects and it would be best to describe their various properties with existing metadata standards for multimedia objects. There exist many such metadata standards tailored for different domains and applications: still images can be described with EXIF, DIG35, or NISO Z39.87; audio content can be described with ID3 or MusicBrainz; and metadata solutions specific to audio-visual content are AAF, MXF DMS-1, and P_Meta. There are also generic metadata standards used to annotate multimedia objects. The most popular are Dublin Core, MODS, XMP, and MPEG-7 [25]. However, none of these metadata solutions include features specific to 3D objects [7].

To fill this gap, various research groups proposed their own solutions or extensions of existing standards. A group from Laboratoire Informatique de Grenoble proposed a MPEG-7 extension called 3D SEmantics Annotation Model (3DSEAM) [4], which introduced two new 3D specific descriptors: Structural Locator and 3D Region Locator [5]. Another 3D metadata solution was developed in the AIM@SHAPE project [18]. The proposed extension of the COLLADA file format introduces an ontology for the description of virtual human

body and for linking semantics to shape or shape parts. A similar idea was researched in the SALERO (Semantic Audiovisual Entertainment Reusable Objects) project [23]. However, the final results of the project [33] are focused more on automation of digital content production than on universal models of metadata of 3D objects.

Currently, the best available solution for metadata of 3D multimedia objects is MPEG-7 complemented with 3DSEAM extensions. Nevertheless, the MPEG-7 and its extensions were designed to address the general, technical, semantic and structural properties of non-interactive objects and are not capable of fully describing interactive 3D objects. Therefore, we have proposed a solution for describing object interactions, which could be used to enhance the descriptive power of MPEG-7 and 3DSEAM. The solution is presented briefly in the next section.

3 Interaction metadata solution

The interaction metadata solution is composed of two main elements: *Multimedia Interaction Model* and *Interaction Interface*. The Multimedia Interaction Model provides the structure for interaction metadata, and the Interaction Interface concept ensures the overall solution is extensible and capable of addressing various aspects of object interactions. Both elements are described in more detail in the following sections.

3.1 Multimedia interaction model

Metadata description of object interactions is a set of separate descriptions of all possible object interactions. The Multimedia Interaction Model (MIM) [8] is an abstract model of an interaction of one *trigger object* (a simple or compound object in a given 3D virtual world that triggers the interaction) with a particular *interacting object* (the object described by interaction metadata). It can be applied to any interaction that occurs in a 3D virtual world. The MIM model describes an interaction from the point of view of the interacting object. This means that the interaction description is built around the object reaction and a context that triggers the reaction. This approach is based on the *Event-Condition-Action* (ECA) paradigm that comes from the active database field [17], and is used currently to express active DMBS features and event-reaction oriented features in other systems [6, 26, 34, 45, 46]. The way the ECA paradigm is used in the Multimedia Interaction Model is the following. The *Event* element corresponds to the trigger of an interaction. The trigger is a change of a selected property (*parameter*) of some object or a virtual environment in which the interaction occurs. The *Condition* element describes the context of an interaction that is required to allow the action. The context is a set of values of selected parameters of the interacting objects or the virtual environment. The *Action* element describes the object reaction. The reaction describes how the object changes itself in a reply to an event in a given context. The Event, Condition and Action elements are composed into metadata of a single interaction created according to the Multimedia Interaction Model (MIM) schema [9, 14].

3.1.1 Event

An event is a change of the state of a trigger object. The state of an interactive object is a set of values of all parameters of the object. The Event component of the MIM interaction metadata contains a set of preconditions that need to be satisfied to trigger

a particular interaction. The preconditions include a specification of a 3D interaction space and a non-empty set of object parameter types. To satisfy the preconditions, the event has to occur inside a specified 3D interaction space and object parameters, whose values are modified, have to belong to one of specified object parameter types. The 3D interaction space is represented by a set of 3D spaces and 3D viewpoints adopted from the X3D language. If no geometry is specified, the 3D interaction space is equal to the whole 3D virtual world.

3.1.2 Condition

A condition is a set of requirements imposed on the states of the trigger object, the environment and the interacting object. A condition needs to be satisfied to trigger an action. The Condition component of the MIM interaction metadata is represented by a mathematical description and a semantic description. The mathematical description has a form of a MahtML logical expression which represents all requirements of a single condition. The semantic description of a condition uses MPEG-7 TextAnnotation descriptor and may be used as a substitute or an enhancement of the mathematical description. The Condition element of the MIM metadata is optional and can be omitted. In such a case, an interaction will be executed at each occurrence of the event.

3.1.3 Action

An action is a change of the state of the interacting object. In the MIM interaction metadata, the Action component describes the outcome of an action, a new object state. The new object state is expressed as a set of mathematical expressions (MathML) used to calculate new values of object parameters and a semantic description (MPEG-7 TextAnnotation). An action can be composed of an arbitrary number of sub-actions that provide more detailed descriptions. Such an approach allows building action descriptions that can be evaluated on different levels of detail.

3.2 Interaction interface concept

Interactive features of an object can be described in terms of functions and attributes related to object interactions. Both elements may be represented by object *parameter types*. It is not feasible to predefine all possible interaction parameter types, and the Interaction Interface concept [10, 12] enables definition of new parameter types and creation of interaction metadata for new domains. Definition of new parameter type requires basic information like parameter type ID, data type, optional semantic description and relation to an external ontology [13]. Such design allows linking interaction metadata provided by MIM with external semantic metadata tools such as MOWL [20] or SIRM [2].

According to the Multimedia Interaction Model, to enable creation of interaction metadata, an interactive 3D object has to implement at least one interaction interface. An object implements a given interaction interface if the set of object parameter types is a superset of the set of parameter types of this interaction interface. This way, each application domain may have its own interaction interface [11]. However, there are some common properties of all 3D objects that allow forming an interaction interface common for all objects. This interaction interface is called: Visual Interaction Interface. It represents object parameter

types related to object visual characteristics, like object geometry (size, shape), position (position, collision, orientation) and appearance (color, texture, transparency). This common interaction interface can be used to build metadata of visual and geometrical interactions.

4 Interaction metadata query sub-language

To make search engines able to use interaction metadata, a query language is required. Since there are no interaction metadata standards, existing metadata query languages do not contain optimized features for querying 3D objects using interaction criteria. To fully exploit the potential of interaction metadata based on the Multimedia Interaction Model such a language has to support basic keyword queries, semantic searches, and object matching. Also, the query language has to be extensible to be able to employ new interaction parameters defined by new interaction interfaces. Finally, as the interaction metadata will always be only a part of a complete object description, the query language has to be modular to enable its use along with different search languages and systems.

To address these requirements, we propose the Interaction Metadata Query sub-Language (IMQL) which complements the MIM model. Combined, they form a complete solution for interaction metadata creation and use. The IMQL provides a set of operators that can be used in value and keyword comparisons, and in semantic searches. IMQL syntax relates directly to the MIM model structure and provides references to interaction interfaces of compared objects and parameter types of these objects. Object matching queries are complex and difficult, especially in the field of interactions. The use of interaction interfaces facilitates object matching queries. The IMQL integrates with Interaction Interface concept permitting the use of new object parameter types and value data included directly in the query. The syntax of IMQL query is similar to that of standard SQL queries, where four main parts are used: SELECT, FROM, WHERE and ORDER BY. In IMQL, the SELECT and ORDER BY parts are implicit. The SELECT part is not included explicitly because IMQL always selects 3D object URIs. The ORDER BY part is implicit because the ordering provided by IMQL is fixed. Details of IMQL ordering rules are explained later. Two parts that IMQL uses explicitly are FROM and WHERE. The meaning of both parts is the same as in SQL queries. The first part defines the source of object metadata, while the second part specifies filtering criteria. The first part can be replaced by results of another search query, while the output of the filtering criteria can be passed for further processing (see Fig. 1). Such solution allows embedding an IMQL query within a larger query expressed in a different query language and performing a complex search by a single search system.

The IMQL is based on the following assumptions. It is assumed that interactive objects are grouped in uniquely identified repositories. Each repository contains a set of interactive

Fig. 1 Diagram of IMQL WHERE clause embedded in SQL query

objects. Each object in a repository is identified by a unique URI and has interaction metadata built according to the MIM model. The IMQL defines the syntax of the query language and the meaning of different language elements and comparison operators. Any specialized comparison or matching algorithms, especially semantic analysis algorithms, are left to be implemented by search engines, to allow for competition among search engine providers.

The IMQL query syntax is the following:

```
imql_query = "FROM" repository_URIs ("WHERE" where_expr)?
```

The IMQL is composed of two clauses: the FROM clause and the WHERE clause. The FROM clause defines the source of interactive objects, and the WHERE clause contains filtering criteria. The result of execution of an IMQL query is always a set of URIs of objects whose interaction metadata match given criteria.

The specification of the syntax of the IMQL is given in this paper using a slightly modified version of the Extended Backus-Naur Form (EBNF) notation defined by W3C for XML recommendations [35]. The only modification is the change of the symbol-expression assignment operator. In W3C notation it is ‘::=’ while in this paper it is ‘=’.

4.1 FROM clause

The purpose of the FROM clause is to identify interactive objects whose metadata are evaluated against conditions defined in the WHERE clause. Objects to be evaluated are taken from repositories of interactive objects identified by *repository_URIs*. The FROM clause is the only required part of a standalone query. If the WHERE clause is omitted, the query returns a set of URIs of all objects from all given repositories. Objects from each repository are taken in an arbitrary order. However, the order of groups of objects taken from subsequent repositories is the same as the order of repositories in the query. Object ordering is illustrated by the following example.

Example The above ordering rule means that if a query selects all objects from repository A containing objects A1 and A2, and from repository B containing object B1 and B2, the result is one of the following: A1, A2, B1, B2; A2, A1, B1, B2; A1, A2, B2, B1; A2, A1, B2, B1. The order of objects from each repository is arbitrary, while the order of groups of objects from different repositories reflects the order of repositories in the query. Any other order of object URIs is invalid.

The ordering of objects resulting from the FROM clause is not final. The results can be reordered after filtering with the WHERE clause. At this stage of IMQL specification, the language does not provide any internal ordering tools. Introduction of ordering facilities is planned as future work and currently, to fill this gap, IMQL allows external ordering mechanisms that can be provided by search systems. Such ordering mechanisms can be based on semantic relevancy factors that depend on particular search engine implementation. The proposed approach enables competition among search systems since better results will be returned by search systems with better relevancy algorithms.

4.1.1 FROM clause syntax

```
from_expr = "FROM" repository_URLs

repository_URLs = single_URI (","single_URI)*

single_URI = ""URI""
```

The URI provided in the FROM clause represents a Uniform Resource Identifier as defined by RFC 3986 [3]. The URI has to be enclosed in single quotes. Multiple URIs have to be separated by commas.

Example Consider the previous example. The notation of the query is the following:

```
FROM 'A', 'B'
```

The above query returns URIs of all objects from repository A and URIs of all objects from repository B. The list of objects is not filtered in any way and the order of objects is as described in the previous Example.

4.2 WHERE clause

The optional WHERE clause serves as a filter for interactive objects identified by the FROM clause. The expression in the WHERE clause, called the *where_expr*, is evaluated once for each of these objects. The *where_expr* is a logical expression of requirements set on interaction properties of evaluated objects. If the Boolean value of the *where_expr* is false, the object is discarded. The order of objects remains the same as defined in the FROM clause.

The IMQL is designed as a modular language and it can be used as an extension of other query languages. The integration is accomplished by embedding the WHERE clause *where_expr* as one of search conditions in the extended query. In such a case the FROM clause is omitted and the search system has to serve as a proxy between a source list of objects and a set of object metadata URIs required for the evaluation of the *where_expr*. The result of *where_expr* evaluation, a filtered set of object URIs, can be passed further for next processing steps.

4.2.1 WHERE clause syntax

```
where_expr = condition_expr (logical_operator condition_expr)*
```

The *where_expr* is composed of at least one *condition_expr*. If there is more than one *condition_expr*, each two are separated by a *logical_operator*.

```
condition_expr = (single_condition (logical_operator single_condition)*)|("where_expr")
```

The *condition_expr* consists of at least one *single_condition* or exactly one *where_expr* enclosed in parenthesis. If there is more than one *single_condition*, each two are separated by a *logical_operator*. The fact that higher-level *where_expr* appears inside *condition_expr* and is enclosed in parenthesis enables creation of multi-level logical conditions with easily defined operation precedence.

```
logical_operator = "&" | "|" | "&!" | "!"
```

The *logical_operator* is either logical AND (&) or logical OR (|). The other two are shortcuts for using these operators in conjunction with logical NOT: logical AND NOT (&!) and logical OR NOT (!|).

```
single_condition = operand comparison_operator operand
```

```
operand = element_identifier | value
```

The *single_condition* consists of exactly two operands separated by *comparison_operator*. The operand is either an *element_identifier* or value.

```
comparison_operator = "==" | "!=" | "<" | "<=" | ">" | ">=" | "=" | "~=" | ">>"
```

IMQL supports nine comparison operators:

- equal ==
- not equal !=
- less <
- less or equal <=
- greater >
- greater or equal >=
- almost equal =
- like ~=
- in >>

The first six operators are basic comparison operators. They work according to mathematical rules and their standard use does not require further explanations. Basic comparison operator can be used also to compare 3D features like 3D points and 3D geometry. For 3D points, two points are equal if all three point coordinates are exactly the same. A 3D point is greater or less than other 3D point if the line from the beginning of the local coordinate system to this 3D point is longer or shorter correspondingly. For 3D geometry, the comparison is done based on the volume of the geometry. For example, a sphere with the diameter equal 2 is less than a cube of 2.

The seventh operator, ‘almost equal’ (=), is useful for comparing keywords with semantic description of the interactions, especially for descriptions linked with a domain specific ontology

or when textual descriptions are analyzed with natural language processing (NLP) algorithms. For such cases, submitted keywords can be matched not only with the exact description term but also with higher-level equivalents from the ontology or terms with similar meaning. In general, the ‘almost equal’ operator can be used for any values for which calculating some similarity measures is possible. For example, for object shape: a 3D shape can be described with an MPEG-7 Shape3D descriptor and some similarity measures can be obtained. The raw result of such comparison is a number: relevancy or similarity factor, not a Boolean value. The final Boolean value is obtained by comparing the resulting number with a given threshold which can be set by the search engine user. If the number exceeds the threshold, the ‘almost equal’ operator returns true. The algorithm for calculating relevancy or similarity measures and the comparison threshold are not included in the IMQL definition and are left to be defined by a search system.

The eighth operator, ‘like’ ($\simeq$), is a comparison operator that works with regular expressions. It returns true if the value of selected object interaction property matches a given pattern. Regular expressions are constructed according to the POSIX extended regular expressions syntax [32]. However, depending on the search system, available regular expressions syntax can be extended, for example, to support Perl-compatible regular expressions (PCRE).

The last operator, ‘in’ ($\gg$), is not a strict comparison operator. It verifies whether a given value is contained in a set provided as the right operand. By default it uses strict comparison to match a given value with values taken from the set. However, it is possible to use a set of regular expressions. In such a case, the operator has to be prefixed with tilde ‘ $\sim$ ’. Also, this operator can be used for verifying 3D features, like a point in a 3D space. In such a case, the right operand is not a set of values but a 3D space, and the left operand has to be a set of 3D points. The ‘in’ operator verifies if the 3D points are contained inside the 3D space.

```

element_identifier = ii | event | condition | action | interaction

interaction = "interaction.paramtype.id" | "interaction.semantics"

ii = "ii" | "ii.id" | "ii.name" | "ii.semantics"

event = "event.space" | "event.space.semantics" | "event.paramtype"
        | "event.paramtype.id" | "event.paramtype.name"
        | "event.paramtype.semantics" | "event.semantics"

condition = "condition." source ".paramtype" | "condition." source ".paramtype.id"
            | "condition." source ".paramtype.name"
            | "condition." source ".paramtype.semantics" |
"condition.semantics"
            | "condition." parameter ".value" ( "." ( "after" | "before" ) )?

action = "action." source ".paramtype" | "action." source ".paramtype.id"
        | "action." source ".paramtype.name"
        | "action." source ".paramtype.semantics" | "action.semantics"
        | "action." parameter ".value" ( "." ( "after" | "before" ) )?

source = "trigger" | "interacting" | "environment"

parameter = source "." opt_id

opt_id = interaction_interface_id "." object_parameter_type_id

```

The *element_identifier* is a reference to the whole interaction or to a specific property of one of the MIM model elements. In the case of a reference to an interaction (*interaction*), the *element_identifier* can address parameter types used in any element of the MIM model (*interaction.paramtype.id*) or interaction semantics (*interaction.semantics*). The *element_identifier* can refer also to interactive interface (*ii*), interaction event, condition, or action. For interactive interfaces the *ii* may refer to interactive interface ID (*ii.id*), name (*ii.name*), semantic description (*ii.semantics*), or all of them (*ii*). Interactive Interface name and ID are both textual values and can be compared with a string (e.g. keyword), regular expression or a term in a domain specific ontology. Interactive Interface semantic description can be a free text, a set of keywords, or a more complex description (structural or dependency annotation). The free text and keyword annotations are handled in the same way as name and ID, while more complex annotations can be analyzed with more advanced text analysis algorithms.

Example Consider a search that takes interactive objects from repository 'LocalRep' and returns only objects implementing fictitious *Aural Interaction Interface*. The query for such search is the following:

```
FROM 'LocalRep' WHERE ii.name == 'Aural Interaction Interface'
```

The same query rewritten to match all interactive interfaces with 'aural' in name, not only the *Aural Interactive Interface*, is the following:

```
FROM 'LocalRep' WHERE ii.name ~= '.*[aA]ural.*'
```

The *event* refers to either an interaction space or types of parameters that trigger the event. The interaction space is represented by a semantic description or a set of 3D geometries and can be compared with other 3D features such as 3D points or 3D geometries. Parameter types of trigger objects are represented by their IDs, names or semantic descriptions. The query may refer to any of these fields separately: ID (*event.paramtype.id*), semantics (*event.paramtype.semantics*), or to all of them at once (*event.paramtype*). Data types of these fields are identical to those presented for Interaction Interfaces.

The *condition* query element is a reference to types of object parameters used in interaction condition expression or to values of specific object parameters. Object parameter types can be referred in the same way as trigger object parameter types for the *event* query element, with an additional component (*source*) which indicates what interaction party is referenced. References to values of object parameters refer to exact parameters of exact objects and include a flag indicating if the query references a value before or after the event. This flag is optional and by default the reference returns the value after the event. The reason for such design of the default value is purely practical. In most cases developers are searching for objects that can behave in a particular way. The result of such behavior is described by a potential state of an interactive object after the interaction, so the value after the event is referenced more often.

The *action* refers to the same fields as the *condition* query element, with the restriction that parameter types and parameter values are taken from the description of the action.

Parameter reference that is used in condition and action query elements is composed of two parts: object identification and parameter type identification. Each parameter is bound to one of three possible interaction objects: trigger object, interacting object or the environment. The identification of an object whose parameters are referred in the query is accomplished with the use of one of three keywords: “*trigger*”, “*interacting*” or “*environment*”. The identification of a specific parameter for a given object relies on the fact that each object can have only one parameter of a specific type. Thus, the parameter identification is done by object parameter type ID.

```

value      = literal | xml

literal     = numeric_literal | string_literal

numeric_literal  = integer_literal | decimal_literal | double_literal

integer_literal  = [+ -]? digits

decimal_literal  = [+ -]? ( "." digits ) | ( digits ( "." [0-9]* )? )

double_literal   = (( "." digits ) | ( digits ( "." [0-9]* )? )) [eE] [+ -]? digits

string_literal   = ( "" ( char [^'] | escape_apos ) * "" )

digits          = [0-9]+

escape_apos     = ""\"

char            = [http://www.w3.org/TR/REC-xml/#NT-Char]

xml             = [http://www.w3.org/TR/REC-xml/#NT-element]

```

The value query element stands for a real value literal. The value can be either a literal representing a simple value or an xml which represents values with complex data types, for example, object color of x3d: IndexedFaceSet type.

The simple value (literal) may be numeric or textual. This is represented by *numeric_literal* and *string_literal* query elements. The *numeric_literal* represents integer (*integer_literal*), decimal (*decimal_literal*) or double (*double_literal*) number. The *string_literal* query element is composed of any Unicode characters (*char*) except the apostrophe (‘). The apostrophe character is used to enclose any *string_literal*. Any apostrophe inside the text has to be replaced by *escape_apos* element. The exact syntax for the char element is defined in <http://www.w3.org/TR/REC-xml/#NT-Char>.

The syntax of the xml element used for representing complex values is defined in <http://www.w3.org/TR/REC-xml/#NT-element>, and the syntax of a concrete data type depends on a data type XML schema included in a definition of a particular interactive interface.

4.3 Embedding IMQL

The MIM metadata has the highest expression power when combined with more general multimedia metadata standards, for example, the MPEG-7. Therefore, the IMQL has to be used in conjunction with query languages capable of processing these general metadata standards. The IMQL can be embedded in queries written in other query languages forming a complex query. Complex queries enable search systems to analyze standard metadata descriptions and interaction descriptions in a single process.

To provide universality of IMQL embedding rules, an IMQL query is embedded as a function. The *imql()* function takes object metadata URI (*object_metadata_URI*) and IMQL WHERE clause contents (*where_expr*) as arguments. The function returns Boolean value of true if given metadata match the specified IMQL *where_expr*. IMQL function syntax is the following:

```
imql_func = "imql" "("object_metadata_URI "," where_expr ")"
```

Embedding an IMQL query as a function allows including the IMQL in an arbitrary query language: from XML related XQuery, to RDF query language SPARQL, to SQL used for querying relational databases. Examples of such embedded queries can be seen in examples below and in Section 5.

4.3.1 Embedding IMQL in XQuery

In XQuery, the IMQL function is used to construct a comparison expression included in the WHERE clause of an XQuery query. The structure of the XQuery comparison expression permits the use of external functions.

The embedded IMQL filtering function is an implementation of such external function feature. The example below presents an XQuery query with embedded IMQL filtering function. The XQuery query uses library of objects described with MPEG-7 metadata and looks for objects created by a company called “Virtual Devices Inc.”. The IMQL is used to narrow down the selection of objects only to interactive objects that react to changes of the shape of a trigger object.

```
LET $obj_metadata := doc('http://www.3dlibrary.org/objects/objs.mpeg7')
FOR
  $m IN $obj_metadata/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  $obj_metadata/Mpeg7//CreationInformation/Creator/* = "Virtual Devices Inc."
  AND
  imql($m//Interactions,
    event.paramtype.id == "V-II.ShapeCustom"
  )
RETURN $obj_metadata/Mpeg7//MediaLocator/MediaURI
```

4.3.2 Embedding IMQL in SPARQL

Metadata represented as RDF documents can be queried with the use of SPARQL. This language designed for querying RDF data sources can be extended with IMQL

functionality by using IMQL function as a filter function in the SPARQL WHERE clause. Additionally to triple patterns used to construct SPARQL queries, the SPARQL query language provides also a FILTER pattern. The FILTER pattern can be included in the WHERE clause of SPARQL query. The filter defines a restriction on the group of solutions in which the filter appears. The example below presents a SPARQL query with embedded IMQL filtering function. The semantics of the query is the same as in previous example.

```
PREFIX objs: <http://www.3dlibrary.org/objects/>
SELECT ?objectURI
FROM <http://www.3dlibrary.org/objects/rdf.owl>
WHERE {
  objs:object objs:creator "Virtual Devices Inc." .
  objs:object objs:URI ?objectURI .
  objs:object objs:interactions ?interactions .
  FILTER imql(?interactions,
    event. paramtype.id == "V-II.ShapeCustom"
  )
}
```

4.3.3 Embedding IMQL in SQL

Also SQL, the third universal querying language, can be extended with the IMQL. The IMQL embedding method used for SQL queries follows the solutions presented for XQuery and SPARQL. The IMQL filtering function is used as an additional condition in the WHERE clause of an SQL query.

By default, it is assumed that the database relation attribute passed as the first argument of the IMQL filtering function contains a complete XML document of MIM metadata. However, it is allowed for query engines to act as a proxy and use MIM metadata decomposed into native database attributes and relations.

The example below presents a SQL query with embedded IMQL filtering function. The semantics of the query is the same as in the two previous examples.

```
SELECT obj.uri
FROM 3dlibrary.objects obj
WHERE
  obj.creator = "Virtual Devices Inc."
  AND
  imql(obj.interactions_metadata,
    event. paramtype.id == "V-II.ShapeCustom"
  )
```

Fig. 2 Sample virtual representations of cultural artifacts (*objects from ARCO project*)

5 Example

Consider a repository of 3D representations of cultural artifacts: sculptures, vases, tools, etc. (Fig. 2). Such objects are used in implementations of virtual museums [41] and each of these objects has a number of versions used in various scenarios [27]. Some of these object versions are static, and others are enhanced with additional behavior required for interactive virtual exhibitions. Creation of new interactive virtual exhibitions involves finding appropriate interactive 3D objects.

5.1 Interaction metadata construction

To allow finding interactive 3D objects by their interaction properties these objects have to be described with metadata that include information about their interactions. This can be done with MIM metadata. Consider a 3D representation of a sculpture, enhanced with a behavior that causes the 3D object to get bigger when approached by a viewer. The goal of such behavior is to present all details of the sculpture to the interested viewer. The MIM metadata of such object is the following.

The *interactive object* is the 3D representation of a sculpture. It has one interaction – ‘adjust size’. The size of the object depends on the distance from the *trigger object*, which is an avatar¹ of the viewer. To trigger the interaction the avatar has to be in the vicinity of the sculpture. The size of the sculpture increases when the avatar gets closer, or decreases when the avatar moves away. The exact size of the sculpture depends on the distance between the sculpture and the avatar.

The ‘adjust size’ interaction is based on the Visual Interaction Interface implemented by all interactive 3D objects and presented briefly in Section 3. The parameters used in this interaction include: *SizeBoundingBox*, *PositionCenter* and *ShapePrimitive*. Data types of the first two parameters are adopted from the X3D language. They are composed of three numerical values, each describing different dimension of an object bounding box or position. The third parameter represents an object shape. The value of the parameter is a textual symbol which identifies one of predefined shapes (e.g. cone, torus, avatar, etc.).

¹ Avatar: a graphical representation of a user.

The ‘adjust size’ interaction is executed for objects that are situated close to the sculpture and the interactive object (i.e. the sculpture) reacts to a change of the distance between the trigger object and itself. Therefore, the Event metadata element contains information about the interaction space, which includes the area around the sculpture, and the parameter representing the distance between objects – *PositionCenter*.

Listing 1 XML description of the Event metadata element

```
<mim:Event>
  <mim:InteractionSpace>
    <mim:Space>
      <x3d:Sphere radius="2"/>
    </mim:Space>
    <mim:Semantics>
      <mpeg7:FreeTextAnnotation>Spherical area around the object
    </mpeg7:FreeTextAnnotation>
    </mim:Semantics>
  </mim:InteractionSpace>
  <mim:Parameter typeId="V-II.PositionCenter"/>
  <mim:Semantics>
    <mpeg7:FreeTextAnnotation>Reacts on changes of position of objects in its vicinity
  </mpeg7:FreeTextAnnotation>
  </mim:Semantics>
</mim:Event>
```

The Condition metadata element contains an expression which verifies whether the trigger object has a shape of an avatar.

Listing 2 XML description of the Condition metadata element

```
<mim:Condition>
  <mim:Expression>
    <math:apply><math:eq/>
      <math:csymbol>S</math:csymbol><math:ci>avatar</math:ci>
    </math:apply>
  </mim:Expression>
  <mim:ExpressionArgument>
    <mim:ParameterValue typeId="V-II.ShapePrimitive" variableBinding="S"/>
  </mim:ExpressionArgument>
  <mim:Semantics>
    <mpeg7:FreeTextAnnotation>The interaction is executed only for avatar objects
  </mpeg7:FreeTextAnnotation>
  </mim:Semantics>
</mim:Condition>
```

The result of the ‘adjust size’ interaction, the new size of the sculpture, is described in the Action metadata element. This metadata fragment contains not only a specification of the modified parameter, but also a mathematical expression which approximates the new parameter value.

Listing 3 XML description of the Action metadata element

```

<mim:Action>
  <mim:SubAction>
    <mim:Parameter typeID="V-II.SizeBoundingBox:x">
      <mim:Result>
        <math:apply><math:times/>
          <math:csymbol>IS:x</math:csymbol>
          <math:apply><math:plus/><math:cn>1</math:cn>
          <math:apply><math:divide/>
            <math:apply><math:minus/><math:cn>2</math:cn>
            <math:apply><math:root/>
              <math:apply><math:plus/>
                <math:apply><math:power/>
                  <math:csymbol>TP:x</math:csymbol><math:cn>2</math:cn>
                </math:apply>
              <math:apply><math:power/>
                <math:csymbol>TP:y</math:csymbol><math:cn>2</math:cn>
              </math:apply>
            <math:apply><math:power/>
              <math:csymbol>TP:z</math:csymbol><math:cn>2</math:cn>
            </math:apply>
          </math:apply>
        </math:apply>
      </math:apply>
      <math:cn>3</math:cn>
    </math:apply>
  </mim:Result>
  <mim:ExpressionArgument>
    <mim:ParameterValue
      typeID="V-II.SizeBoundingBox" variableBinding="IS" object="interacting"/>
  </mim:ExpressionArgument>
  <mim:ExpressionArgument>
    <mim:ParameterValue
      typeID="V-II.PositionCenter" variableBinding="TP" object="trigger"/>
  </mim:ExpressionArgument>
</mim:Parameter>
</mim:SubAction>
<mim:SubAction>
  <mim:Parameter typeID="V-II.SizeBoundingBox:y">
    ...
  </mim:Parameter>
</mim:SubAction>
<mim:SubAction>
  <mim:Parameter typeID="V-II.SizeBoundingBox:z">
    ...
  </mim:Parameter>
</mim:SubAction>
</mim:Action>

```

5.2 Interaction metadata usage

Having a repository of interactive 3D objects described with interaction metadata, such as the interaction metadata presented above, it is possible to search for objects with required interaction properties. Consider a developer creating a new interactive exhibition for a

virtual museum. Instead of creating new instances of 3D cultural objects, the developer can search for existing interactive 3D objects that match interaction criteria of the new exhibition. The repository available to the developer contains 3D objects described with MPEG-7 metadata extended with MIM metadata. Metadata descriptions are saved in XML format and stored inside an Oracle database. The developer may build search queries using SQL or XQuery languages, either without or with embedded IMQL filtering function.

Information provided directly in the MIM metadata can be exploited in queries built without the IMQL filtering function. However, the use of IMQL makes such queries simpler and easier to implement. In the presented scenario, the developer may want to find objects that react to changes of position of other objects (Query 1), or objects that interact only with avatar shaped objects (Query 2), or objects that react by changing their size (Query 3). To fully compare complexity of queries with and without IMQL all the mentioned queries are implemented in SQL, SQL + IMQL, XQuery, and XQuery + IMQL.

Query 1.1 SQL

```
SELECT o.uri
FROM repository.virtual_objects o
WHERE
  CONTAINS (
    o.interactions,
    'V-II.PositionCenter INPATH(/Interactions//mim: Event//mim:Parameter/@typeID)'
  ) > 0
```

Query 1.2 SQL + IMQL

```
SELECT o.uri
FROM repository.virtual_objects o
WHERE
  imql(o.interactions,
    event.paramtype.id == "V-II.PositionCenter"
  )
```

Query 1.3 XQuery

```
LET $doc := doc('http://www.virtual-objects.org/ad-signs')
FOR
  $m IN $doc/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  fn:starts-with($m//Interactions//mim:Event//mim:Parameter/@typeID, "V-II.Position")
RETURN $/MediaLocator/MediaURI
```

Query 1.4 XQuery + IMQL

```

LET $doc := doc('http://www.virtual-objects.org/ad-signs')
FOR
  $m IN $doc/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  imql($m//Interactions,
    event.paramtype.id == "V-II.PositionCenter"
  )
RETURN $/MediaLocator/MediaURI

```

Query 2.1 SQL

```

SELECT o.uri
FROM repository.virtual_objects o
WHERE
  CONTAINS (
    o.interactions,
    'avatar INPATH(
      /Interactions//mim:Condition/mim:Expression//math:eq[
        ../math:csymbol = /Interactions//mim:Condition/mim:ExpressionArgument/
        mim:ParameterValue[
          @typeID = "V-II.ShapePrimitive"
        ]/@variableBinding]../math:ci
      )'
    ) > 0

```

Query 2.2 SQL + IMQL

```

SELECT o.uri
FROM repository.virtual_objects o
WHERE
  imql(o.interactions,
    condition.trigger.V-II.ShapePrimitive.value == "avatar"
  )

```

Query 2.3 XQuery

```

LET $doc := doc('http://www.virtual-objects.org/mpeg7.xml')
FOR
  $m IN $doc/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  $m//Interactions//mim:Condition/mim:Expression//math:eq[
    ../math:ci = "avatar"
  ]/../math:csymbol = $m//Interactions//mim:Condition/mim:ExpressionArgument/
  mim:ParameterValue[@typeID = "V-II.ShapePrimitive"]/@variableBinding
RETURN $/MediaLocator/MediaURI

```

Query 2.4 XQuery + IMQL

```

LET $doc := doc('http://www.virtual-objects.org/ad-signs')
FOR
  $m IN $doc/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  imql($m//Interactions,
    condition.trigger.V-II.ShapePrimitive.value == "avatar"
  )
RETURN $/MediaLocator/MediaURI

```

Query 3.1 SQL

```

SELECT o.uri
FROM repository.virtual_objects o
WHERE
  CONTAINS (
    o.interactions,
    'V-II.Size INPATH(/Interactions//mim:Action//mim:Parameter/@typeID)'
  ) > 0

```

Query 3.2 SQL + IMQL

```

SELECT o.uri
FROM repository.virtual_objects o
WHERE
  imql(o.interactions,
    action.interacting.paramtype.id ~= "V-II.Size.*"
  )

```

Query 3.3 XQuery

```

LET $doc := doc('http://www.virtual-objects.org/ad-signs')
FOR
  $m IN $doc/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  fn:starts-with($m//Interactions//mim:Action//mim:Parameter/@typeID, "V-II.Size")
RETURN $/MediaLocator/MediaURI

```

Query 3.4 XQuery + IMQL

```

LET $doc := doc('http://www.virtual-objects.org/ad-signs')
FOR
  $m IN $doc/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  imql($m//Interactions,
    action.interacting.paramtype.id ~= "V-II\Size.*"
  )
RETURN $/MediaLocator/MediaURI

```

To confirm that the use of IMQL makes queries simpler and easier to implement, selected Halstead metrics [24] were calculated for each of the presented queries. Calculation of Halstead metrics is based on the following numbers: number of distinct operators ($n1$), number of distinct operands ($n2$), number of total operators ($N1$), and total number of operands ($N2$). Operators are elements of the query that are defined in the language specification (reserved words, operators). Operands are other elements of the query (identifiers, type names, constants). For example, the sets of operators and operands for the Query 1.1 are the following (separated with space):

- operators: SELECT . FROM . WHERE CONTAINS () , , ' INPATH () / / / / @ >
- operands: o uri repository virtual_objects o o interactions V-II. PositionCenter Interactions mim: Event mim: Parameter typeID 0

Consequently, the numbers required for calculating Halstead metrics of the Query 1.1 are the following:

- distinct operators ($n1$)=12
- distinct operands ($n2$)=11
- total operators ($N1$)=18
- total operands ($N2$)=13

In the presented queries, the difference between queries with and without IMQL is only in the WHERE clause. Therefore, it is enough to calculate Halstead metrics only for the contents of the WHERE clause. The numbers of operators and operands, and calculated Halstead metrics of experimental queries are presented in Table 1.

The calculation results presented in Table 1 reveal that queries with IMQL have better metrics. This confirms that the use of IMQL makes queries simpler and easier to implement. For simple queries (Query 1 and Query 3) the improvement is not very large, but noticeable. The real power of IMQL can be seen for more complex queries (Query 2). In such a case all measures for queries with embedded IMQL are far better than for standard SQL and XQuery queries.

Sometimes, the developer might have some very specific requirements, which cannot be verified using only information provided directly in the metadata. For example, the new interactive virtual exhibition may require objects that do not react with other objects when they are 3 m away (Query 4), or allow only objects that will not get larger than 2 m (Query 5). Such information is not provided directly in the metadata, but it may be calculated using mathematical expressions included in MIM metadata. Standard query languages like SQL or

Table 1 Halstead metrics of experimental queries

	n1 distinct operators	n2 distinct operands	N1 total operators	N2 total operands	N Program length	n Vocabulary size	V Program volume	D Difficulty level	E Effort to implement
Query 1									
SQL	10	8	13	8	21	18	87,6	5,0	437,8
SQL + IMQL	6	4	6	4	10	10	33,2	3,0	99,7
XQuery	8	6	10	6	16	14	60,9	4,0	243,7
XQuery + IMQL	7	4	7	4	11	11	38,1	3,5	133,2
Query 2									
SQL	14	15	29	17	46	29	223,5	7,9	1772,8
SQL + IMQL	8	4	8	4	12	12	43,0	4,0	172,1
XQuery	8	13	25	16	41	21	180,1	4,9	886,6
XQuery + IMQL	9	4	9	4	13	13	48,1	4,5	216,5
Query 3									
SQL	9	8	12	8	20	17	81,7	4,5	367,9
SQL + IMQL	7	3	7	3	10	10	33,2	3,5	116,3
XQuery	8	6	10	6	16	14	60,9	4,0	243,7
XQuery + IMQL	8	3	8	3	11	11	38,1	4,0	152,2

XQuery do not have the ability to evaluate MathML expressions, but it is possible to formulate an IMQL filtering query with appropriate requirements. The Query 4 is implemented using SQL+IMQL, the Query 5 using XQuery + IMQL.

Query 4 SQL + IMQL

```
SELECT o.uri
FROM repository.virtual_objects o
WHERE
  imql(o.interactions,
    event.space >> "<x3d:Sphere radius='3'/>"
  )
```

Query 5 XQuery + IMQL

```
LET $doc := doc('http://www.virtual-objects.org/mpeg7.xml')
FOR
  $m IN $doc/Mpeg7/Description/MultimediaContent/Multimedia
WHERE
  imql($m//Interactions,
    action.interacting.V-II.SizeBoundingBox:y.value.after <= 2
  )
RETURN $/MediaLocator/MediaURI
```

Queries presented above illustrate that IMQL facilitates the construction of even complex queries and makes it possible to build queries that use interaction properties with arbitrary values that cannot be simply compared with values provided in the metadata.

6 Conclusions

The overall interaction metadata solution composed of the Multimedia Interaction Model, the Interaction Interface concept, and the Interaction Metadata Query sub-Language provides tools that cover the most important aspects of composition and exploitation of interaction metadata of 3D objects. The structure provided by the MIM model is general enough to represent information about almost any interaction. The Interaction Interface concept allows defining new interaction parameter types, making the metadata solution extensible and applicable to new specific domains. The Interaction Metadata Query sub-Language enables efficient use of information embedded in interaction metadata. These tools can be used to enhance the descriptive power of existing metadata standards such as MPEG-7 and allow them to describe not only general, semantic, production, technical, and management information, or object geometry, but also to describe object behavior and analyze potential states of an interactive object.

The Interaction Metadata Query sub-Language presented in this paper facilitates creation of queries that take into account object interaction properties and permits building queries with requirements related to potential states of an interactive object. The IMQL is characterized by extensibility and independence, and was tailored to make it easier to exploit information provided by the MIM interaction metadata. Extensibility of the IMQL is assured by the ability to build queries that use arbitrary interaction parameter types defined in new interaction interfaces, so the IMQL maintains the extensibility introduced by the Interaction Interface concept. Independence of the IMQL was achieved by providing a self-contained definition and general embedding rules, which allow using interaction metadata queries as standalone queries or as additional filtering rules in queries written in general query languages such as SQL, XQuery, or SPARQL.

The experiment presented in this paper confirms that the use of IMQL makes interaction metadata queries simpler and easier to implement. Interaction metadata queries written using just general query languages (SQL, XQuery) require more complex expressions which makes them error prone and entails more effort. Moreover, tight integration with the Multimedia Interaction Model makes it possible to exploit information that is not provided directly in metadata but can be calculated using mathematical expressions included in MIM Condition and Action elements. Such calculations enable analyzing potential new values of object interaction parameters (i.e. potential object states), which is not supported by general query languages.

The domain of interaction metadata is an emerging research area. The MIM concept may be a starting point for several research initiatives. The most natural area of research is development of new search engines focused on object interaction properties and specialized object comparison and matching algorithms that could be used in conjunction with the *almost equal* IMQL operator. Additionally, the interaction structure employed in the MIM model is general and may be used also to describe almost any interaction, not only interactions of 3D objects. For example, it could be used to create documentation of social interactions that happen among a group of people. Such description, based on the MIM model, could be later used for various social analyses. In such a case the IMQL is a great tool for profiling – selecting people with particular interaction patterns. Therefore, with more research, the presented approach and the IMQL language could become very useful, not only for interactive 3D objects.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

1. 3D Model Search Engine (2012) Princeton shape retrieval and analysis group. <http://shape.cs.princeton.edu/search.html>
2. Apaydin A, Çelik D, Elçi A (2010) Semantic image retrieval model for sharing experiences in social networks. 2010 IEEE 34th annual computer software and applications conference workshops, pp 1–6, 2010
3. Berners-Lee T, Fielding R, Masinter L (2005) RFC 3986, Uniform Resource Identifiers (URI): Generic syntax. IETF, 2005
4. Bilasco IM, Gensel J, Villanova-Oliver M, Martin H (2005) On indexing of 3D scenes using MPEG-7. Proceedings of ACM Multimedia 2005, Singapore, 2005, pp 471–474
5. Bilasco IM, Gensel J, Villanova-Oliver M, Martin H (2006) An MPEG-7 framework enhancing the reuse of 3D models. Proceedings of Web3D Symposium 2006, Columbia, Maryland USA, 2006, pp 65–74

6. Bry F, Patrânjan P (2005) Reactivity on the web: paradigms and applications of the language XChange. Proceedings of the 2005 ACM symposium on applied computing, Santa Fe, New Mexico USA, 2005, pp 1645–1649
7. Cellary W, Walczak K (2012) Interactive 3D content standards. Interactive 3D multimedia content - models for creation, management, search and presentation. Springer, London, pp 13–35
8. Chmielewski J (2008) Metadata model for interaction of 3D Objects. The 1st International IEEE Conference on Information Technology, Gdansk, Poland, May 18–21, 2008, ed. Gdansk University of Technology, Andrzej Stepnowski, Marek Moszyński, Thaddeus Kochański, Jacek Dąbrowski, pp 313–316
9. Chmielewski J (2008) Interaction descriptor for 3D objects. The IEEE International Conference on Human System Interaction, Cracow, Poland, May 25–27, 2008, IEEE, pp 18–23
10. Chmielewski J (2008) Interaction Interfaces for unrestricted multimedia interaction descriptions. In: Gabriele Kotsis, David Taniar, Eric Pardede, Ismail Khalil (eds) The 6th international conference on advances in mobile computing and multimedia MoMM 2008. Association for Computing Machinery, Linz, pp 397–400
11. Chmielewski J (2008) Architectural modeling with reusable interactive 3D objects. The 22nd European Conference on Modelling and Simulation ECMS 2008, June 3–6, 2008, Nicosia, Cyprus, pp 555–561
12. Chmielewski J (2009) Building extensible 3D interaction metadata with interaction interface concept. The 1st international conference on knowledge discovery and information retrieval KDIR 2009, Funchal - Madeira, Portugal, October 6–8, 2009, Institute for Systems and Technologies of Information, Control and Communication, pp 160–167
13. Chmielewski J (2009) Ontology enabled metadata for interactive 3D objects. Information systems architecture and technology - service oriented distributed systems: concepts and infrastructure. Wrocław University of Technology Press, Wrocław, pp 59–69
14. Chmielewski J (2012) Describing interactivity of 3D content. Interactive 3D multimedia content - models for creation, management, search and presentation. Springer, London, pp 195–221
15. Croquet (2010) The croquet consortium. <http://www.opencroquet.org/>
16. Demiris A, Diamantakos G, Walczak K, Reusens E, Kerbiriou P, Klein K, Garcia C, Marchal I, Wingbermhle J, Boyle E, Cellary W, Ioannidis N (2001) PISTE: mixed reality for sports TV. Proceedings of the International Workshop on Very Low Bitrate Video Coding 2001, October 2001, Athens, Greece
17. Dittrich KR, Gatzju S, Geppert A (1995) The active database management system manifesto: a rulebase of ADBMS features, rules in database systems. Lecture notes in computer science 985:1–17
18. Falcidieno B, Spagnuolo M, Alliez P, Quak E, Vavalis E, Houstis C (2004) Towards the semantics of digital shapes: the AIM@SHAPE approach. European Workshop on the Integration of Knowledge, Semantic and Digital Media Technologies November 25–26, 2004, Royal Statistical Society, London
19. Funkhouser T, Min P, Kazhdan M, Chen J, Halderman A, Dobkin D, Jacobs D (2003) A search engine for 3D models. ACM Trans Graph 22(1):83–105
20. Ghosh H, Chaudhury S, Kashyap K, Maiti B (2007) Ontology specification and integration for multimedia applications. Ontologies: a handbook of principles, concepts and applications in information systems, pp 265–296, Springer
21. Goczyła K, Grabowska T, Waloszek W, Zawadzki M (2005) The cartographer algorithm for processing and querying description logics ontologies. Proceedings of the Third International Atlantic Web Intelligence Conference, Advances in Web Intelligence AWIC 2005, Lodz, Poland, 6–9 June 2005, pp 163–169
22. Google Earth (2011) Google. <http://earth.google.com/>
23. Haas W, Thallinger G (2005) SALERO - semantic audiovisual entertainment reusable objects. The 2nd European Workshop on the integration of knowledge, semantics and digital media technology, EWIMT 2005, November 30–December 1, 2005, London, UK, pp 339–340
24. Halstead MH (1977) Elements of software science (Operating and programming systems series). Elsevier Science Inc, New York
25. Manjunath B, Salembier P, Sikora T (eds) (2002) Introduction to MPEG-7. Wiley, West Sussex
26. Papamarkos G, Poullovassilis A, Wood PT (2003) Event-condition-action rule languages for the semantic web. Workshop on Semantic Web and Databases, 2003
27. Patel M, White M, Walczak K, Sayd P (2003) Digitisation to presentation - building virtual museum exhibitions. Proceedings of international conference on vision, video and graphics, Bath, UK; Editor: Peter Hall and Phil Willis; pp 189–196, July 2003
28. Project Looking Glass (2009) Sun microsystems. http://sun.com/software/looking_glass/
29. Pulli K, Aarnio T, Miettinen V, Roimela K, Vaarala J (2007) Mobile 3D graphics: with OpenGL ES and M3G. Morgan Kaufman, 2007

30. Rakkolainen I, Timmerheid J, Vainio T (2000) A 3D city info for mobile users. Proceedings of the 3rd international workshop in intelligent interactive assistance and mobile multimedia computing (IMC2000), November 9–10, 2000, Rockstock, Germany, pp 115–212
31. Solomon J (2007) The science of shape: revolutionizing graphics and vision with the third dimension. ACM Crossroads Magazine, Volume 13 Issue 4, June 2007, New York, USA
32. Spencer H (1994) regex — POSIX 1003.2 regular expressions. Unix Manpages, chapter 7, 1994
33. Thallinger G, Kienast G, Mayor O, Cullen C, Hackett R, Jose J (2009) SALERO: semantic audiovisual entertainment reusable objects. International Broadcasting Conference IBC 2009, September 2009, Amsterdam, The Netherlands
34. Thome B, Gawlick D, Pratt M (2005) Event processing with an oracle database. Proceedings of the 2005 ACM SIGMOD international conference on management of data, Baltimore, Maryland USA, 2005, pp 863–867
35. W3C Extended Backus-Naur Form (EBNF) notation (2008) Extensible markup language (XML) 1.0, W3C Recommendation. <http://www.w3.org/TR/REC-xml/#sec-notation>
36. Walczak K (2009) Configurable virtual reality applications. Poznan University of Economics Press, Poznan
37. Walczak K, Cellary W (2003) X-VRML for advanced virtual reality applications. IEEE Computer 36 (3):89–92
38. Walczak K, Cellary W (2005) Architectures for interactive television services. e-Minds. Int J Hum Comput Interact 1(1):23–40
39. Walczak K, Cellary W, White M (2006) Virtual museum exhibitions. IEEE Computer 39(3):93–95
40. Walczak K, Cellary W, Chmielewski J, Staniak M, Strykowski S, Wiza W, Wojciechowski R, Wojtowicz A (2004) An architecture for parameterised production of interactive TV contents. Proceedings of the 11th international workshop on systems, signals and image processing; international conference on signals and electronic systems IWSSIP 2004, 13–15 September 2004, Poznań, Poland, pp 465–468
41. Walczak K, Wiza W (2007) Designing behaviour-rich interactive virtual museum exhibitions. The 8th international symposium on virtual reality, archaeology and cultural heritage VAST 2007, Brighton (UK), November 26–30, 2007, pp 101–108
42. Walczak K, Wojciechowski R, Cellary W (2006) Dynamic interactive VR network services for education. Proceedings of the 13th ACM symposium on Virtual Reality Software and Technology - VRST 2006, November 1–3, 2006, Limassol, Cyprus, pp 277–286
43. White M, Mourkoussis N, Darcy J, Petridis P, Liarokapis F, Lister P, Walczak K, Wojciechowski R, Cellary W, Chmielewski J, Stawniak M, Wiza W, Patel M, Stevenson J, Manley J, Giorgini F, Sayd P, Gaspard F (2004) ARCO - An architecture for digitization, management and presentation of virtual exhibitions. Proceedings of the Computer Graphics International CGI 2004, Hersonissos, Crete, Greece, pp 622–625
44. Wiza W, Walczak K, Cellary W (2004) Periscope - a system for adaptive 3D visualization of search results. Proceedings of the ACM SIGGRAPH, 2004, pp. 29–39
45. Wojciechowski R (2008) Modeling interactive educational scenarios in a mixed reality environment. Dissertation, Gdansk University of Technology
46. Zhou X, Zhang S, Cao J, Dai K (2004) Event condition action rule based intelligent agent architecture. J Shanghai Jiaotong University 38(1):14–17

Jacek Chmielewski received the M.Sc. degree in Electronics and Telecommunication from the Poznan University of Technology in 2001. During the studies he participated in commercial projects concerning the development of Internet applications. From 2000 to 2001 he was a network administrator in the Regional Government IT Center in Poznan. Since 2001 he has been with the Department of Information Technology at the Poznan University of Economics. He received the Ph.D. degree in Computer Science in 2008 from the Technical University of Gdansk. His current research field is the human side of the Internet of Things (smart multi-device applications, distributed user interfaces, natural user interfaces). Other research interests include mobile technology and its business applications, mobile and web applications, and HCI issues. He runs several courses related to computer, web and mobile programming, and one about interfaces for the Internet of Things. He is a reviewer of research projects and journal and conference papers.

Currently dr Jacek Chmielewski manages the research in the “Ubiquitous mobile user interfaces for industrial monitoring and control applications” project funded under the ABB Software Research Grant and he leads the “User interface personalization and adaptation” team in the project “New Information Technologies for Electronic Economy and Information Society based on SOA paradigm”, funded within the national “Innovative Economy” Operational Program. In the past dr Jacek Chmielewski was working on several international and national research projects (Web 2.0 ERC, MM4ALL, ARCO, PISTE, iTVP, ...), where he collaborated with teams from EU and USA.