MANAGED CODE INTEROPERABILITY IN DIGITAL EARTH TECHNOLOGY

Kolar, Jan. Centre for 3DGI, Aalborg University, Denmark
Kolar, Jan. Charles' University in Prague, the Czech Republic

Keywords: interoperability, managed code, geo-visualization, GRIFINOR

Summary

This article is about interoperability in relation to the digital earth technology. The focal point of the article is on the managed code approach, which is qualitatively different from traditional solutions based on implementations of standard exchange data formats. The article is presented from the standpoint that a solution to interoperability is efficient when data and software can coexist as much as possible. This is in contrast to having data formats dictating to software what is possible and what is necessary in order to inter-operate between systems. Such a hypothesis produces an interesting question; is it generally acceptable as well as technologically possible to have data and software coexisting on the same level? This article provides arguments for a positive answer regarding systems where interoperability is a crucial question. Arguments for managed code based solutions are provided in this matter. Since digital earth is meant to be an interactive media technology---a geo-visualization counterpart to the web technology-, interoperability with managed code is a candidate that is worth consideration. In the text is described an example exploiting the managed code of Java in order to address interoperability in GRIFINOR platform for geo-visualization in 3D. Based on the example, some specific features of managed code are pointed out.

1 Managed code

Interoperability is one of the important issues in the design and development of information systems in general. With the evolution of geographic systems turning to the vision of digital earth technology (DE) (Gore, 1998) we face an overwhelming number of features and phenomena for which we need suitable digital representations and modeling. In DE development we encounter further increase in the already high diversity of phenomena addressed by the contemporary geographic applications, which in context of DE may encompass nearly everything we experience on our planet. Due to this diversity of modeled phenomena, interoperability remains a burning problem in this domain of development. One consequence of more and more geographic applications addressing different spatial coverage, various descriptions of environmental aspects or specific analysis, is that definitions of *exchange data formats* (EDF) become extensive. This leads to their frequent revisions, costly implementations and number of independent yet redundant initiatives to define a new, "better" standard.

Managed code (MC) seems to be more suitable in these complex scenarios, when chances are high that a slightly different solution is necessary for many cases addressing the same subject from similar yet diverse perspectives. MC may also address the situation when we are sure only about basics of the representation but the optimal solution requires more research. The term "managed code" is borrowed from Microsoft's terminology for computer instructions executed by Common Language Runtime (CLR) *virtual machine* of .NET initiative. However in this article MC refers to a concept rather than to the concrete .NET technology. The main point about MC is its ability to be stored and passed between

systems like data using EDF, with one important difference that MC may contain executable functionality in addition to mere data. This is an integral concept for many technologies, such as Java technology led by Sun, Flash technology by Macromedia or the Apple universal binary. All these implementations (together with other technologies based on this concept) are referred to as MC-based technology in this text.

MC is created by pre-translating (compiling) source code to an intermediate representation, which is also known as *byte-code*. This is a binary representation of both a data representation and an executable program, designed to be executed by a virtual machine rather than by a dedicated platform architecture or hardware (although the possibility to build a dedicated hardware exists). This means that MC may be portable among computer architectures. In some technologies (Apple binary) MC may contain also machine code, which makes MC optimal for execution but dependent on the platform architecture. For system development, it is important that MC performs better compare to interpreted script languages, which originates from the pre-translating of entire files rather than interpreting each line separately at run-time. This also allows blocks of MC being stored in separate files, which may be exchanged in the same manner as files using a standard EDF. Therefore we can have many files of MC, which can store only data, only a program module or both.

The goal of this paper is to better explain the features of MC in the context of interoperable systems and to provide an example related to DE technology. The application of MC as a solution to interoperability in the domain of geographic systems has been addressed earlier by Vckovski (1998), by pointing out a number of relevant features of Java technology in this matter. This article contributes by focusing on differences between EDF and MC. The rest of the article is organized as follows; in the next section a conceptual comparison between EDF and MC is presented. It is argued that EDF-based solutions are less convenient for interoperability in complex scenarios such as DE technology. The comparison is based on an evaluation of selected aspects that are relevant in development of interoperable systems in general. Section 2 provides an example of definition and creation of MC content in the GRIFINOR (Kolar, 2006) platform for geo-visualization. GRIFINOR is used here as an example of a visual information technology that provides MC-based solution to interoperability. The properties of the MC content provided by the example are described before conclusions in Section 3.

2 Comparison of Approaches to Interoperability

Standard EDF have been proven convenient or even necessary for elementary data chunks processed by computer systems. For example the representation of numbers such as integers or floating-point numbers with float decimal point may be considered as such elementary chunks of data for which it is convenient to have a standard EDF. For the same reason there are no serious discussions whether standard character-sets such as ASCII or UTF are necessary. These EDF are convenient because there is enough clear, convincing and logical arguments for how to represent, let say, an integer, in terms of bits and bytes. Regarding representation of roads, terrain or even primitive geometries the situation is far more complex and variable depending on application. The relatively simple and fundamental concepts for integer EDF apply to many parties using numbers in their systems anyway, and the agreement on their unified representation in form of EDF brings crucial support to interoperability, which is one important benefit brought by such a standard in this case. Therefore the challenge behind adoption of low level EDF is primarily administrative.

The challenges of standard EDF changes when we consider more complicated representations, e.g. raster images, vector geometry, fonts, video etc. These types of data might still seem quite basic but concepts

behind them may become very complicated. For example EDF for video files are very complicated. Additionally there may exist many possible alternatives for such a representation, none of which is the best for all cases. In such a situation it becomes particularly challenging to reach an agreement between many parties.
A consequence of this is that the need or convenience for a standard EDF shifts, and other political and economical factors enter the context. As a result we have a broad spectrum of EDF for raster images, video files as well as other types of files addressing advanced representations. This heterogeneity dissolves the main idea in the EDF approach to interoperability based on the following reasoning: if every system will be able to read and write data in accord with the specification of a standard EDF, an interoperability between the systems will be possible through mere exchange of the data in EDF.

The boundary when MC becomes more suitable for interoperability than EDF is only hypothetical. But we believe it is somewhere around the level of complexity for EDF addressed in the previous paragraph. Before comparing EDF and MC in terms of interoperability it should be stressed that the two are NOT on the same level of data definition. In fact each MC technology has certain EDF understood by virtual machine. The only thing that can change is the structure within the given meta-EDF. This is similar in concept to XML as all XML-based applications have one meta-EDF. Actually MC has several analogies with XML in terms of interoperability and data format, as both are meta-standards. An important property that MC has and XML does NOT is that MC is coupled with its virtual machine, which is part of the standard and allows to exchange functionality together with the data. This is also what differentiates the MC from the EDF approach in general. The consequence of this is that any application of MC may be regarded as a standard (or an extension of the standard) at the moment of its creation. This is not true of XML applications which become real standards after they are used by number of parties and after a software that uses the standard is devebped. Before that any XML application has only theoretical value.

2.1 Evaluation of Interoperability Factors

We provide conceptual comparison between EDF and MC based on nine factors relevant to interoperability. The factors evaluated in Table 1 are; 1) heterogeneity 2) extensibility 3) parsing 4) functionality exchange 5) openness, 6) programming language, 7) hardware architecture, 8) security and 9) storage.

Table 1. Factors of interoperability.

Heterogeneity: EDF are prone to heterogeneity---every new EDF standard raises the difficulty to deliver interoperable systems by having more EDF to consider. In contrast MC-based solutions basically eliminate heterogeneity when a single MC technology is deployed. For example MC-based systems may be interoperable within .NET technology but NOT with MC-based solution made in Java. However, heterogeneity of MC technologies is negligible compare to the one we face with the EDF approach.

Extensibility: MC has advanced properties in this matter. On the other hand XML is a good example that demonstrates the extensibility of EDF. As a meta-EDF, XML is used indirectly by systems. Instead systems use applications of XML that define a concrete representation, e.g. XML application, for representation of roads. For interoperability is crucial the extensibility of the road XML application, NOT of the XML meta-EDF. That kind of extensibility, however, is possible only via new versions of the road EDF, which would always enforce changes in the system implementation, which is always a problem for interoperable systems. MC allows for morphing or evolution of an existing content over time without re-implementation. With smart deployment of MC capabilities, standards may evolve over a longer period of time. Since MC approach allows for straightforward conversion to new solutions, the best currently

available solution will become a real standard automatically due to its qualities.

Parsing: Every EDF has to be parsed before the data encapsulated in that format can be actually processed. With MC parsing can be avoided completely even on the level of system development. This is considered the crucial difference between using EDF and MC in terms of development of interoperable systems. Whether parsing will be actually avoided using MC depends on the design of the system. Not all systems based on MC technology avoid parsing automatically, but it is one of the main advantages of MC that is worth exploiting. It should be stressed that parsing is an extensive burden on system development because what EDF can be parsed also dictates what data-sets the system can possibly deal with.

Functionality exchange: Functionality is an important aspect of interoperability . This is often neglected in the traditional EDF-based approach, which is based on strict separation of data from functionality. Per dictionary definition, the term "interoperability" means "ability to work or function together", in addition to exchange data or information. In this respect EDF support interoperability but cannot be regarded as being the solution for interoperability. MC-based solutions allow an exchange of functionality along with the data in a uniform manner and thus can directly address interoperability. We consider this as the main difference between EDF and MC solutions in terms of interoperability.

Openness: Interoperability between two (or more) systems is possible under the condition that it is known at both ends how the system on the other end works. This must be fulfilled at least to the extent covering the subject of interoperability. For example if one system wants to print using another system, it has to "know" messages and their format that should be sent along with the data and what to expect in response. What else the other system can do is unimportant if printing is the only subject of interoperability . Therefore it is feasible to deploy "open" protocols and EDF for system components that are intended to inter-operate. Many (not all) standard EDF are open, which allows for development of compliant and interoperable applications. The only standard MC technology that is "open" at the time of writing this is Java technology (available under GPL license) developed by a community led by Sun.

Programming language: As mentioned above, a typical feature of EDF is the strict separation of data from the system functionality. Since the system functionality is always implemented using a concrete programing technology, one can say that EDF is independent of the programing language. For example documents in HTML format can be used by software written in Fortran the same way as by software written in Python.
MC based solutions are more restrictive in this respect and at least the part of the interoperable system dealing with communication must be written in a language supported by the compiler for the virtual machine. This might be a limitation for the development of some interoperable systems.

Hardware architecture: The fact that EDF are typically independent of any programing language is often projected in the fact of EDF being often independent of computing platform too. On the other hand MC based solution can run only on platforms for which an implementation of the virtual machine exists. Although Java technology is available for a large variety of hardware architectures, including all major platforms, the limitation regarding platform exist. This, together with the limitation regarding programming language, is deemed to be main disadvantage of the MC-based solution.

Security: Both EDF and MC approaches to interoperability can address security through encryption of the data. EDF allows only this way to address security. This means that security of the inter-operation between systems can be ensured only indirectly by EDF, because the operation is performed by the

system independently of the data. In these terms MC is more powerful than the EDF approach since MC can also enforce a secure operation with the data itself, which might be advantage. On the other hand this power opens a possibility for insecure or even dangerous functionality within MC-based solutions in situations when it is used improperly or with insufficient care.

Storage: Although storage is remotely related to interoperability, it is worth mentioning that MC and EDF approaches are NOT equivalent in this manner. While EDF can be either binary (JPEG) or human readable (KML) the byte-code of MC solution is always binary. The reason for trying to view content of files by users is important when we need to figure out more about the EDF itself, access the actual data or both. Human readable EDF is necessary in situations when corresponding software is unavailable or the EDF cannot be determined, which happens relatively often in practice. Byte-code, on the other hand, is optimized for execution by the virtual machine. Although this is relatively efficient in terms of space, it cannot be read by humans directly. However, it is readable in a standard manner as any other part of the system, using the same MC technology. MC technology also usually provides a possibility to store (serialize) objects in a human readable form, but it cannot be considered a typical form of MC storage.

3 Implementation

In order to demonstrate an interoperable application based on MC, we elaborate a simple example using GRIFINOR platform (Kolar, 2006). GRIFINOR is a development effort within the field of DE technology, it is based on geo-centric coordinate system, its primary content is 3D representation of geographic features, which we call a *model-map* (Kjems and Kolar, 2005) and it provides users with ability to explore information on Earth by means of geo-visualization of the model-map. The model-map content in GRIFINOR is exclusively composed of Java objects, which are exemplification of MC. The design of GRIFINOR was made with an MC-based interoperability in mind. A solution to this is ensured by requiring every content object being derived from a single, abstract definition. In formal terms this means that a content object must be an instance of a custom class that extends the seminal "GO" class, example of which is depicted on line eight in Figure 1.

It should be stressed that primary mission of content classes in GRIFINOR is to deliver geometry that correspond to the shape of geographic features they represent. GRIFINOR currently provide four geometric primitives: point; line array; indexed triangular mesh and indexed triangle strip. These representations can be found in various visualization technologies (Schroeder et. al., 2002). This is basically all to the requirements for developing DE applications using GRIFINOR: Java technology, extension of the seminal class and providing geometric primitives. Under these rules any model-map content regardless its data representation including any later improvements or new alternative representation, regardless any analytical functions defined on the content or any other functionality associated with the content will remain interoperable with GRIFINOR platform. This alone does not make GRIFINOR better DE technology but it provides a robust ground for development of interoperable DE solutions.

3.1 Class Definition

Let's consider a content class that defines data for a shape of a road element. The main role of each instance of the road element class would be to display its shape at corresponding location in the virtual space as depicted in Figure 5. The result allow for various application about roads based on user interaction with the model-map. Using GRIFINOR as well as with any MC-based solution this can be achieved by infinite number of solutions without hampering interoperability. In GRIFINOR we can

distinguish between three types of content objects depending on their properties determined by definitions of the class. Note however, that these properties, which aredescribed in the following text, are not exclusive to each other and can be combined in other class definitions.

Figure 1. Definition of instantiated content class.

Instantiated content objects are the basic and simplest type. Objects of this type contain data for all the coordinates and topology required by the selected geometricprimitives. As mentioned earlierthese are necessary to describe the shape of the represented feature. The definition of a road element as an instantiated content class may look as depicted in Figure 1. The geometry of an object is kept in the field "gm" defined on line nine, and its content must be provided when the object is created as defined on line ten. From lines fourteen and fifteen we can also see that extension of the seminal class "GO" induce definition of the "convert" method, which is used when the object is visualized. The last thing enforced by the seminal class "GO" in all of its derivative classes is definition of the object's referencing point and its level of visual magnitude on line eleven. This is on line eleven and both of these quantities are provided at initialization of an instance on line ten.

Procedural content objects do not contain all the geometry data. Instead they contain set of optional parameters, from which the geometry can be constructed. The procedure (algorithm) how to construct the resulting geometry is part of the class definition. A possible definition of a road element as a procedural class is depicted in Figure 2. The change from the previous definition is the custom method "roadElementProcedure" which computes the geometry when it is needed for visualization. The field "gm" that contains the geometry is NOT stored, which denoted by "transient" on the line fifteen. Only the parameters are stored and therefore must be provided when the object is created on the line sixteen.

Figure 2. Definition of procedural content class.

Referenced content objects contain insufficient amount of data to provide geometry from its own resources. Instead they include data and functionality to request necessary data from external sources. The procedure how to obtain the data is part of the class definition. This can be a query to a relational database system (RDBMS) or retrieving data from a URL over a network or from a local disk.
An example definition of a road element as a referenced content class is depicted in Figure 3. The new definition compare to the procedural case starts on the line thirty in Figure 3. This method is used to retrieve data from RDBMS and store them in the field "element" from the line 29, which is NOT stored with the object. When instantiated, this field contains enough data for computation of the geometry. This means that the only data stored with this object is the identifier of the road element in the RDBMS.

Figure 3. Definition of referenced content class.

3.2 Creation of Objects

Definition of the classes is important part ofMC-based solution. More important than in EDF approach because there the definition is often given by a standard that we want to keepunchanged. However, in both MC and EDF approach, making of the content remains equally important. --- Definitions of classes cannot be visualized in GRIFINOR, only the objects can be shown and explored in GRIFINOR. In Figure 4 is depicted how to createconcrete instances of the three classes defined above. In this example we

assume having a database in RDBMS from which an axis of each road segment can be obtained in geographical coordinates via its identifier, e.g. "s1", "s2" etc. The complexity of creating objects is usually inversely proportional to the complexity of their class definition. The instantiated content classes are the simplest to define (see Figure 1), but their content is close to what graphical hardware can deal with, which means that the input data must be pre-processed accordingly before an object can be created and stored. On the other hand, the referenced or the procedural objects have relatively more complex class definitions (see Figure 2 and 3), because they must address what is omitted during construction. In other words, more is done as pre-processing more is saved at run-time, but the processing must happen at some point. This is a traditional trade-off in computer systems development, but MC-based solution allows to take the best choice for special cases of broad spectrum of applications without giving interoperability at stake.

The instantiated content object is created on line 44 in Figure 4. All up to line 27 is what it worth to pre-process the data available from RDBMS into a triangle strip, which represents geometry of the particular instance of the road element. The procedural content object is then created on line 51. The data are retrieved from the RDBMS at the line 47, but the pre-processing is omitted. Therefore lines 31 through 43 would be part of the method defined on line 25 in Figure 2. Finally the referenced content object is created on line 57 in Figure 4. The identifier "s3" of the road element in RDBMS is the only data the object is provided with about its shape. As defined in the method on line thirty in Figure 3 the data will be retrieved from RDBMS by the object itself always when its geometry will be necessary for visualization.

The three created objects were visualized in GRIFINOR together with some other objects as shown in Figure 5. From the figure is apparent that with GRIFINOR it might be impossible to recognize different representations of features even though they might have very different properties. As addressed in this example the referenced objects are the most flexible because they can refer to external resources that might be operated by other external systems such as RDBMS, which have nothing to do with GRIFINOR. Whenever the source data are changed in RDBMS their next visualization of the corresponding object in GRIFINOR viewer show the new data as long as the procedure will work with the new data. Both procedural and referenced objects can be very efficient in terms of storage space, because they might need no data (when everything for a request is part of the class definition) for retrieving large amount of data from external sources. This, on the other hand, might be inefficient in terms of time because an extra communication and extra processing must be performed for each object.

Figure 4. Code for creation of class instances and loading to an object database.

Conclusion

We have discussed two qualitatively different levels of solutions to interoperability. The traditional EDF approach provides indirect solution to interoperability and convey good conditions for interoperability when heterogeneity of EDF is low. MC-based approach to interoperability is capable of addressing interoperability directly and is considered more suitable for interoperable systems dealing with modeling of various complex phenomena. This is relevant for development of DE technology. Based on nine factors relevant to interoperability a conceptual comparison has been elaborated, which shows that MC-based solutions are more suitable for interoperable systems dealing with complex models. This is deemed to be the case for most information systems that are used today by humans. It is also possible to point out Java technology as a concrete MC technology with the best overall evaluation of the factors of interoperability. Java is unique compare to other MC technologies in openness and extraordinary in

platform portability.

MC technology has great potential, even outside the scope of interoperability, but does NOT provide interoperable solution automatically. To exploit the potential of MC requires identification of the core subject for interoperability and a careful design of the solution. It has been recognized that properties of extensible EDF are still limiting to interoperability. The problem is that these EDF, such as XML, are extensible as an exchange to being abstract, which cannot be considered as a solution when it comes to interoperability between systems. Systems always inter-operate in terms of a specific application. We have experienced elimination of parsing in MC-based solutions as strong advantage in development practice. At the same time it is still possible to involve development of parsers for specific EDF when necessary. The main disadvantage of MC-based solution are limitations in selection of programing technology and computing platform, which may be critical for some systems.

This article shows a concrete example how data and functionality can be handled in an interoperable way using MC. Both the definition and the creation of a model-map content has been presented using GRIFINOR platform for geo-visualization. On the example it has been also showed how MC-based approach allows for a seamless integration of new or alternative representations without refactoring the interoperable system. This ability of gradual refinement is considered important for complex representations used in DE technology in general.

The case of GRIFINOR solution currently requires to store associated classes explicitly. For example the PgSQL object at line 31 in Figure 3 would have to be either stored in GRIFINOR object database manually or the library would have to be available for the viewer at runtime. There are other practical and theoretical concerns in GRIFINOR platform such as handling level-of-detail or controlling transmission of linked objects, which are important for development of DE technology. However they can be treated separately from the concepts of MC-based interoperability addressed in this article.

Figure 5. Geometric representation of three consecutive road elements in GRIFINOR.

References

Gore, A. (1998). The digital Earth: understanding our planet in the 21st century. , Keynote: Grand Opening Gala of the California Science Center, Los Angeles, CA, USA.

Kjems, E. and Kolar, J. (2005). From Mapping to Virtual Geography. *In Proceedings of CUPUM 2005*, electronic publication.

Kolar, J. (2006). On the Road to 3D Geographic Systems: Important Aspects of Global Model-Mapping Technology. *Innovations in 3D Geo Information Systems*, Springer-Verlag, pp. 207–223

Schroeder, W. J., Martin, K. M. and Lorensen, W. (2002). Basic Data Representation. *The visualization toolkit 3rd ed.*, pp. 111-148.

Vckovski, A. (1998). Interoperable and Distributed Processing, Taylor & Francis, 1 Gunpowder Square, London, EC4A 3DE, U.K.