Component-based Software Architecture Applied for Design of Heritage Content

Oleg Iliev, Radoslav Yoshinov

Laboratory of Telematics, Bulgarian Academy of Sciences
Bl. 8, Akad.G.Bonchev Str., 1113 Sofia, Bulgaria
ilievo@cc.bas.bg, yoshinov@cc.bas.bg

Abstract. The paper designs a model for creating heritage content, composed as a combination of multiple components, thanks to the opportunities provided by a component-based software architecture. The result is the ability to annotate different types of heritage objects available in public databases, such as video, sound, image, text, 3D model, and more, to obtain meaningful content. Moreover, the "component" types that contribute to a more complete presentation of the information delivered can be constantly increased without requiring a significant change in the software environment, but only a simple configuration change.

Keywords: Heritage Objects Modeling, Heritage Content, Component-Based Software Architecture.

1 Introduction

There are large number of databases with heritage objects (video, sound, image, text, 3D model, etc.), publically accessible over the Internet, described in great detail with metadata according to approved standards for digital library content stores. The United Nations Educational, Scientific and Cultural Organization (UNESCO) offers one of the largest databases of its kind. Databases provides access to maps, videos, photos, documents and other items related to heritage sites. In most cases, this data is available for free use, which allows the stored heritage objects to be aggregated, annotated and presented according to individual preferences.

Using the granulation of the data related to the heritage sites suggested by UNESCO's database, it is possible to determine the size of the components that grouped together could describe a complete heritage content. Determining the correct level of granularity ensures the creation of meaningful objects that can be used both autonomously and reused in different context or layout. Although it is not directly related to heritage content, Wagner's Learnativity Content Model (Wagner, 2002) was used to describe the size of individual objects and the characteristics of the ready to use content.

The component-based software architecture designed in the paper is aimed at creating a flexible web-based environment for representing heritage content. It has 5 main components: 1. Components that do not offer any visual presentation (user interface);

Digital Presentation and Preservation of Cultural and Scientific Heritage. Conference Proceedings. Vol. 11, Sofia, Bulgaria: Institute of Mathematics and Informatics – BAS, 2021. ISSN: 1314-4006, eISSN: 2535-0366

2. Components with visual interface; 3. Components aggregators of other individual components; 4. List of system definitions; 5. Orchestrator/composer of components. The components providing visual interface are directly related to the way of presenting the heritage content. The question of what should be considered as a user interface component is directly related to the specified granularity of the heritage objects and set the conditions for software representation of these objects.

According to the model designed in the paper, the annotation of the individual components is controlled by a tree structured definition and can be completely dynamic. In other words, the software environment can not only be supplemented with new components, but also their application can be dynamically controlled.

The paper also presents a prototype of a software environment for annotation of heritage content. The prototype uses a set of technologies, including a high-level object-oriented programming language – C#, in combination with the .NET Framework and a tool for pre-compilation of the visual part of the system.

2 Classification and Granulation of Heritage Objects

The United Nations Educational, Scientific and Cultural Organization (UNESCO) defines the so-called World Heritage Sites (landmark or area with legal protection by an international convention) for having cultural, historical, scientific or other form of significance. According to UNESCO, a heritage site could be described as "cultural and natural heritage around the world considered to be of outstanding value to humanity". Moreover, the organization classifies heritage objects into three groups: cultural sites, natural sites and a mix of natural and cultural sites. (UNESCO World Heritage Centre, 2021) Numerous databases storing objects related to the heritage sites are used for the sites' maintenance and the determination of their significance. For the purposes of this study, the UNESCO World Heritage List (UNESCO World Heritage Centre, 2021) and the Australian Heritage Database (Australian Heritage Database, 2021), both containing information about natural, historic and Indigenous places, have been examined.

The databases provide access to maps, videos, photos, documents and other elements related to a specific heritage site. Moreover, a partial annotation of these elements is proposed to present a heritage content. It is important to note, however, that the examined databases offer free access to the resources stored in them and anyone interested can use them to create new heritage content.

When it comes to determining the components, which grouped together form a heritage content, it is important to determine the right granularity to be used. In other words, what can we accept as a stand-alone component (that can be used independently and/or to create a new content). UNESCO suggest its vision of the granularity of resources related to the heritage site by providing images, sound and video files, text documents and maps. At the same time, others would say that the components can be granulated even more deeply.

The basic components can be considered as digital objects. In the broad sense, a digital object is an information object that has a digital form (at least one) and is described with metadata. Digital objects may be audio-visual objects, e.g. images, text,

web pages, sound, animation, etc., which are usually grouped into collections to certain criteria and are stored in special storage facilities together with their meta-descriptions. (Arapi, 2016) These, according to the Library of Congress (METS, 2021), are:

- Descriptive metadata: information relating to the intellectual contents of the object, akin to much of the content of a standard catalogue record: this enables the user to find the object and assess its relevance (Arapi, 2016)
- Administrative metadata: information necessary for the manager of the electronic collection to administer the object, including information on intellectual property rights and technical information on the object and the files that comprise it (Arapi, 2016)
- Structural metadata: information on how the individual components that make up the object relate to each other, including the order in which they should be presented to the user: for example, how the still image files that comprise a digitized version of a print volume should be ordered (Arapi, 2016)

Although it is not directly related to the description of a heritage content, the Learnativity Content Model (Wagner, 2002) illustrates the concept of assembling content into higher-level objects and could be used in the context of heritage information presentation. The objects examined by the model are assembled from information objects and assets into higher-order collections. This model is very useful for describing granularity and granularity is very useful to achieving reusability. Reuse not only saves time and money, but also enhances the quality of digital learning experiences, resulting in efficient, economic and effective learning. (Arapi, 2016)

The basic components of the Learnativity content model are the following (METS, 2021):

- **Content Asset:** Content Assets include raw media such as images, text snippets, audio clips, applets, etc.
- **Information Object:** A text passage, Web page(s), applet, etc. that focus on a single piece of information. It might explain a concept, illustrate a principle, or describe a process
- **Learning Object:** In the Learnativity content model a Learning Object is a collection of Information Objects that are assembled to teach a single learning objective
- **Learning Component:** A Learning Component is a generic term for things like lessons and courses are typically connected with a higher level learning objective and have multiple learning objectives since they are composed of multiple Learning Objects
- **Learning Environment:** The "Learning Environment" is a catch-all phase for the combination of content and technology with which a learner interacts. A combination of learning components with communication tools and/or other features that facilitate an e-learning experience can be aggregated into a learning environment (e.g. LMS)

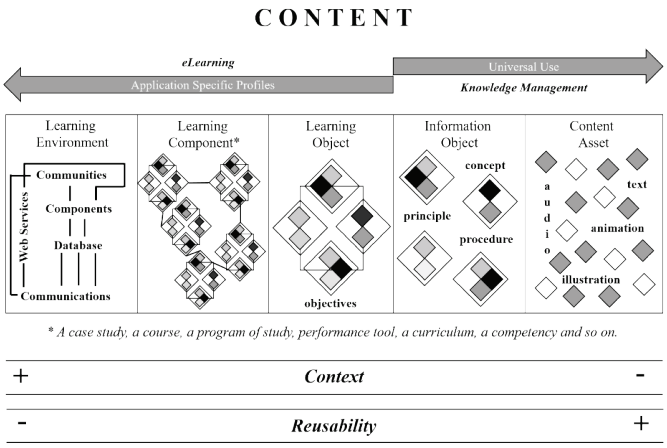

Fig. 1. The Learnativity content model (Wagner, 2002)

It is commonly accepted that there is an inverse relationship between the size of an object and its reusability. Fine-grained objects/components have the potential to be flexibly assembled into meaningful content, whereas the already coupled content is often not suitable for use in a different context. (Arapi, 2016) This fact is also illustrated in Figure 1.

For the purposes of this paper, a higher level of granularity for the constituent parts of the heritage content has been selected. The goal is not to achieve a learning environment, training course or even a lesson, but rather to have the annotation level close to one offered by UNESCO in the World Heritage List. The study considers a heritage component as an equivalent of a "Content Asset" (different types of media files, 3D objects, text, etc.) from the Wagner model, and the end result of aggregated components would be equal to an "Information Object" (orchestrating one or more components and present them into a valid and meaningful heritage content) from the same model.

3 Component-based Software Architecture

Breaking big things into smaller components and concentrating on them has always aroused the interest of engineers. In their designs, the engineers aim to provide easy maintenance of small replaceable parts that, when combined together, have high quality and performance. Software development is no exception to this principle of operation. A fundamental concept behind object-oriented programming is the creation of parts of programing code that group semantically related logic and can be maintained independently of other parts of the system. This is clearly modeled in the SOLID programming principles (Joshi, 2016).

Component-based software engineering considers the principles for ensuring the reusability of the individual components of a system, while ensuring the necessary separation of concerns. The component based-software architecture is based on this software development model. It aims to look beyond the well-known design patterns and

to provide loosely coupled components that are completely autonomous and independent of each other. (Iliev, 2020)

3.1 The Building Parts of Component-based Software Architecture

The main building part of the component-based software architecture are the components. They must be independent of each other, be able to be maintained and developed absolutely individually, and also be able to work autonomously. The components are combined logically according to a specific definition by a component orchestrator and thus ensures the completeness of the software system.

This paper suggests that the components needed to create an environment for managing and presenting heritage content. Given the extremely high popularity of the web services in the last 5+ years and their complete replacement of the well-known desktop applications in the past, the model designed in the paper assumes that it will support a web-based environment. According to the model, the system has 5 types of components (Figure 2):

1. **Components that do not offer any visual part**: Types of software libraries that aim to contain shared business logic between individual components
2. **Components with a visual interface**: Small building block parts of a web-based solution – modal windows, components for presenting a heritage object, headers, footers and more
3. **Components aggregators of other components:** With their help it is possible to annotate the components and create meaningful and valid content. They also could be considered as components with visual interface
4. **List of system definitions, referred in the paper as Manifest:** Defining the components of which the software environment is composed
5. **Orchestrator / composer of components, referred in the paper as Bootstrapper**: Aiming only to load a set of components according to the list of system definitions.

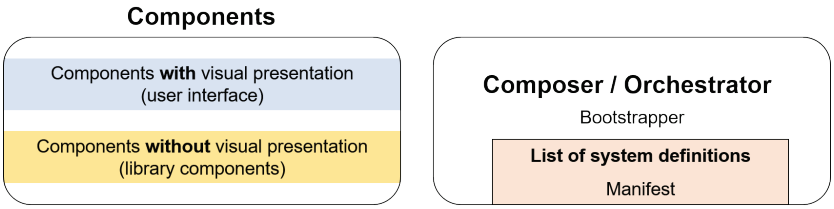

Fig. 2. Component-based software architecture – building block

The list of system definitions presents a list of all used by the system components. Adding a new component in the system and thus extending its functionalities is extremely simple and requires no more than updating the definition file. In this way, Bootstrapper automatically loads each subsequent component (Figure 3).

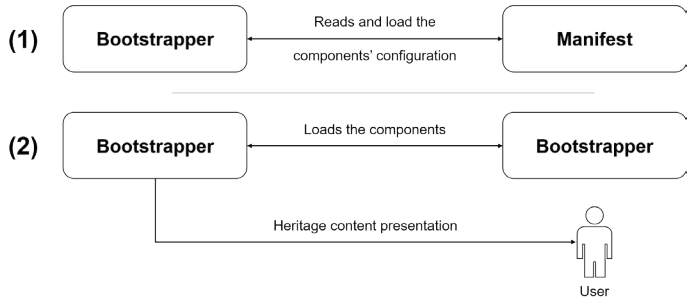

Fig. 3. Component management in component-based software architecture

4 Environment for Management and Presentation of Heritage Content

In terms of an environment for managing and presenting heritage content, the component-based software architecture can provide opportunities for easy annotation of heritage content. With its help, the individual components building a heritage content can be created and maintained independently and permanently, and each subsequent "component" contributes to an even more valuable presentation of the content.

This paper defines the term "component", also referred as heritage object, in the context of heritage content. Using this definition, we could define the term "software component" in an environment for managing and presenting heritage content. The principles of component-based architecture, defined in the previous section, must also be followed.

Suitable components for an environment for the presentation of heritage content are (Figure 5):

- **User Interface (UI) Components:** All types of components that can represent heritage objects. With their help, the data extracted from the databases for heritage content can be presented visually. For example, it is possible to create components for video presentation, image presentation, sound files, text, 3D models, Virtual Reality models and others.
- **Page Components:** They aim to aggregate several UI Components and then to present the result in the meaningful and well formatted content
- **Fundamental Components**: Components not having visual interface, but some business logic of fundamental use that could be shared between multiple components (equivalent of software libraries)

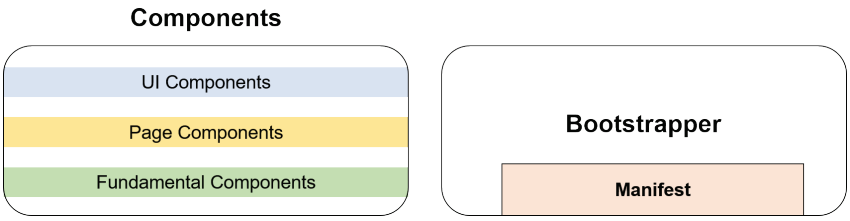

Fig. 4. Component design of an environment for the presentation of heritage content

The flexibility provided by the UI Components and subsequently their aggregation by the Page Components allows the easy extension and maintenance of the system. For example, a new UI Component could be created to support a live video stream from a heritage site that was not previously supported by the system. It then only needs to be added and configured in the Manifest.

The definition of the contextual connection between the individual components (the use of one component in another) can be represented by a relational database describing a tree structured relationship between "root" (Page Components) and "leaves" (UI components) (Figure 5). For example, a Page Component may support video presentation (Video Component) and Virtual Reality (VR Component).

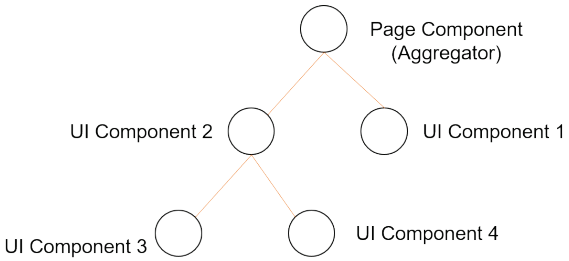

Fig. 5. Tree structured relationship between "root" (Page Components) and "leaves" (UI components)

Each component is versioned and the version is then represented in its description and is part of the Manifest, which in turn also has its own description – Manifest ID (Figure 6).

```
{
  "ManifestId": "Manifest-1",
  "Components": [
    {
      "Name": "HomePageComponent",
      "Version": "1.0.0"
    },
    {
      "Name": "LibraryComponent",
      "Version": "1.0.0"
    },
    {
      "Name": "ImageComponent",
      "Version": "1.0.0"
    },
    {
      "Name": "VideoComponent",
      "Version": "1.0.0"
    }
  ]
}
```


Fig. 6. Model of a Manifest

Each new version of a component is being deployed to the web server in off-state, but it is being "activated" with a change of the Manifest, where its version should be referenced (Figure 7).

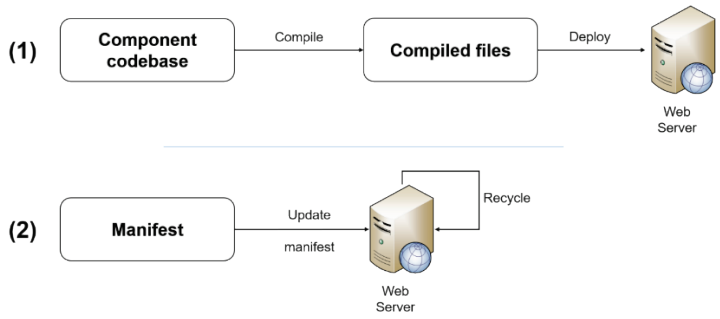

Fig. 7. Development life cycle of a component and its way for activation

The advantages of this model for software architecture are many, but some of the most important are: (1) ability to independently support each part of the system; (2) flexibility to constantly extend the system; (3) the accelerated software development life cycle. Unlike the software applications that use a monolithic architecture, in the component-based software architecture, changing a small part of the system does not require update and release of the entire system. For example, changing the image visualizations and styling is isolated to the Image Component only – the other components do not change at all. This accelerate the software development life cycle and increases its productivity. The delivery of the changed components is instantaneous, and their "activation" is possible by only changing the Manifest. In other words, activating but also deactivating delivered changes in components or introducing new components is a matter of seconds. Moreover, the time for testing and quality assurance of the changes made is also shortened, as the part of the system to be tested can be closely isolated to a single component without the need to test the whole system. Moreover, the components with a visual interface can be tested absolutely independently of the other components.

5 Aggregation of Heritage Object

The aggregation of the individual UI components must be easy and convenient in order to achieve a powerful Content Management System (CMS) (Boiko, 2004). The model designed in the paper considers the CMS from three perspectives (Figure 8):

- **Template management of the individual Page Components:** The logic (potential relationship with UI Components) and layout (markup and styles) contained in the Page Component itself. An exact location of an individual UI Component or set of UI Components (children – without specified a single one) could be configured and styled.
- **Component relationship management:** Describes the relationship between the individual components (UI/Fundamental/Page Components), for example the relationship between the Page Component – Home page, with the UI Component to present images – Image Component
- **Component data management:** Standard functionality for any CMS. It is controlled by the database, where the metadata of the individual heritage object is stored

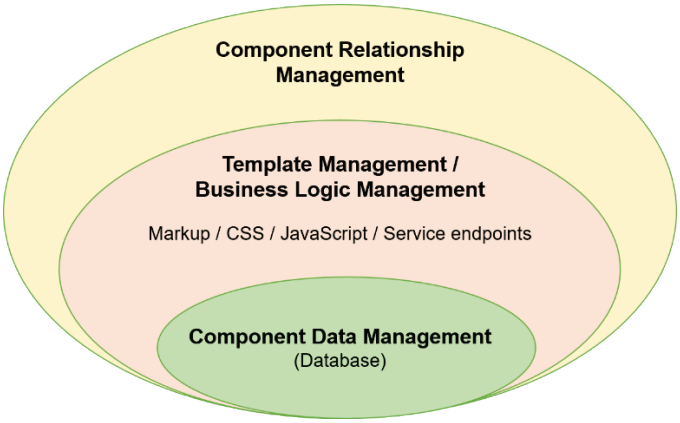

Fig. 8. CMS perspectives considered

6 Prototype of an Environment for Management and Presentation of Heritage Content, Built with Individual Heritage Objects

For the design and development of a prototype of an environment for management and presentation of heritage content, built with heritage objects, a language for high-level object-oriented programming language – C#, in combination with the .NET Framework and a tool for pre-compilation of the visual part of the system – was selected. The reason for choosing such a technology is the possibility to pre-compile the finished product, but not to leave it be interpreted by the web server. This makes the components com-

pletely wrapped and independent of each other. In this case, the components are compiled to files with the extension ".dll". Another reason for choosing these technologies is that they provide an easy way to implement the Model-View-Controller (MVC) architecture, which then allows an easy control of the HTTP request processing cycle (Freeman, 2014).

Typically, the presentation part of .NET Framework MVC systems is a view, which is represented by a separate file and it is not being part of the compiled package (".dll") (Freeman, 2014). In order to achieve absolute autonomy of the components in the prototype of the model designed in the paper, a tool called RazorGenerator (Razor Generator, 2021) was used. Thanks to the use of the tool, each part of the system is then precompiled and delivered entirety as a single ".dll" file.

For the purposes of the prototype, 2 types of UI Components were design and developed – ImageComponent (presenting image heritage objects) and VideoComponent (presenting video heritage objects), as well as one Page Component – HomePageComponent (presenting a prototype of a home page and aggregating the two UI components to achieve meaningful heritage content) (Figure 9). Additionally, a component (Fundamental Component) that has no visual representation, but contains shared business logic used by the individual components (Page and UI Components), has been designed and developed – LibraryComponent.

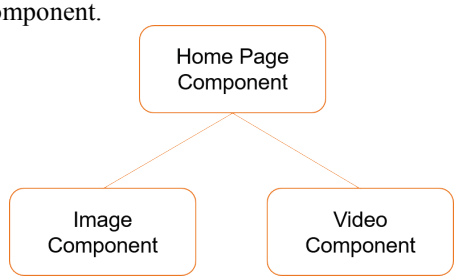

Fig. 9. Prototype's component relationship

A simple file defining the components of which the prototype is composed – Manifest, was created (Figure 6).

The component relationship between the individual components with visual interface used by the prototype's system is described into a simple JSON file called ComponentManagement (Figure 10)

```
{
   "ChildrenComponents":[
      {
         "Name":"ImageComponent"
      },
      {
         "Name":"VideoComponent"
      }
   ]
}
```

Fig. 10. Component management file, describing the relationship between the individual components with visual interface

The orchestration and aggregation of the components is done by Boostrapper, which reads the Manifest file when booting the system and loads all the listed in the file components with their required version in one solution and thus the assembles that are seemingly separated and independent from each other creates a monolith like system (Figure 11).

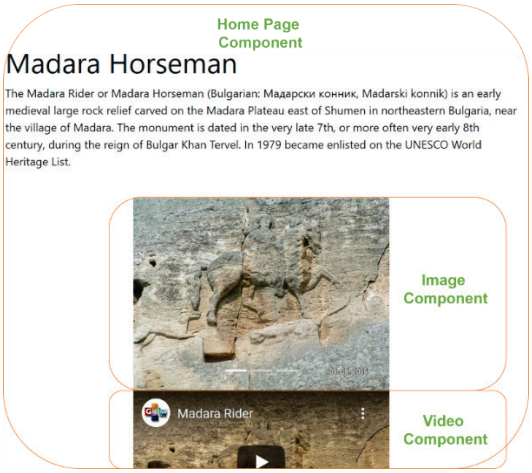

Fig. 11. An aggregated heritage content, build using multiple individual heritage objects

7 Conclusion and Future Work

The large number of databases with heritage objects (video, sound, image, text, 3D model, etc.), publically accessible over the Internet allows the creation of a new heritage content, aggregated according individual preferences. At the same time, having a lot of data does not necessarily mean that it is easy to group and eventually product valuable content. It is important an appropriate granularity to be determined, so the individual object may be used in more than one particular context. The paper suggests a level of granularity of the heritage objects, which is then used to determine the characteristics of the software components, forming a component-based architecture.

The model design in the paper provides a way to dynamically create new components, which are responsible for the implementation of more and more types of data available in databases with heritage content into a software environment. This increases the value of the heritage content created by the environment.

One of the great advantages of the component-oriented software architecture is the optimization of the development life cycle. This architecture shortens the time between completing work on a software implementation and its further release to web servers. Moreover, the testing process is significantly optimized. Unlike the software applications that use a monolithic architecture, in the component-based software architecture, the testing of a software change could be isolated to a small part of the system (single component) and do not require full regression testing of the whole system.

The testing of the individual components could be further optimized and extra supported by the model designed in the paper. In the current version of the model, the testing of a component goes through the "activation" of the updated component as a part of the whole system. However, this could be isolated to a single component testing. In order to archive that the model need to be extended to support such kind of individual component testing.

References

Arapi, P. (2016). Supporting Personalized Learning Experiences on top of Multimedia Digital Libraries. *International Journal of Education and Information Technologies*, NAUN, vol. 10, 152-158.

Australian Heritage Database. (2021). *Australian Heritage Database*. Retrieved from https://www.environment.gov.au/cgi-bin/ahdb/search.pl

Boiko, B. (2004). *Content Management Bible*. Wiley Publishing.

Freeman, A. (2014). *Pro ASP.NET MVC 5*. Apress.

Iliev, O. (2020). "Hot releases" in learning management systems and learning content management systems. *Forty-ninth Spring Conference of the Union of Bulgarian Mathematicians*, (pp. 137-143). Borovets.

Joshi, B. (2016). *Beginning SOLID Principles and Design Patterns for ASP.NET Developers*. Apress.

METS. (2021). *METS. Metadata Encoding and Transmission Standard.* Retrieved from Metadata Encoding and Transmission Standard (METS): http://www.loc.gov/standards/mets/

Razor Generator. (2021). *Razor Generator*. Retrieved from https://github.com/RazorGenerator/RazorGenerator

UNESCO World Heritage Centre. (2021). *World Heritage*. Retrieved from World Heritage List: https://whc.unesco.org/en/list/

Wagner, E. (2002). Steps to creating a content strategy for your organization. *The e-Learning developers' journal*, 103-120.

Received: June 20, 2021
Reviewed: July 18, 2021
Finally Accepted: July 28, 2021