Development of a CASE-tool for the Service-Based Software Construction

M.Zinn, K.P.Fischer-Hellmann and A.D.Phippen

Centre for Security, Communications and Network Research,
University of Plymouth, Plymouth United Kingdom
e-mail: mail@marcuszinn.de, K.P.Fischer-Hellmann@digamma.de,
andy.phippen@plymouth.ac.uk

Abstract

In today's software development, applications for computer aided software engineering (CASE) are widespread and necessary. Most procedure models or technologies use CASE-tools. This publication shows the conversion of the fundamental idea of service-based software construction into a software system. The focus is set on the system architecture, the fundamental information model and the transformation model based on it.

A goal is to represent the needed information to build up and use a CASE-tool for service-based software construction.

Keywords

Computer aided software engineering (CASE), service-based software construction process, software construction artefact, unit of modelling, transformation, integration

1. Introduction

Software development processes as well as the actual software development are in many cases supported by applications (CASE-tools). Well-known CASE-tools are e.g. Innovater, Rational Rose, Enterprise Architect und Together. Typical contents are support modes like for example graphic user interface, information preparation, input simplification (Wizards) and automation of process steps. (Deneva, 1999)

Some applications are single technology applications. That means these applications usually support only one type and/or technology of software development processes/software development in form of an executable application. Other CASE-tools offer the possibility of extension.

Typical examples of this kind of applications are development environments. With the consideration of Eclipse, for example, it becomes clear that various technological approaches can be realised. These include component- and service technology as well as software development approach models like model driven development. One of the most important factors is the interoperability between different CASE-tools and the supporting applications and processes (IEEE, 2007) (Garcia-Magarino and Gomez-Sanz, 2008). (Deneva, 1999) and (Deneva and Terzieva, 1996) show that CASE-tools are an important factor of success for approach models or technologies

since this supports the propagation of the approach by a practical application. The important factors are: Simplification in handling, possible integration in available tools or environments and easiness to in understanding.

The service-based software construction process is an add-on for existing procedure models. It supports the developer in reusing software units (classes, services and components) and all important information like documentation and specification which is related to these units. In addition, the reuse is supported by a transformation model, which can transform software units into another form. This reuse is an added value because it offers more information from a single data. Furthermore, this reuse is build upon a new information model. As a novelty this data management and transformation system is provided by a single service. As pointed out in a previous publication, software development is not a question of the location of a developer or the needed data. The question is how the developer can handle this data (Zinn, 2007).

The aim of this publication is to show which models and architecture can be used to create a CASE-tool for the support of the service-based software construction process. This process is the underpinning methodology. Therefore, it is necessary to explain the basic concepts. The next section shows an overview of different models and technologies needed for the service-based software construction. This includes the explanation of artefacts, units of modelling, service-based software construction procedure model, information model, transformation model and integration model and builds a common understanding. Following this, a section demonstrates the structure of the CASE-tool built upon the explained models and technologies.

2. Basic Concepts

2.1 Artefacts and units of modelling

For the explanation of the application shown in this publication, it is necessary to define some terms. In the area of software development, which deals with composing of bigger software units (McConnell, 1996), the term “unit of modelling” is used (Wang and Fung, 2004) and (Zinn, 2008). The service-based software construction uses components, services and classes (objects) as units of modelling (UOM). This shows the scope of this kind of software development: Objects (Object-oriented construction), Components (Component-based construction) and Services (Service-oriented construction). See (Sommerville, 2007), (Szyperski et al., 2002), (Papazoglou et al., 2007) and (W3C, 2004) for the used definitions classes(objects), services and components.The service-based software construction process extends the view of UOMs. As a result, a UOM is not only the implementation of a software unit. A UOM in this area is a container for the implementation and all other information which depends on the implementation and which is important for the reuse. For example, (Pfleeger and Atlee, 2009) show that documentation, specification and test information are also important for reuse. Another very important reason for the storage of all this data is the search for UOMs. Reusing a UOM is easier than to find it again. The “correct” search for data is very complicated and is based on the metadata that can be used for the search. Today semantic search, based on different ontologies, is very popular. This means data and definitions will

be connected in one database by the use of semantics. These semantics can be used to find data entities (see (Stuckenschmidt, 2009) and (Hitzler, 2008)). Important information of a UOM is "Transformation". A UOM does not only contain describing information, but also supports to carry information which can transform UOM data. A transformation can be, for example, a transformation of one technology into another such as transforming Java byte code into .NET byte code (Frijters, 2008) or a UML diagram into Java source code. Thus the original UOM can be used in another (technology) domain. The service-based software construction process focuses on the transformation of implementation data. Moreover, it can be necessary to transform describing information. The service-based software construction process defines "Transformation" like the definitions of Mode Driven development (MDD/MDSD) (Meimberg et al. 2006), (Stahl, 2007) and Generative Programming (GP) (Czarneck and Eisenecker, 2000). Transformation means in this area that information will be transformed into the same or another domain specific model for later reuse. (Garcia-Magarino, 2008) shows different modelling languages and frameworks in the area of CASE-tool interoperability. (Czarnecki and Helsen, 2003) show a classification of different model transformation attempts. Within the scope of this research MDD and GP are focused. In this publication transformation is done by using existing transformation tools.

Another important term is "Software Construction Artefact (SCA)". An SCA is a container for UOMs which corresponds to the same problem area. For example: An Artefact for calculating the mathematical GAUSS function carries 3 UOMs. The first is a Java Class, the second a .NET component and the last one a Web service. This example shows that the common scope is important to put UOMs into the same artefact and not the technical view on the UOMs.

2.2 Service-based software construction procedure model

The service-based software construction process is the underpinning methodology for the software described by this paper. It contains a maximum of four possible phases.

1. Development of an outer structure (Precondition phase)
2. Choice of the units of modelling and their transformation rules
3. Creation of missing artefacts / units of the modelling (optionally)
4. Transformation of the units into the external structure

The first phase is to create an outer structure. This means the developer has to create, for example, a class structure which can be extended by adding UOMs. In the second phase the user searches for artefacts which correspond to his search criteria. The list of possibly suitable artefacts is then examined closer to identify certain units of the modelling which can be used. In addition, the units of modelling contain transformation data which can be used to customise the unit. If the research proves that components are missing or no components can be found in the available repositories, the components must be created independently. This means that step three is an optional step. When all components are identified, they are transmitted into the created external structure (see first phase) by means of the transformations selected in phase two. The creation of the external structure is a step preceding the

software construction. The service-based software construction process does not define the creation of the outer structure but uses it as a precondition. Example: A developer wants to create a small program which is able to calculate Pi. To do this he starts to create an empty class structure (outer structure). Afterwards he continues by searching a UOM for calculating Pi. The result of the search is an SCA which provides him four UOMs: A webservice, a java component, a C# Class and a transformation which transforms the C# Class into a VB class. The developer selects the webservice and adds it to the outer structure. The phases 2, 3 and 4 also represent the procedure model. Through the consideration of this procedure model the Use Cases become visible. This, however, only constitutes summarised Use Cases: 1. Select UOM, 2. Transform artefact and 3. Create UOM.

2.3 Basic information, transformation and integration model

The fundamental information model defines the contents of a software construction artefact. A software construction artefact contains different implementations of the three kinds of units of modelling, but all implementations refer to the same problem area. Each unit contains "legible", "illegible" and "reference" information. Legible information serves the user during the identification and transformation phase. This can be documentation information, cost information, specification information and modelling information. "Illegible" data are information which can be ascribed to the automated use by the machine. Thereby, component-specific and component-unspecific information are distinguished. Specific data are implementation (with objects) and binary data (with components). Unspecific data can occur with more than one component kind such asinterfaces, UnitTest and transformation data. In figure 1 the illegible data is described as “NRDataType” and legible data as “RDataType”.

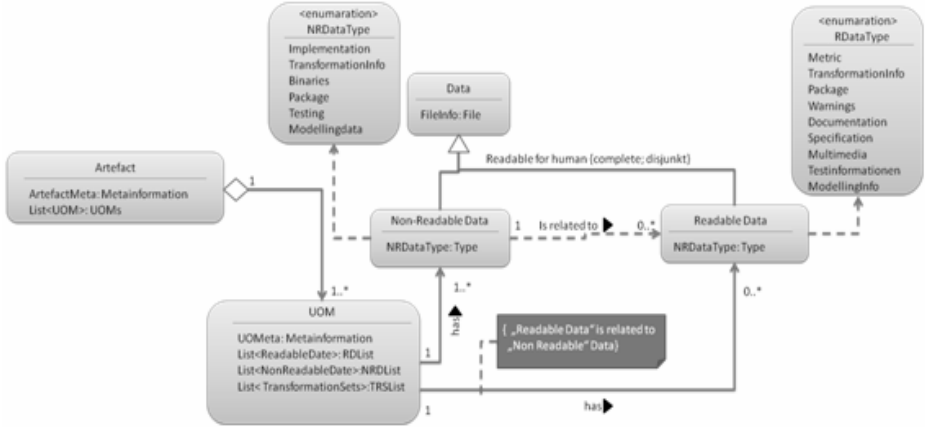

Figure 1: Information model core

This figure also shows a sketch of the information model core. The important statement is the relationship between the Artefact and the UOMs. Also the content of the UOM is important. Beside the basic data model, there is one more transformation and integration model. The transformation model describes the input and output data

as well as the transformation rules of a transformation. The input data are a special subset of data from the information model whose contents depend on the respective unit of modelling. A transformation rule is the description on how to transform the input into the output. Since this area has not deepened further in the actual research yet, an application-supported transformation is anticipated in this case. Therefore, a transformation needs a set of parameters and the respective start-up files which launch a transformation. The output results from the used transformation rule or the used transformation application. This scenario requires an enhancement of the basic information model. UOMs must contain transformation sets and these sets are composed of transformation rules. Figure 2 shows the enhancement of the information model with transformation entities. So a UOM can have transformation sets, which consist of different transformation rules. Input and the output of these rules are file based data.

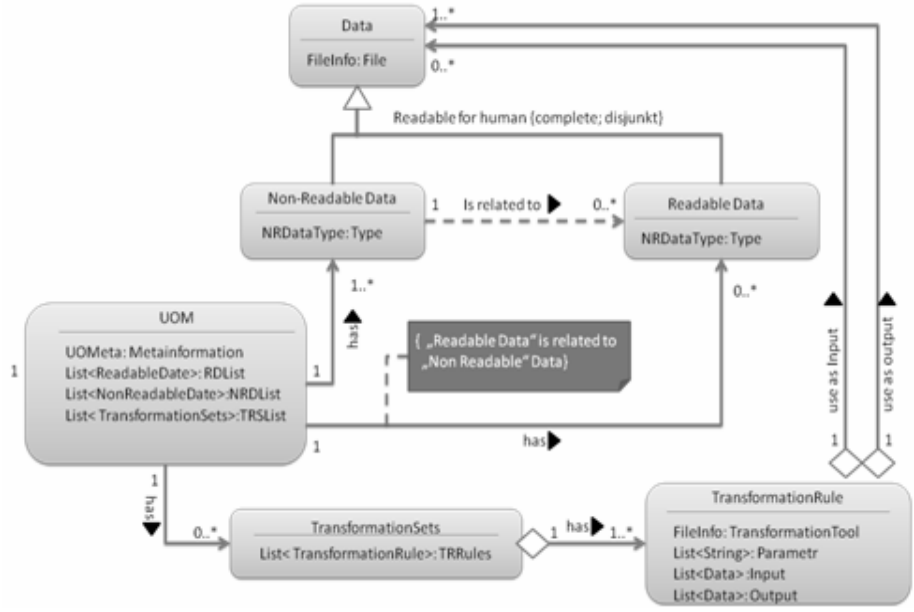

Figure 2: Transformation entities in the information model core

After the transformation of data, these must be integrated into the respective development environment. Therefore, a model is required which recognises the destination structures and mounts the input data. Figure 3 sketches the relationship between the transformation and integration model. The left side shows the transformation concept which is explained above. The right side illustrates the integration into an IDE, whereas the output files of the transformation are the input files of the integration.

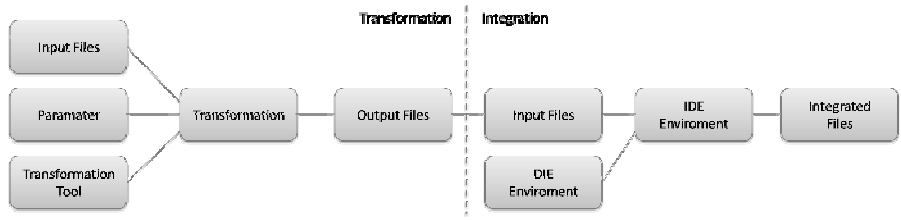

Figure 3: Simple transformation and integration model

3. A service-based software construction CASE-tool

Upon this information, a transformation and an integration model, which are sketched in the last paragraph, a set of CASE-tools were built. This section shows the important points relating to the service-based software construction process.

3.1 System architecture

During the development of the application, the following system architecture was realised (see figure 4):

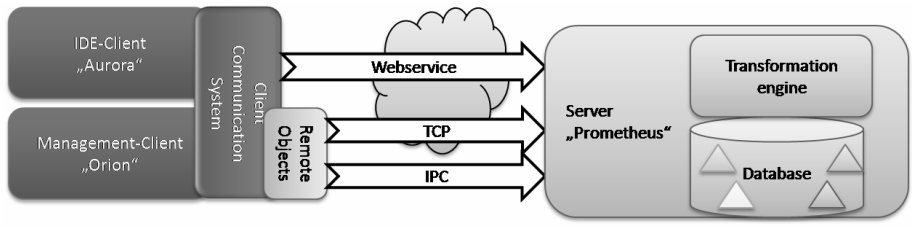

Figure 4: System Architecture

Hereby, the server provides the administration of artefacts and their information, for example, save, delete and search of artefact information. Here, artefact information is saved in a data base. In addition, the server contains the transformation engine. The packaging and provision of this engine, the artefact database and the search functionality through the server as a service belong to the central core of the service-based software construction and represent one novelty of the service-based software construction approach. The management client provides a simple graphic user interface for the server functionality. On one hand, this serves for the management of artefact information and, on the other hand, as an easy test environment of functions (during the development phase). This client is available as a stand-alone application and as an integrated user interface in the development client.The development environment client serves for the use of the server functions within an IDE as a CASE application. This client is able to query for artefact information. The representation of detailed information occurs in an additional web browser as a window within the IDE or in an external browser. In contrast to the management client, the development client does not influence the artefacts such as the deletion of an artefact on the server. The real job beside the search of artefacts and units of

modelling is the transformation and the integration of transformation results in the current development project. Three different forms as a communication-form are possible : Inter process channel (IPC) (Microsoft, 2009), Transmission Control protocol (TCP) (Microsoft, 2005) and a web service. The IPC serves as a remote means of communication between processes within local systems, as for example client and server are executed on the same machine. In contrast, TCP serves for the communication between client and server and, as a result, the programs can be located on different systems. At closer inspection it can be recognised that IPC and TCP constitute the basic communication methods. The web service, shown in figure 4, serves the provision of a uniform interface and uses the basic communication methods again.

3.2 Interfaces

The system exhibits interfaces in two points: 1. with the use of the remote object for service-based construction. 2. With the use of the web service for service-based construction. The first includes features such as create, update and create artefacts or UOMs. In addition, another remote object for the control of the server is available. This is not explained closer since it concerns standard methods for the control of an application (for example restart and stop). Figure 5 shows the interfaces of the remote objects and the web service.

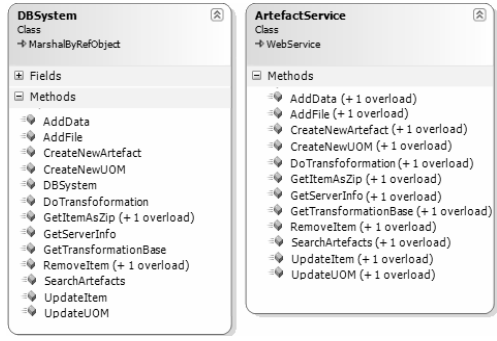

Figure 5 – Remote object and web service interfaces

The basic methods of the interface DBSystem exhibit self created data structures (classes) as parameters which are technology specific (in this case .NET technology). To guarantee the compatibility of the SCA and UOM description with other technologies, all self made (description)classes were made serialisable in the first step. An example for this is that the objects of such classes can be transformed into a form of XML. Since XML is basically a text based form, it can also be converted into other technologies, for example, by the use of web services. Furthermore, attention was paid to the fact that all basic types used in this approach are also serialisable. In the second step, overloaded methods (secondary methods) were developed which contain only serialisable data. This includes, for example, that parameters as well as return values are textual information. Furthermore, serialisation is important to exchange describing data. Figure 6 shows an example of a simplified serialised SCA description which includes a UOM and meta information.

```
<Artefact TypeOfArtefact="Function" ID="1b929d23-ec39-4926-856f-50cc7bf097b4" CreationDate="03.07.2009 08:58:51"
         Path="1b929d23-ec39-4926-856f-50cc7bf097b4">
  <ListOfUOMS>
    <UOM UomType="Component" ID="ec3e9c49-625c-415e-8c49-c400a00f9614" Path="ec3e9c49-625c-415e-8c49-c400a00f9614">
      <TransformationSets />
      <ListOfFileStructure />
      <Metainformation Author="TestInformation" Description="TestInformation" Version="TestInformation" />
    </UOM>
  </ListOfUOMS>
  <Metainformation Categories="MathFunction" Server="Dedalus" WebSite="www.marcuszinn.de"
                   ChangeLog="" Description="Gauss function" Version="1.0">
    <Categories>
      <string>Zinn</string>
    </Categories>
  </Metainformation>
</Artefact>
```

Figure 6: Example of a serialised software construction artefact

3.3 Transformation engine

The transformation engine is part of the repository server and can be used by a web service call. It transforms input information into another form by executing transformation rules sets. These rule sets are stored in UOMs. Each rule set consists of transformation rules. A rule in turn contains a program fetch of a transformation tool, a set of parameters which correspond to the transformation tool and input files (see figure 3). These input files can be files of the UOM or output files of a previous rule in the same rule set. Example: To transform WSDL information into a C# client and server stub, the SVCUTIL tool of the Windows SDK can be used. SVCUTIL needs the parameter "language:C#". So the program fetch is "*svcutil.exe test.wsdl language:C#*". The output of this call is a C# file (test.cs) which includes different server and client classes and all types are needed. The transformation rule set contains only one transformation rule. This rule is based on a transformation base (svcutil.exe) and two corresponding parameters (language:C#) and the input file (test.wsdl) which is part of the original UOM. The developer now selects this transformation and the repository server executes it. The result will be transferred to the user. If the user uses the IDE client, the result will automatically be added to his project environment.

4. Conclusion and future work

The service-based software construction process is an extension of procedure models and focuses on reuse. It consists of three parts and contains technical innovations. The first part is the idea of a common database for software construction artefacts (SCA) and units of modelling (UOM). This means all necessary information of a software unit (class, component, service), such as implementation, documentation, specification and test information, will be stored in a UOM. Thus a UOM includes all data which was produced during its development. This data can be used by different roles (like software architect or developer) and in different development phases (like requirement or development phase) for reuse. An SCA includes all UOMs which refer to the same problem as a solution. Therefore, the developer can easily manage and search for technical solutions. This stored data focuses on reuse. This means that it is a software unit reuse database / content management system. It

is important for reuse to have a single system to manage entities and their knowledge which can be reused. Users, like software architects or developers, use different data to make decisions. If this data is stored at one place, it is easier to find and use it. The second part includes a transformation engine which can transform UOMs. Transformation means that a UOM can be converted into another form, as for example a Java binary code into a .NET binary code. This also implicates that possible transformation rules will be stored inside the UOM. These rules should be able to transform the implementation of a UOM implementation into another technical or domain specific form and give added value for the reuse of the UOM. Thus the reuse database becomes extended with transformation rules. Together with this transformation engine and the underlying transformation model the database changes from a content management system to a software construction system. At the moment no system is existing which stores all UOM data combined with transformation data and functionality to generate an added value. Therefore, the shown system is an innovation. Part three is a service which offers all the database and transformation functionality to the developer. The innovation of this service is that the complete functionality of the construction system (managing, searching and transformation) is capsulated in one single interface. By using a service as a communication node to such a software construction system, different developers can use one single repository from different locations. As an added value of this, the costs for software developing will be reduced. Lower license fees for different content management systems and transformation tools is only one example for this reduction of costs.The system shown in this publication shows the important parts of the service-based software construction process as a technical implementation. It also shows, that the up to now ascertained models (communication, artefact information and transformation) can be realised in a CASE-tool. The reason is that the ascertained Use Cases were realised. That means construction, search, deletion and fitting of artefacts or units of modelling are basically possible. It was also shown that transformation and use of transformations on artefact information is possible.

Some details of the important data model are not shown in this publication if the output of a transformation rule is saved in a stand-alone unit of modelling, such as the solved consistency problem. But this approach still has open question. One of these questions is the need for a semantic search. By using all possible information, as for example documentation and specification with a full text search, the search results will not be very useful. The system shown in this publication uses full text search. But the developer needs a system which only offers him applicable results for his search question. Therefore, semantic search is an important research area for the service-based software construction process. Another important question is the research for other methods of transformation. The transformation model shown in this publication only supports tool-based transformation. It must be analysed whether there are other transformation methods which must be added to this model.

5. References

Czarneck, K. and Eisenecker, U. (2000), “Generative Programming”, Indianapolis, USA Addison Wesley. ISBN: 978-0-201-30977-5

Czarnecki K. and Helsen S., (2003), "Classification of Model Transformation Approaches", *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, pp. 32-49, http://www.softmetaware.com/oopsla2003/mda-workshop.html

Daneva, M. and Terzieva, R. (1996), "Assessing the Potentials of CASE-Tools in Software Process Improvement A Benchmark Study", *Proceedings of the Fourth International Symposium on Assessment of Software Tools*, pp. 104-108**.** Toronto, Canada

Deneva, M. (1999), "A Best Practice Based Approach to CASE-tool Selection", Proceedings. *Fourth IEEE International Symposium and Forum on Software Engineering Standards*, pp. 100-111,Curitiba, Brazil

Frijters, J. (2008), "IKVM.NET Home Page", http://www.ikvm.net/, (accessed 02 2009).

Garcia-Magarino, I. and Gomez-Sanz, J.J. (2008), "Model-based Methodologies Framework for Defining Model Language Metamodels for CASE Tools", *Proceedings of the 2008 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software***,** pp. 14-23, Budapest, Hungary

Hitzler, P., Krötzsch, M., Rudolph, S. and Sure, Y. ,(2008), "Sematic Web: Grundlagen", Springer Verlag, Heidelberg, Germany

IEEE (2007), "IEEE Recommended Practice for CASE Tool Interconnection — Characterization of Interconections", *Software and Systems Engineering Standards Committee*, http://ieeexplore.ieee.org/, ISBN: 0-7381-5233-1

MConnel, S. (1996), "Who cares about software construction.", *IEEE, Software Vol. 13 Issui 1*, 01, pp. 128-129

Meimberg, O., Petrasch, R., Thoms, K. and Fieber, F. (2006), "Model Driven Architecture.", Heidelberg: DPunkt, ISBN: 3-89864-343-3

Microsoft (2009), "Interprocess Communications", http://msdn.microsoft.com/en-us/library/aa365574(VS.85).aspx (accessed 06 2009).

Microsoft (2005), ".NET Remoting 2.0", http://msdn.microsoft.com/de-de/library/bb979586.aspx (accessed 06 2009).

Papazoglou, P., Traverso, P., Dustdar, S. and Leymann, F (2007), "Service-Oriented Computing: State of the Art and Research , available Challenges", http://ieeexplore.ieee.org (accessed 12 2007).

Pfleeger, S. L. and Atlee J. M. (2009), "Software Engineering Theory and Practice", 4th Edition, Prentice Hall, USA, ISBN: 978-0-13-606169-4

Stahl, T., Völter, M., Efftinge, S. and Haase, A. (2007), „Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management", Dpunkt Verlag, Heidelberg , Germany, ISBN: 978-3-8986-4448-8

Stuckenschmidt, H. (2009), „Ontologien", Springer Verlag, Heidelberg, ISBN:978-3-540-79330-4

Sommerville, I. (2007), „Software Engineering", Vol. 8th, Addison-Wesley, Munic, Germany, ISBN: 978-3-8273-7257-4

Szyperski, C., Gruntz, D. and Murer, S. (2002), "Component Software Beyond Object-Oriented Programming", Vol. 2nd Edition, pp.35-48, Addison-Wesley., New York, USA, ISBN 0-201-74572-0

W3C (2004), "W3C Web Service Glossary", http://www.w3.org/TR/ws-gloss/, (accessed 12 2006).

Wang, G. and Fung C. K. (2004), "Architecture Paradigms and their influence and impacts on component-based software system", pp. 902 72a, *Proceedings of the 37th Annual Hawai International Conference on System Science*, ISBN: 0-7695-2056-1

Zinn, M. (2007), "Service based software construction process.", *Proceedings of the 3rd collaborative research symposium on Security, E-learning, Internet and Networking (SEIN '07),* pp. 169-184, Network Research Group, University of Plymouth, Plymouth, GB. ISBN: 978-1-8410-2173-7

Zinn, M. (2008), "Definition of software construction artefacts for software construction.", *Proceedings of the 4th collaborative research symposium on Security, E-learning, Internet and Networking (SEIN '08),* pp 79-91, Network Research Group, Wrexham, GB, ISBN: 978-1-8410-2173-6