ORIGINAL SCIENTIFIC PAPER

ACTA GRAPHICA 236

Dynamic Frames Based Generation of 3D Scenes and Applications

Author

A. Kvesić, D. Radošević*, T. Orehovački

University of Zagreb
Pavlinska 2, HR 42000 Varaždin
Faculty of organization and informatics
Hrvatska
E-mail: danijel.radosevic@foi.hr

Abstract:

Modern graphic/programming tools like Unity enables the possibility of creating 3D scenes as well as making 3D scene based program applications, including full physical model, motion, sounds, lightning effects etc. This paper deals with the usage of dynamic frames based generator in the automatic generation of 3D scene and related source code. The suggested model enables the possibility to specify features of the 3D scene in a form of textual specification, as well as exporting such features from a 3D tool. This approach enables higher level of code generation flexibility and the reusability of the main code and scene artifacts in a form of textual templates. An example of the generated application is presented and discussed.

Keywords:

3D scene, dynamic frames generator, Unity

1 Introduction

Dynamic frames based code generator (Radošević and Magdalenić, 2011.) was introduced with the aim of making source code generators capable to produce full applications, not just skeletons. Such generators have to be independent from the programming language of a target application, as well as from the used programming technologies and components. The usage of this code generator has been so far mostly used in automatic generation of web applications because of a variety of code that should be produced from the same specification, like Html code with pieces in JavaScript, different kinds of server side scripts (e.g. PHP and Python scripts), CSS etc.

The building of 3D scene based program application include usage of a 3D modeling toll together with a program interface. Tools like Unity (Unity3D, 2014) significantly facilitate this process. Among others, these tools enable 3D modeling with physical model included as well as programming in standard programming languages, like C#.

Unity has also an additional possibility to enable programming of a program editor as well as the application itself. But, on the other hand, 3D modeling is still faced with significant problems that are outcome of the 3D scene complexity.

Usage of code generators enables different, textual form of 3D scene representation that can be, as demonstrated in SCT based generator (Azri et al., 2012), expressed in the form of program specification. In the process of 3D scene editing, separation of textual specification parts is much easier than extraction of the appropriate model parts using 3D editor which represents model elements graphically. Instead of working with the whole 3D scene, some parts could be generated from separated specification part and later edited in a visual 3D editor that can export updated specification. This means that 3D scene can be edited as a textual representation as well as by using of the 3D editor. Apart from the set forth, it is possible to extract just a part of textual representation, edit appropriate part of 3D scene in an 3D scene editor, and update the textual specification.

Considering the general case of code generation, there is an issue of integrating generators into the software development systems because generated code should not be modified outside the generator (because these modifications could be lost after the next code generation). This paper demonstrates one possible way of overcoming this problem in the domain of 3D modelling.

The remainder of the paper is structured as follows. Brief literature review is provided in section 2. Foundations of SCT generator model are offered in section 3. Section 4 describes the process of 3D scene generation. The example is presented in section 5. Concluding remarks are given in the last section

2 Background to the research

There many different approaches in Automatic Programming that share similar goals and approaches as our research. Generative Programming (GP) was introduced in the late 1990's and deals with designing and implementing of software modules [Czarnecki]. Modules are combined to generate specialized and highly optimized systems fulfilling specific requirements [Eisenecker]. In its base, GP uses the advanced concepts of Object-Oriented Programming and Generic Programming, together with Metaprogramming, Domain Engineering and Aspect Oriented Programming (AOP). AOP is focused on the crosscutting concerns in complex software [Kiczales]. GP offers more flexibility in development of generators and applications, by using more generators in code production and higher level of reusability of program artefacts.

Our approach in building of generators is named as the SCT generator model [Radošević and Magdalenić 2011]. It is based on the dynamic frames (as described in chapter 3) unlike some other frames based models, like XVCL [Jarzabek]. Application Specification defines the features of the final application are defined, similar to Feature Oriented Programming (FOP)[Prehofer]. Code templates are the main building blocks, containing connections that can be used for adding of different crosscutting concerns. Finally, SCT uses the Configuration to make problem-domain adjustments, which is done by pre-processor definitions in the approach given by Rosenmüller [Rosenmüller].

There some recent attempts of using code generators and other automatic programming based tools in generation of 3D models and belonging applications.

Sugihara and Kikata (Sugihara and Kikata., 2013) have introduced an integrated system that automatically generates 3D urban models like buildings, residential areas, and city plans from building polygons such as ground plans or top views. By employing the proposed polygon expression together with the partitioning scheme, their system is able to generate two hundred 3D building models in less than thirty minutes.

Witte et al. (Witte, 2008) proposed a concept of generating accurate 3D military models. The generation of 3D object models in their system is based on the interpretation and combination of near-term laser range data and infrared images collected by reconnaissance carried out in advance.

Lim et al. (Lim et al., 2014) generate 3D models from a set of the training shapes in form of binary images and this method works independently of the object shape, geometry, or topology. Lee et al. (Lee et al., 2014) have created a method for 3D modelling of particular vessels, which is used in diagnosing coronary artery diseases. Azri et al. (Azri et al., 2012) have identified four different approaches for automatic generation of 3D indoor models. Semantics dependent generation approach is based on the analysis of the text, interview records, and video files that contain semantic information about a building. Information fusion approach integrates semantic information (e.g. names, attributes, or states of building elements) gathered from diverse sources (e.g. CAD files, digitalized blueprints, ID tags, etc.) with geometric information (e.g. height of a building, dimensions of floors, etc.). The third approach enables transformation of building representations in different models. In the fourth approach, the automatic generation of indoor models is enabled with the use of novel techniques for tracking people's motions such as gesture recognition sensors, computer vision, accelerometers in the mobile phones and mobile augmented reality.

Dachselt and Rukzio (Dachselt and Rukzio, 2003) have proposed an Extensible 3D model based declarative framework called Behavior3D meant for modeling behaviors of 3D objects.

Klöckner et al. (Klöckner et al., 2012) have introduced a scripting-based technique meant for Graphics Processing Units (GPU) run-time code generation that address several issues related to programming the GPUs, including the automated selection of the best code variant in terms of the predefined metric such as execution speed as well as the high-performance abstractions and cost-benefit flexibility in generating the needed number of code variants.

It can be seen that generation of 3D models increasingly attracts academic attention. But the researches addressing the generation of 3D scene and belonging applications are still rather scarce and this has motivated us to initiate a research into the design of a generator that would enable automatic generation of 3D scene objects from textual specification. Features of the generator and its model are described in the following section.

3 SCT generator model

SCT generator model is based on previously introduced Scripting generator model (Radošević et al., 2005). This model has used textual specification in a form of attribute-value pairs and the set of program templates as the building artefacts of the target program. Also, the scripting model introduces the graphical diagrams to specify the structure of specification (Specification diagram) and the connections of the program templates (Configuration diagram) that are kept in the SCT. The abbreviation SCT (Radošević and Magdalenić, 2011) comes from Specification, Configuration and Templates, as the building elements of the generators. The main novelty of the SCT is the textual form of configuration; i.e. they are no more need to program the configuration. That separation of configuration from generator code enables much more flexibility of the generators, even in a way that generator can produce and execute programming code on demand, as described in (Magdalenić et. al. 2013).

SCT is based on dynamic frames (Radošević and Magdalenić, 2011), named as SCT frames (presented in Figure 1), unlike some other frames based generator models, such as XVCL (Jarzabek et al., 2003) and Basset's frames (Bassett, 1997).

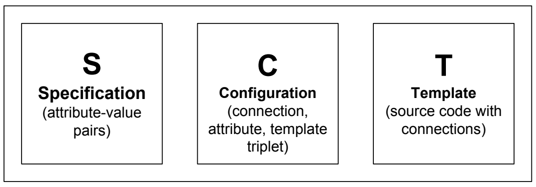

S
Specification
(attribute-value pairs)
C
Configuration
(connection, attribute, template triplet)
T
Template
(source code with connections)

Figure 1. SCT frame (Radošević et al., 2011)

Specification contains features of generated application in form of attribute-value pairs. Each Template contains source code in target programming language together with connections (replacing marks for insertion of variable code parts). Configuration defines the connection rules between Specification and template.

Starting SCT frame contains the whole Specification, the whole Configuration, but only the base template from the set of all Templates. Other SCT frames are produced dynamically, for each connection in template, forming generation tree (Figure 2):

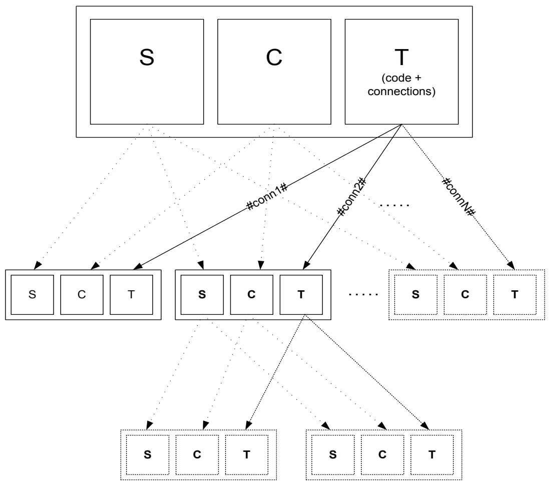


Figure 2: The generation tree (Radošević et al., 2011)

The aforementioned indicates that user has to define only top-level frames, while the others are dynamically allocated during the generation process.

SCT is suitable for producing applications that consist of different types of code (e.g. web applications that are comprised of snippets of code written in diverse languages such as HTML, XML, JavaScript, etc.). All parts of such heterogeneous applications can be produced from the same Specification thus achieving the high level of reusability. Up to now, the SCT model has been employed for the development of Autogenerator (Radošević et al., 2012), generation of student assignments (Radošević et al., 2010), determining error messages that occur at the level of generator (Radošević, Magdalenić and Orehovački, 2011), and design of a framework for building generators (Radošević et al., 2013).

3.1 Specification diagram

Graphically, the hierarchy among Specification attributes can also be represented by a Specification Diagram (Radošević et al., 2005), as shown in Figure 3. The Specification diagram is a hierarchic diagram that defines the proposed application properties in the form of a tree-like feature model.

Attributes defined as containers are marked by '[]', while groups have suffix '_'. In a textual form of Specification, the hierarchy is specified by '+' sign.

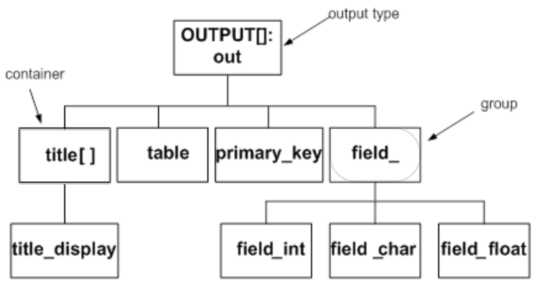


Figure 3: Example of Specification Diagram

3.2 Configuration diagram

Configuration is specified by three-element groups (Radošević and Magdalenić, 2011), containing:

- connection (base element),

- source (attribute from Specification) and

- template (lower level template, if present)

Connections occur in Templates, and should be replaced by the program code during the process of generation. The source refers to the value of a particular attribute or all values from a group to be used in code generation. The template can be omitted, which means that the connection should be replaced by the appropriate source. If the template is specified, then it should be used for each appearance of the specified source. These three elements are also used in specifying Configuration graphically (Figure 4) by a Configuration diagram (Radošević and Magdalenić, 2011).

Basic elements are mutually connected in a Confuiguration diagram, as shown in Figure 5:

The more complex generator is made by superposition of several single-level generators.

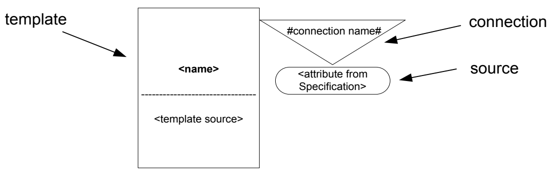


Figure 4: Elements of Configuration diagram

4 Generation of 3D scene

In order to successfully accomplish 3D scene generation and to build a functional 3D environment, it is important to use the software which is capable of reading specified input and producing the appropriate results. However, regardless of which software is used, few steps should be made and experiments carried out in order to achieve the complete symbiosis between 3D modeling and generation of the 3D scene. First of all, software functionalities should be exhaustively explored in order to determine which of them are most appropriate for the generation process. These do not specifically include advanced 3D modeling capabilities or carrying out modern graphics, special effects etc. but support for certain programming language and corresponding editor which represent working environment. Another important step is the employment of properties for the purpose of defining the models in a 3D space.

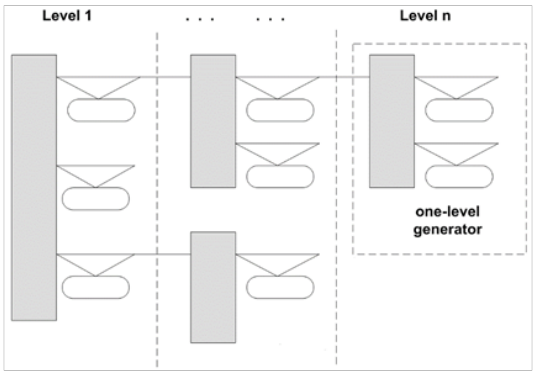


Figure 5: Example of the Configuration diagram

After software abilities are identified and 3D models are defined, connection between generator and 3D software should be established. This includes creation of common language – interpreter that would be able to represent 3D models and support communication. More specifically, 3D software should be able to produce an understandable form (e.g. textual) of 3D models and present their properties to the generator. Likewise, generator could then communicate with 3D software and, for example, through the common language, change some of the 3D object's properties, create a new object, remove some of the existing objects, etc.

Last part refers to the selection of the appropriate 3D software. Considering the objective of our research and features of available software for 3D modeling, for the purpose of generation and building of 3D scenes we have chosen game development software Unity 3D (Unity3D, 2015). The main advantages of Unity include programmatic allocation of artifacts which are part of the 3D scene and support for building the 3D scene in an executable file. Apart from the set forth, Unity's editor can also be programmatically managed. This feature is especially interesting in the development of generators because it enables modifications of generated 3D scene in the editor. As depicted in Figure 6, the process of generating 3D scene consists of following operations:

- Building and updating the SCT generator (Radošević et al., 2013) in order to produce the necessary application and editor code for dynamic allocation of 3D objects and scene. Both application code and editor code are being generated.

- Loading the generated 3D scene into the 3D editor. Editing the 3D scene includes adding/deleting 3D artifacts as well as changing their features (position, rotation, scale, texture, etc.). The scene can be exported in a form of Specification for the purpose of the generator.

- Building an executable application. After Specification is completed, the generator produces the program code. The application can be compiled by means of the standalone compiler and without using the interface of the 3D editor.

There are several important features meant to be used for the development of SCT generator and aimed for building of 3D scene based applications:

- Generator is meant for producing 3D scene from previously prepared 3D artefacts.

- The behavior of 3D objects and scene, as well as other application features, is defined in

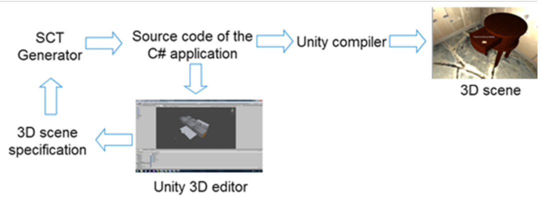


Figure 6. 3D scene editing and generation process

program Templates which can be applied to different scenes.

- The Specification consists of attributes that specify main objects and with them associated child objects. It is possible to extract only selected main/child objects for the purpose of editing in 3D editor or producing the executable application. This feature can be used for decreasing the complexity of 3D model while editing.

5 Uniti 3D

Software chosen for creation and building the 3D scenes – Unity, is most commonly described as a cross-platform game development software or game engine. It has built-in support for three programming languages – C#, JavaScript and Boo. Regarding 3D scenes and application view, Unity offers Scene view and Game view, which can be described as editor (Scene view) and application (Game view). The working environment of the Unity 3D editor is presented in Figure 7.

Considering the Unity's main purpose as software, i.e. its capabilities as the game engine, it has given us many features and different ways of using and developing the connection between the generator and the Unity, as well as the generator itself.

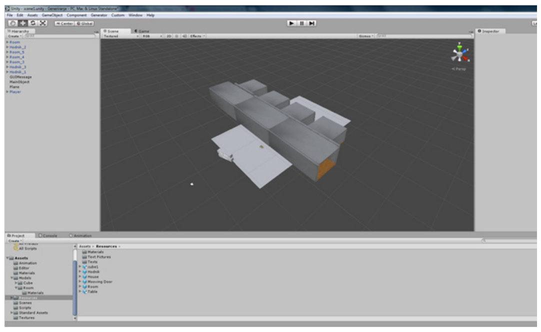


Figure 7. Unity 3D editor's interface

One of the most interesting features regarding Unity and its use with our generator is a programmable built-in editor. Programming code execution in Unity is possible not only during application runtime, but also during scenes editing inside Unity's projects. This allows us to generate scenes during the application runtime, i.e. its start and also during the scene editing, which can be very useful because we do not have to start our application every time we want to make changes to our scene and update it from Specification. Likewise, it is possible to update Specification in real-time and save every changes we made during scene's editing. This feature can be described as the "save" functionality, because it allows us to store complete status of 3D scene, including its objects, interactions, states etc. and load that state in any later moment. This can also be useful in situations which require quick changes, e.g. moving of one object along the axis, rotation around the axis or something similar. Lastly, there is another Unity's feature that allows adding custom menu items which are mapped to specific static methods. As illustrated in Figure 8, these methods can then be executed with a single click on the custom defined menu item.

6 An Example

An example of 3D scene generation that will be presented in this paper includes previously mentioned approaches related to Unity and generator. The whole creation process of the 3D scene includes variety of different steps and possible approaches to the solution, i.e. the final result.

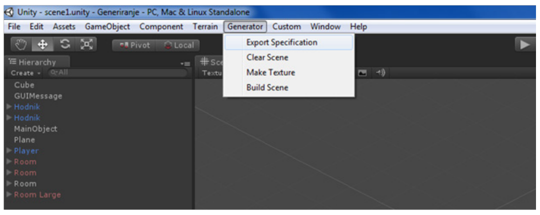

Unity - scene1.unity - Generiranje - PC, Mac & Linux Standalone
File Edit Assets GameObject Component Terrain Generator Custom Window Help
Export Specification
Clear Scene
Make Texture
Build Scene
Hierarchy
Cube
GUIMessage
Hodnik
Hodnik
MainObject
Plane
Player
Room
Room
Room
Room Large

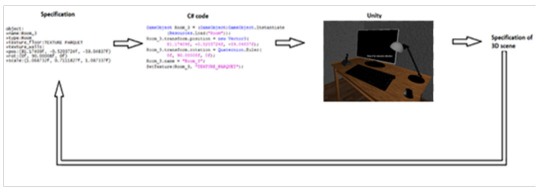
Figure 8. Custom menu items in Unity

First of all, basic 3D scene consists of various 3D models which are represented in the 3D space through their properties, like position, rotation, size, etc. Our example shows the 3D scene in which 3D objects are generated according to predefined properties, which are passed to Unity through Specification. On the other hand, Unity can also export chosen 3D object's properties, i.e. export Specification, which creates connection and closes the circle between generator and Unity.

Every 3D object in the scene is represented by corresponding part in Specification, but it is worth mentioning that most of the objects share basic properties (position, rotation, size, texture, etc.), which are considered as most important and fundamental for the purpose of representation and manipulation of 3D objects. It is important to mention that objects do not share values of properties, but the properties itself.

Figure 6 illustrates the interaction between generator and Unity, where Specification plays most important part. 3D objects that we want to include in the 3D scene are defined in Specification and then forwarded to Unity using code generation. These 3D objects are previously embedded into Unity in order to be used in a scene Specification.

Unity then applies generated code (in our case C#) which includes methods like the one presented in the Figure 9. In this specific example, the method is called Instantiate and it is used for the creation of new objects in a 3D scene. The result of the process presented in the Figure 6 are 3D objects populated through the scene which are determined by their position, rotation, size, texture and any other property included in the Specification. This is a simple example and demonstration of possibilities and ways of using Unity for generation and creation of 3D scenes, which vary with respect to different approaches, desired result, expected outcomes etc.



```
object:
+name:Room_3
+type:Room
+texture_floor:TEXTURE_PARQUET
+texture_walls:
+pos:(81.17409f, -0.5203724f, -59.04837f)
+rot:(0f, 90.00000f, 0f)
+scale:(1.068732f, 0.7111827f, 1.067337f)
```

```
GameObject Room_3 = (GameObject)GameObject.Instantiate
        (Resources.Load("Room"));
Room_3.transform.position = new Vector3(
        81.17409f, -0.5203724f, -59.04837f);
Room_3.transform.rotation = Quaternion.Euler(
        0f, 90.00000f, 0f);
Room_3.name = "Room_3";
SetTexture(Room_3, "TEXTURE_PARQUET");
```

Figure 9. Communication between generator and Unity

Interaction between generator and Unity goes both ways, i.e. it is possible to generate Specification from given 3D scene which can then be used for generation of similar or even drastically changed scenes. This can also be achieved using Unity and tags that are one of its features. Tags can be used to represent and identify one or multiple game objects inside Unity. They are more often used for identifying multiple objects so they can easily be accessed and manipulated. Using tags in the specific example gives many possibilities. For example, it is possible to assign the same tag to the multiple objects that are included in Specification and simply ignore other objects and parts of the scene. Unity can then export Specification which consists only of chosen objects and can be used to recreate, change and manipulate given 3D scene in different ways and approaches. One example of using this Unity functionality is modifying only the part of 3D scene, i.e. part of the Specification by labeling this part with tags, which makes maintaining and updating of 3D scene much easier.

Another important and interesting feature is real-time generation of 3D objects. For example, one floor of a building or a room inside a hallway can be generated and populated in a given time and thus provide dynamic and changeable environment. Likewise, it is possible to remove one part of the 3D scene given by Specification and consider saving the resources or temporarily reducing the load. Generation of the 3D scene does not necessarily have to imply building and creation of whole environment, but also manipulating and managing its parts.

Example described in this part of the paper also includes movement, interactions with the environment, graphical user interface and messages presented to the user, but these functionalities are not covered by Specification itself. They can be, however, manipulated and conditioned through Specification and that is something worth investigating and studying in the future.

7 Conclusions

This paper introduces a new approach in building of 3D scenes based on SCT generator. This approach enables specification of the 3D scene in a textual form of specification. But this approach also demonstrates how to overcome limitations of integrating source code generators in the development of software systems. The 3D scene in proposed approach can be modified by means of both 3D editor and textual specification for the generator. Furthermore, the employment of the SCT generator reduces the complexity of the 3D scene during its modification in the 3D editor.

The given example of generating the 3D scene have clearly illustrated an interplay between the SCT based generator and Unity 3D editor. The textual Specification consisting from attributes and their values enables building and managing of the 3D scene. The set forth includes representation of Unity's editor as well as executable application that deals with the 3D scene. All other features of the application, like movements and interactions with the environment are included in code Templates, and they are common for different generated 3d scenes.

In our future work, we plan to define the common building elements for different types of 3D scenes and, respectively, to enable modeling of different kinds of inner and exterior space.

8 References

Azri, S.; Isikdag, U.; Rahman, A. A. Automatic Generation of 3D Indoor Models: Current State of the Art and New Approaches. In International Workshop on Geoinformation Advances, Johor, Malaysia, 2012, http://www.academia.edu/2604376/Automatic_Generation_of_3D_Indoor_Models_Current_State_of_the_Art_and_New_Approaches, downloaded April 4th 2014.

Bassett, P.G.: Framing software reuse - lessons from real world. Prentice Hall, Upper Saddle River, NJ, USA, 1997.

Czarnecki K,. Eisenecker, U.W. Generative Programming: Methods, Techniques, and Applications. Addison-Wesley, 2000.

Dachselt, R., Rukzio, E.: Behavior3D: an XML-based framework for 3D graphics behavior. In Proceedings of the 8th international conference on 3D Web technology, pages 101-112, St. Malo, France, 2003.

Eisenecker U. Generative Programming: Beyond Generic Programming, Proc. Dagstuhl Seminar on Generic Programming, April 27--May 1, 1998, Schloß Dagstuhl, Wadern, Germany, 1998.

Jarzabek, S., Bassett, P., Zhang, H., Zhang, W.: XVCL: XML-based variant configuration language. In Proceedings of the 25th International Conference on Software Engineering, pages 810-811, Los Alamitos, CA, USA, 2003.

Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C. V., Loingtier J.-M., Irwin J. Aspect-Oriented Programming. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), volume 1241 of Lecture Notes in Computer Science, pp. 220-242. Springer Verlag, 1997.

Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., Fasih, A.: PyCUDA and PyOpenCL: A scripting-based approach to GPU runtime code generation. Parallel Computing, 38(3):157–174, 2012.

Kvesić, A., Radošević, D., Orehovački, T.:"Using SCT Generator and Unity in Automatic Generation of 3D Scenes and Applications", Proceedings of the 25th Central European Conference on Information and Intelligent Systems (Ceciis 2014), ISSN 1847-2001, Faculty of Organization and Informatics, Varaždin, 17.-19.09.2014. ,pp. 312-317, 2014.

Lee, N-Y.; Lee, J-J.; Kim, G-Y.; Choi, H-I. Automatic 3D Model Generation based on a Matching of Adaptive Control Points. In You, K. (Ed.) Adaptive Control. InTech, Shanghai, China, 2009, http://www.intechopen.com/books/adaptive_control/automatic_3d_model_generation_based_on_a_matching_of_adaptive_control_points__, downloaded April 4th 2014.

Lim, S-J.; Udupa, J. K.; Souza, A.; Jeong, Y-Y.; Ho, Y-S.; Torigian, D. A. A New, General Method of 3D Model Generation for Active Shape Image Segmentation. In SPIE Proceedings 6144, Medical Imaging 2006: Image Processing, http://dx.doi.org/10.1117/12.653751, downloaded April 4th 2014.

Magdalenić, I., Radošević, D., Orehovački, T.: Autogenerator: Generation and Execution of Programming Code on Demand. Expert Systems with Applications, 40(8): 2845-2857, 2013.

Magdalenić, I., Radošević, D., Skočir, Z.: Dynamic Generation of Web Services for Data Retrieval Using Ontology. Informatika, 20(3): 397-416, 2009.

Prehofer C. Feature-Oriented Programming: A Fresh Look at Objects. In Proceedings of the European Conference on Object- Oriented Programming (ECOOP), volume 1241 of Lecture Notes in Computer Science, pp. 419-443. Springer Verlag, 1997.

Radošević, D., Kliček B.: "The Scripting Model of Application Generators", Proceedings of The 16th INTERNATIONAL DAAAM SYMPOSIUM "Intelligent Manufacturing & Automation: Focus on Young Researchers and Scientists", ISSN 1726-9679, Opatija, 19.-22.10.2005.

Radošević, D., Magdalenić, I., Orehovački, T.: Building process of SCT generators. In Proceedings of the 36th International Convention on Information and Communication Technology, Electronics and Microelectronics, pages 1037-1042, Opatija, Croatia, 2013.

Radošević, D., Magdalenić, I., Orehovački, T.: Error Messaging in Generative Programming. In Proceedings of the 22nd Central European Conference on Information and Intelligent Systems, pages 181-186, Varaždin, Croatia, 2011.

Radošević, D., Magdalenić, I.: Python Implementation of Source Code Generator Based on Dynamic Frames. In Proceedings of the 34th International Convention on Information and Communication Technology, Electronics and Microelectronics, pages 369-374, Opatija, Croatia, 2011.

Radošević, D., Magdalenić, I.: Source Code Generator Based on Dynamic Frames. Journal of Information and Organizational Sciences, 35(2): 73–91, 2011.

Radošević, D., Orehovački, T., Magdalenić, I.: Towards Software Autogeneration. In Proceedings of the 35th International Convention on Information and Communication Technology, Electronics and Microelectronics, pages 1076-1081, Opatija, Croatia, 2012.

Radošević, D., Orehovački, T., Stapić, Z.: Automatic On-line Generation of Student's Exercises in Teaching Programming. In Proceedings of the 21st Central European Conference on Information and Intelligent Systems, pages 87-93, Varaždin, Croatia, 2010.

Rosenmüller M., Siegmund N., Saake G., Apel S. Code generation to support static and dynamic composition of software product lines. GPCE '08: Proceedings of the 7th international conference on Generative programming and component engineering, October 2008.

Sugihara, K.; Kikata, J. Automatic Generation of 3D Building Models from Complicated Building Polygons. Journal of Computing in Civil Engineering, 27(5):476-488, 2013.

Unity3D, Unity - Game Engine, http://unity3d.com/, downloaded: May 5th 2014. (Witte, 2008) Witte, C.; Armbruster, W.; Jäger, K. Automatic generation of 3D models from real multisensor data. In Proceedings of the 11th International Conference on Information Fusion, pages 1823-1828, Cologne, Germany, 2008.