

# Development Tools in Component-Based Structural Computing Environments

Uffe Kock Will  
Department of Computer Science and Engineering  
Aalborg University Esbjerg  
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark  
ukwill@cs.aue.auc.dk

## Abstract

The purpose of this paper is to present an overview of the development tools available in the Construct component-based structural computing environment. In particular, we focus on the most recent development tool that generates IDL specifications from UML diagrams (UML Tool). The development tools lower the entry barrier for service developers by allowing high level specification of new services in UML or IDL and by auto-generating much of the component source code based on well-defined design patterns and templates. The paper is organized into five parts: an introduction to the research area, a brief description of the Construct development environment, a brief overview of UML Tool, a detailed scenario using the development environment, and, finally, our conclusions.

## 1. Introduction

Open hypermedia systems have been around for more than a decade (e.g., [1,4,5,8,16]). A recent trend in open hypermedia research and development is to develop systems as sets of services wrapped in separate service components. These component-based open hypermedia systems are more flexible, in particular with respect to developing and adding new services to the open hypermedia environment. Another recent trend is to provide different types of structure services (in addition to the standard navigational structure service) within the same environment. The term *structural computing* was coined to describe such environments [6]. In 1999 a workshop series was initiated to discuss important issues and advance the thinking about structural computing [7,11].

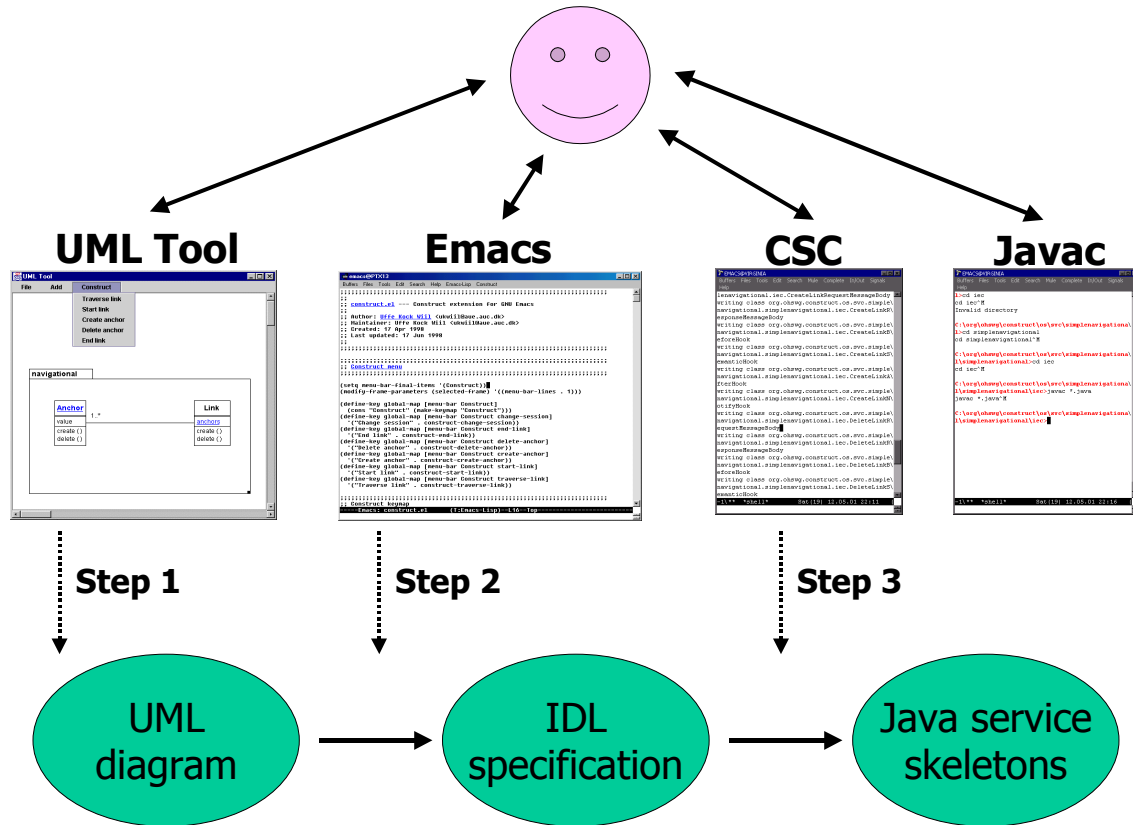
Construct [3,13] is a component-based structural computing environment developed as the common code base successor of the DHM [4], HOSS [8], and HyperDisco [16] open hypermedia systems. Construct is an attempt to take the best of the previous approaches and combine them into one environment. A strong part of HOSS was its development tools, the protocol definition compiler (PDC) and the generalized process template (GPT) [8]. The purpose of these tools was to assist the developer in creating new services for the environment. Following from this, an important point of focus in the Construct project is the provision of tools that can assist service developers in the process of developing new service components [15].

Development of new functionality oftentimes requires expert knowledge of the target system. The goals with the Construct development environment is to lower the entry-barrier for developers by basing the environment on components that use well-defined design patterns and templates. The development tools take care of many of the complex design decisions and coding tasks. The developer does not need to know details of synchronization, multi-threaded programming, or remote method invocation. New service interfaces are defined in high-level specifications with the help of dedicated development tools. Only a moderate level of programming expertise is required to develop new service components in Construct.

The paper is organized into the following sections. Section 2 gives an overview of the Construct development environment. UML Tool is described in Section 3. Section 4 provides a development scenario and Section 5 concludes the paper.

## 2. Construct Development Environment

Figure 1 provides an overview of the development tools available in the Construct environment and the first few steps in the process in which they are deployed. The circle at the top represents the service developer using the development tools (represented by screen shots). The ovals at the bottom represent products at different stages in the development process.



**Figure 1.** The first few steps in the development process with the Construct development tools. The remaining steps in the development process are the classical tasks of coding and compilation. Emacs can also be used for those steps.

The overall steps in the development of new services in the Construct structural computing environment are:

1. UML Tool is used to create a UML diagram specifying the interface of the required service. When the diagram is ready, UML Tool can generate an IDL specification from the UML diagram. UML Tool is a client of the Construct navigational structure service. Both class names and field names can serve as endpoints of links.
2. Emacs is used to display and edit IDL specifications. If the developer prefers to write an IDL specification of the service interface directly (instead of starting with a UML diagram), then this step is the first step in the development process. Emacs is a client of the Construct navigational structure service as well as the Construct metadata service [12]. Arbitrary text strings can serve as endpoints of links. Each file in Emacs can have a metadata record attached. A metadata record consists of arbitrary key/value pairs.
3. The Construct Service Compiler (CSC) transforms the service interface specification in IDL into a Java skeleton service component. The CSC runs inside a shell tool (e.g., Emacs).

4. Emacs is used to edit the Java source code in the skeleton service. The developer creates the semantic parts (method bodies) of the operations defined in the service interface. When the semantic parts are added to the skeleton service, the service is fully developed.
5. The Java source code for the skeleton service is compiled into a Construct service component that automatically is operational in the Construct structural computing environment. The Java compiler (Javac) runs inside Emacs in its own shell.

As illustrated, the developer can decide to work at different levels of abstraction when specifying the service interface. Services interfaces can be specified graphically as UML diagrams in UML Tool or textually as IDL specifications in a text editor (e.g., Emacs). The use of the development tools makes the coding step (Step 4 above) easier. Many difficult design decisions are made by the tools and much of the complex code is generated by the tools based on the interface specifications.

### 3. UML Tool

UML Tool is a development tool that raises the level of abstraction in service development. New Construct service components can be specified in UML diagrams defining the service interfaces. UML Tool is described in detail in [2]. The user interface of UML Tool consists of a white canvas and several operations grouped into three menus (Figures 2 through 4).

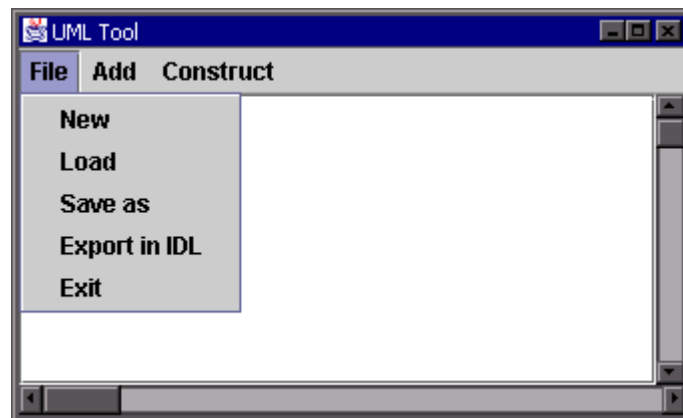


Figure 2. The *File* menu in UML Tool.

The *File* menu provides operations to create a new UML diagram (*New*), to load an existing UML diagram from a file (*Load*), to save a diagram to a file (*Save as*), to translate a UML diagram into IDL and store it in a file (*Export in IDL*), and, finally, to exit UML Tool (*Exit*). The *Load*, *Save as*, and *Export in IDL* operations open standard file selector windows to specify/select files in the file system.

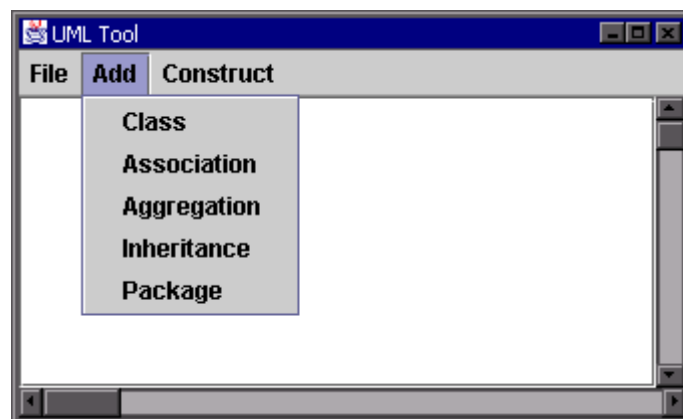
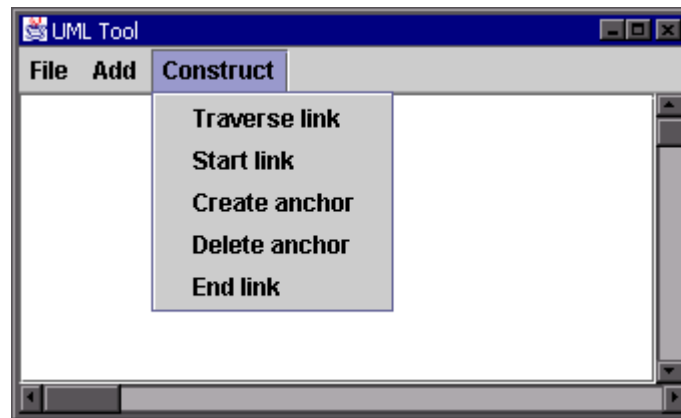


Figure 3. The *Add* menu in UML Tool.

The *Add* menu allows the UML abstractions to be created. UML Tool supports packages, classes and different relationships between classes (association, aggregation, and inheritance). Classes can be grouped inside packages. Classes and relations that have been created can be manipulated (updated or deleted) by double-clicking or right clicking on them. Classes can be moved around on the canvas by clicking on them and dragging them to a new location.



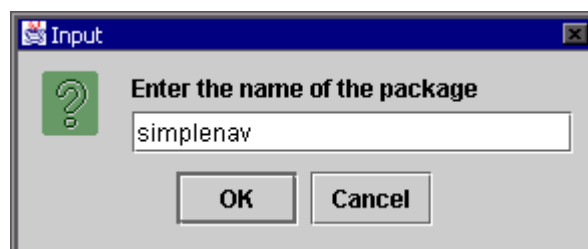
**Figure 4.** The *Construct* menu in UML Tool.

The *Construct* menu provides the interface to the Construct navigational structure service with five basic operations to create, delete and traverse links. Links are created in the following sequence of operations: *Start link*, select endpoint (class or field name), *Create anchor*, select endpoint, *Create anchor*, and *End link*. The second endpoint can be either inside UML Tool or inside another client of the Construct navigational structure service (e.g., Emacs, Microsoft Word, or Netscape 6). Links are traversed by selecting an endpoint (anchors appear as blue, underlined text in UML Tool) and initiating the *Traverse link* operation.

## 4. Development Scenario

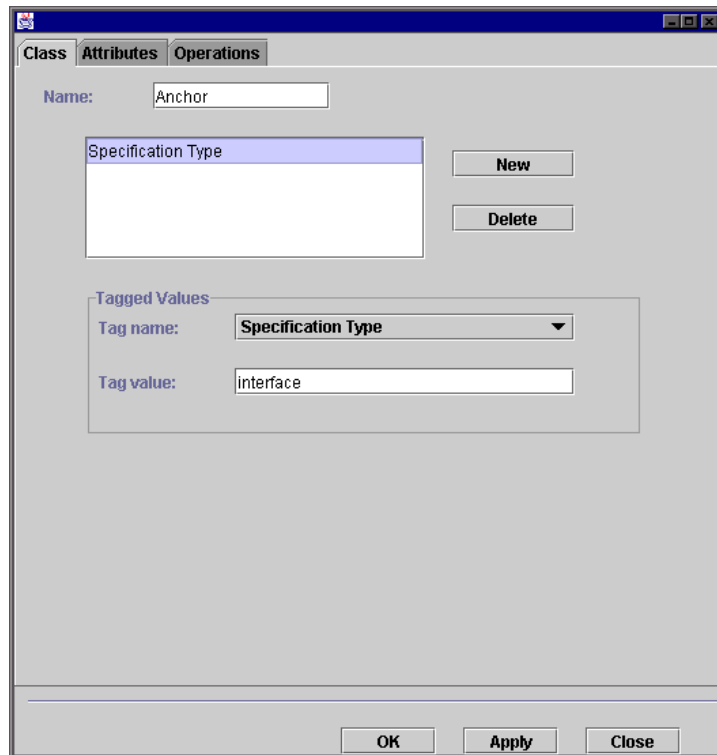
This following scenario shows some of the steps involved in the development of a simple navigational structure service using the Construct development tools. The focus of the scenario is on the specification of interfaces using UML Tool. The remaining stages in the Construct development process are described in [14], which contains details about the development of a file-based storage component.

Jakob is a graduate student at some university. He decides to follow the structural computing course given by the Department of Computer Science. As a part of the course, students are required to get hands-on experience in developing structure services. Jakob decides to develop a simple navigational structure service. Jakob has no prior knowledge of hypermedia or structural computing systems and decides to make use of the public domain Construct development tools to assist him in the development process. The first step is to make the overall design of the simple navigational structure service. Jakob decides that the service should support two types of structural abstractions: *Anchor* and *Link*. He finds this sufficient for his simple model. Jakob now uses UML Tool to assist him in the specification of the simple navigational component. He starts by defining a package named *simplenav* (Figure 5).



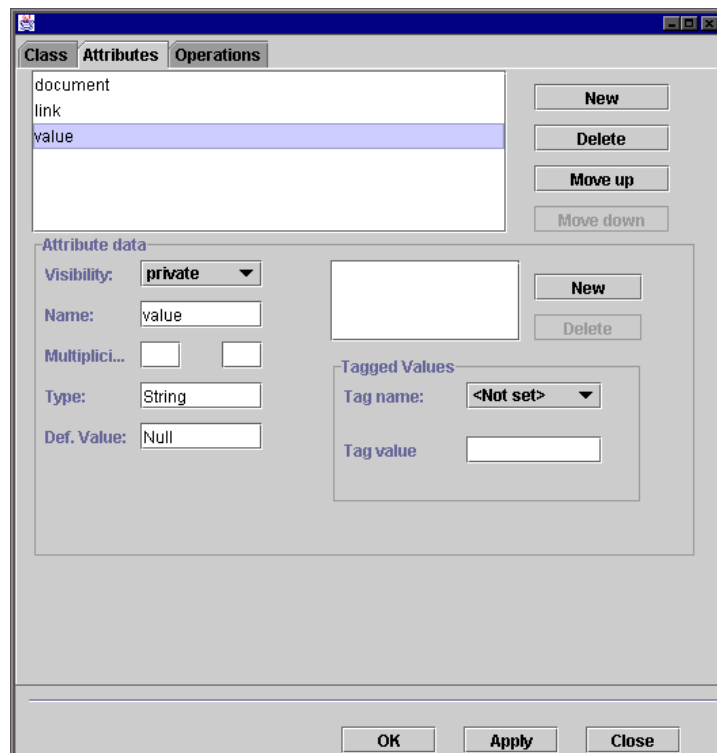
**Figure 5.** The *Add Package* interface in UML Tool.

Then, he creates the **Anchor** interface (Figures 6 through 8) and, subsequently, the **Link** interface. In Figure 6, Jakob has entered the name “Anchor” and has specified that **Anchor** is an interface.



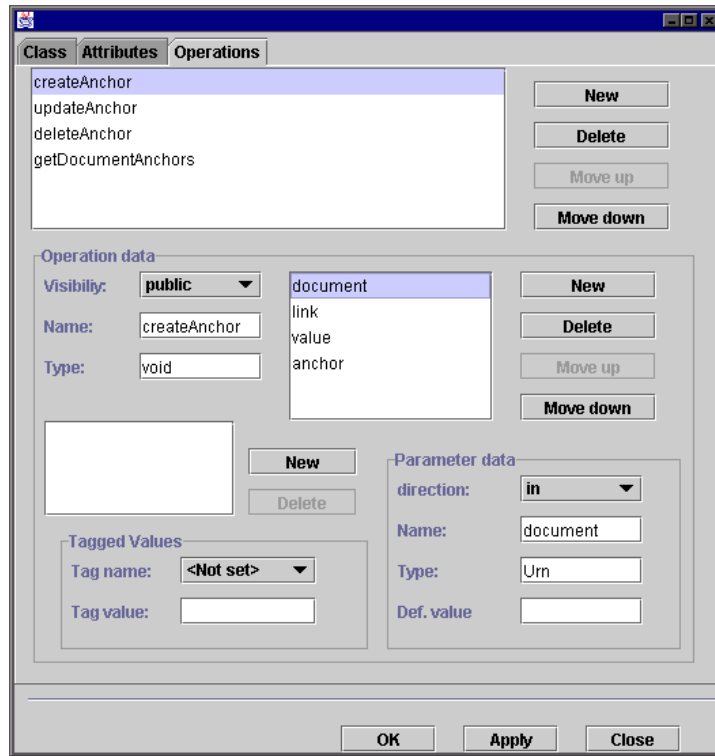
**Figure 6.** The *Add Class* interface in UML Tool.

In Figure 7, Jakob has added three attributes to the **Anchor** interface (*document*, *link*, and *value*). The attribute named *value* is private in scope, is of type String, and has the default value Null.



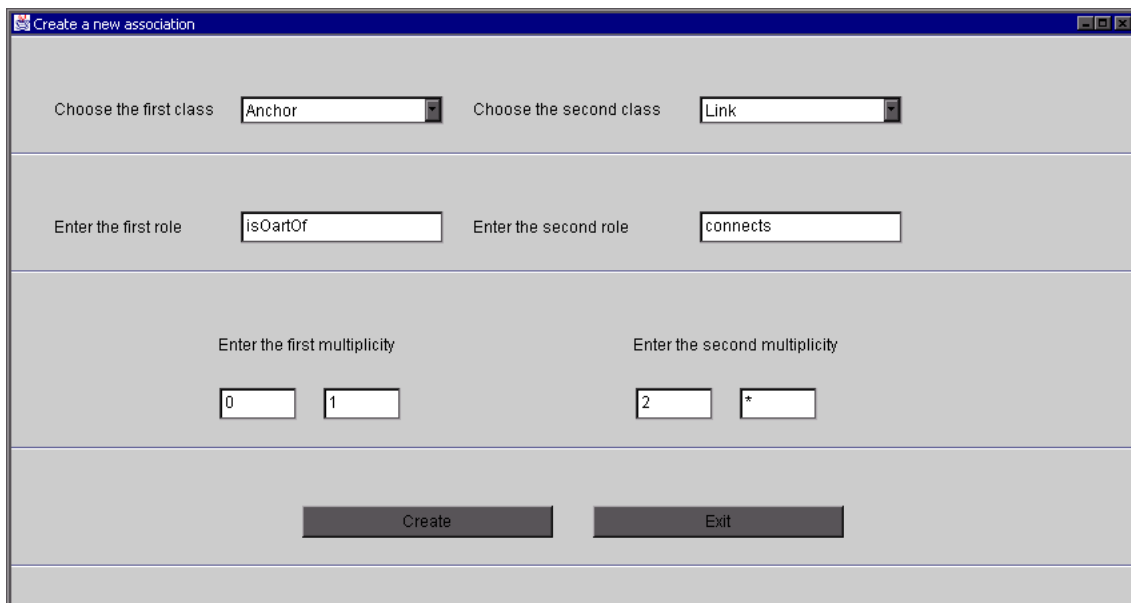
**Figure 7.** The *Add Class Attributes* interface in UML Tool.

In Figure 8, Jakob has added four operations to the **Anchor** interface (*createAnchor*, *updateAnchor*, *deleteAnchor*, and *getDocumentAnchors*). The *createAnchor* operation is public in scope, is of type void, and has four parameters (*document*, *link*, *value*, and *anchor*). The *document* parameter is an input parameter (indicated by the direction In) of type Urn.



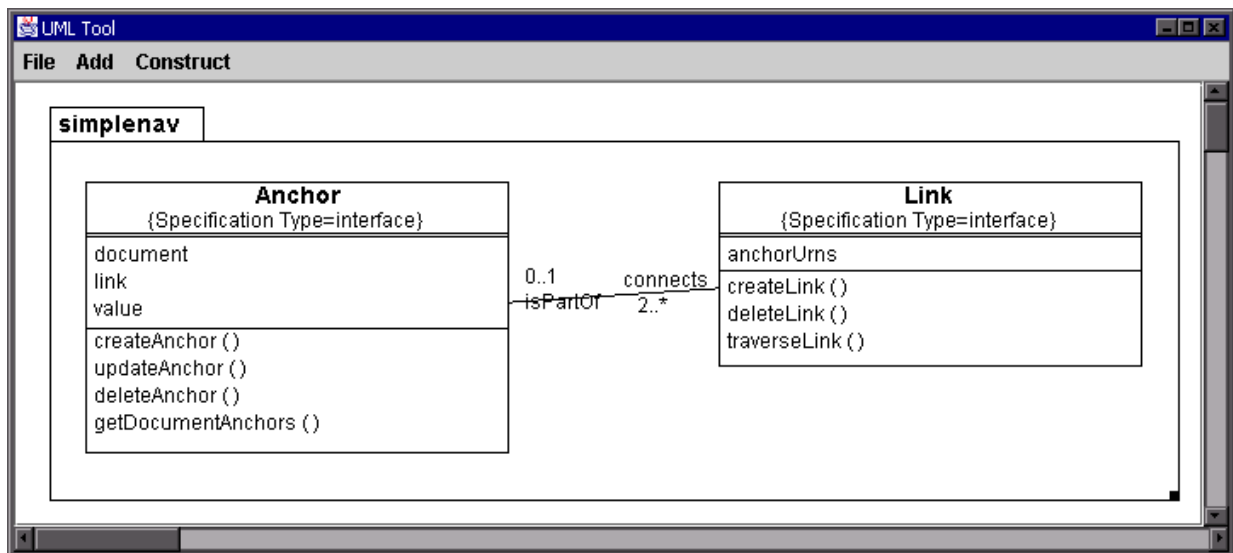
**Figure 8.** The *Add Class Operations* interface in UML Tool.

Finally, Jakob creates an association between the **Anchor** and **Link** interfaces (Figure 9). He specifies that an anchor instance is part of 0 or 1 link instances, and that link instances connect 2 or more anchor instances.



**Figure 9.** The *Add Association* interface in UML Tool.

The resulting UML diagram for the simple navigational structure service interface is shown in Figure 10.



**Figure 10.** The resulting UML diagram of the simple navigational service in UML Tool.

The next step in the development process involves the translation from the UML diagram to an IDL specification. Jakob initiates the *Export in IDL* command and the IDL specification in Figure 11 is generated by UML Tool and saved in a separate file.

```

module simplenav {

  interface Anchor {
    attribute Urn document;
    attribute Urn link;
    attribute String value;
    void createAnchor(in Urn document, in Urn link, in String value, out Anchor anchor);
    void updateAnchor(in Urn anchor, in String value);
    void deleteAnchor(in Urn anchor);
    void getDocumentAnchors(in Urn document, out LinkedList anchors);
  };

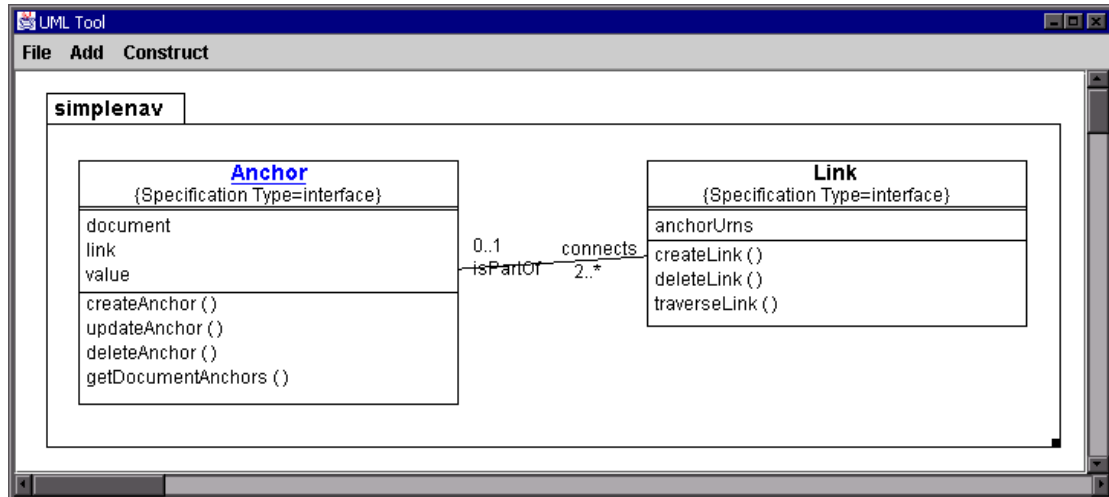
  interface Link {
    attribute LinkedList anchorUrns;
    void createLink(in LinkedList anchorUrns, out Link link);
    void deleteLink(in Urn link);
    void traverseLink(in Urn anchor, out LinkedList anchors);
  };

};
  
```

**Figure 11.** The IDL specification generated by UML Tool.

Jakob decides that he is happy with the interface specification and starts to document the design and the decisions leading to the specification. He creates a text file in Emacs (named *navigational.txt*) describing the interface. Then he creates a two-way link between the **Anchor** interface in the UML diagram (Figure 12) and the phrase “Anchor” in the text file in Emacs (Figure 13). He continues to document the design and creates links between the different types of development artifacts as part of the documentation of the simple navigational structure service interface.

In Figure 12, Jakob has started to create a link from the **Anchor** interface.



**Figure 12.** A link endpoint created from the **Anchor** interface in UML Tool.

In Figure 13, Jakob has finalized the link from the **Anchor** interface in the UML diagram to the phrase “Anchor” in the text file in Emacs containing the documentation about the **Anchor** interface.

```
EMACS@VIRGINIA
Buffers Files Tools Edit Search Help Mule Construct
The navigational domain is supported by two structural elements,
namely link and anchor.

The anchor has an attribute named value where it stores the region in
an artifact that is the the endpoint of a link. A region could be a
position in a text file such as "5 10" - meaning that the anchor point
is the text from character 5 to character 10.

Anchors can be created and deleted.

The link stores a set of anchors...

--\-- navigational.txt Tue(19) 8.05.01 13:54 (Text)--L9--All-----
```

**Figure 13.** The other link endpoint created from the phrase “Anchor” in the file “navigational.txt” managed by Emacs.

The remaining steps in the development process are to use the CSC to generate Java source code, to add the method bodies, and to compile the Java source code. As mentioned above, [14] contains a detailed example of how the CSC works, what source code it generates, etc., based on the development of a file-based storage component.

## 5. Conclusions

The paper has examined how the development tools (in particular UML Tool) can assist service developers in developing services for the Construct component-based structural computing environment. We have demonstrated that the development tools lower the entry-barrier for structural computing service developers. New services can be specified as UML diagram or IDL specifications and many difficult design decisions relating to the coding process are automatically handled by the development tools.

The development tools share common features and goals with commercial development environments such as Rational Rose [10]. We have found two noticeable differences between our approach and approaches like Rational Rose. First of all, the Construct software is publicly available free of charge.

Secondly, we have taken an open systems approach to the development environment, which aims at re-using and integrating with existing applications, tools, and services in the computing environment.

Our effort has resulted in a set of stand-alone tools that can be deployed in different phases of service development. In our future work we plan to examine ways to achieve a smoother integration of the development tools by deploying control integration techniques. A simple example of control integration is to have UML Tool initiate the CSC with the generated IDL specification. This will allow developers to generate the service component skeleton from the UML diagram in one step.

Based on the experiences with the Construct development tools, we would like to propose that the Open Hypermedia System Working Group [9] adopt UML as one of its primary ways to specify service interfaces – together with IDL.

## Acknowledgments

Several people have been involved in the development of Construct tools and services. Peter Nürnberg developed the Construct Service Compiler (CSC). Stéphane Blondin and Jérôme Fahler, two Socrates exchange students from the University of Bretagne Occidentale in Brest, France, implemented the UML Tool during a three-month visit at the Department of Computer Science and Engineering, Aalborg University Esbjerg during the Summer of 2000.

## References

1. Anderson, K. M., Taylor, R., and Whitehead, E. J. 1994. Chimera: Hypertext for heterogeneous software environments. In *Proceedings of the 1994 ACM European Conference on Hypertext*, (Edinburgh, Scotland, Sep.), 94-107. ACM Press.
2. Blondin, S., Fahler, J., Wiil, U. K., and Nürnberg, P. J. 2000. UML Tool: High-level Specification of Construct Services. Technical Report CSE-00-01, Department of Computer Science and Engineering, Aalborg University Esbjerg.
3. Construct. 2001. <http://www.cs.aue.auc.dk/construct>.
4. Grønbaek, K., and Trigg, R. 1999. *From Web to Workplace – Designing Open Hypermedia Systems*. MIT Press.
5. Hall, W., Davis, H., and Hutchings, G. 1996. *Rethinking Hypermedia – The Microcosm Approach*. Kluwer Academic Publishers.
6. Nürnberg, P. J., Leggett, J. J., and Schneider, E. R. 1997. As we should have thought. In *Proceedings of the 1997 ACM Hypertext Conference*, (Southampton, UK, Apr.), 96-101. ACM Press.
7. Nürnberg, P. J., Ed. 1999. *Proceedings of the First Workshop on Structural Computing*. Technical Report CS-99-04, Department of Computer Science, Aalborg University Esbjerg, Denmark.
8. Nürnberg, P. J., Leggett, J. J., Schneider, E., R., and Schnase, J. L. 1996. HOSS: A new paradigm for computing. In *Proceedings of the 1996 ACM Hypertext Conference*, (Washington, DC, Mar.), 194-202. ACM Press.
9. Open Hypermedia System Working Group. 2001. <http://www.ohswg.org>.
10. Rational Rose. 2001. <http://www.rational.com/rose>.
11. Reich, S., and Anderson, K. M. Eds. 2000. *Open Hypermedia Systems and Structural Computing. Proceedings of the 6th Workshop on Open Hypermedia Systems and the 2nd Workshop on Structural Computing*. Lecture Notes in Computer Science 1903, Springer Verlag.

12. Wiil, U. K., Hicks, D. L., and Nürnberg, P. J. 2001. Multiple open services: A new approach to service provision in open hypermedia systems, In *Proceedings of the 2001 ACM Conference on Hypertext*, (Århus, Denmark, August). ACM Press.
13. Wiil, U. K., Nürnberg, P. J., Hicks, D. L., and Reich, S. 2000. A development environment for building component-based open hypermedia systems. In *Proceedings of the 2000 ACM Hypertext Conference*, (San Antonio, TX, Jun.), 266-267. ACM Press.
14. Wiil, U. K. 2000. Using the Construct development environment to generate a file-based hypermedia storage service. In [11], 147-159.
15. Wiil, U. K., and Nürnberg, P. J. 1999. Evolving hypermedia middleware services: Lessons and observations. In *Proceedings of the 1999 ACM Symposium on Applied Computing*, (San Antonio, TX, Feb.), 427-436. ACM Press.
16. Wiil, U. K., and Leggett, J. J. 1997. Workspaces: The HyperDisco approach to Internet distribution. In *Proceedings of the 1997 ACM Hypertext Conference*, (Southampton, UK, Apr.), 13-23. ACM Press.