Software Contracts for Component-Based Web Engineering

Martin Gaedke
University of Karlsruhe, Germany

Martin Nussbaumer
University of Karlsruhe, Germany

Emma Tonkin
University of Karlsruhe, Germany

INTRODUCTION

As an emerging technology, the Web is full of unique challenges for developers, designers – and engineers. Its use for increasingly complex applications such as e-commerce and banking, involving connection to many data sources, highlighted a number of common difficulties. During the design phase, good modeling of complex sites is difficult; during implementation, the design models are often found to be difficult to translate into an implementation model; afterwards, maintenance and long-term management of the site's evolution offer problems of their own. Qualities such as integrity, security, and usability are often afterthoughts, or left unconsidered – most particularly during the site's post-deployment evolution. Many organizations working for the Web still use ad-hoc, chaotic development and maintenance methods, even though the issues surrounding development are now widely known.

Solving these issues called for software engineering – a discipline covering design, development and use of computer software (Weik, 1989) – to be applied to the Web. Researchers in the field noticed a gap between the granularity of design models and that of implementation models for the Web (Gellersen, Wicke & Gaedke 1997) – that is to say, the models that worked best for designers did not provide the right levels of detail for the implementation stage.

BACKGROUND

Software engineering approaches have been brought to the Web through design models especially suited for Web and other hypermedia technologies, for example OOHDM (Schwabe, Rossi & Barbos, 1996), RMM (Isakowitz, Stohr & Balasubramaninan, 1995), and UML (Conallen, 1999). Nevertheless, the lack of representation of higher-level concepts in the implementation remains.

A proposed solution to this issue came from *compositional design and reuse*, a concept from the software engineering domain within which components are designed such that they may be reused as building blocks. An application may then be developed – composed – from the available components. Should additional components be required, they are developed; they may then be reused in future compositions. This concept resulted in a new sub-discipline: component-based Web engineering (CBWE) (Gaedke & Turowski, 2002).

Components seem to solve the granularity problem, as components can be described in whatever granularity suits both parties. For example, a component might represent a formatted paragraph of text, a navigational menu, an active component such as a calendar – or a single link. Components can represent a workflow as a set of pages, forms, and underlying "business logic". Once developed, such components can be stored and reused. Components can even represent an assemblage of smaller components.

The advantages of this approach, christened the *WebComposition* approach (Gaedke, 2000), were demonstrated by the development of proof-of-concept language WCML, the Web-Composition Markup Language (Gaedke, Schempf & Gellersen, 1999), which offers a convenient way to define and represent components. It uses the eXtensible Markup Language (XML), an almost ubiquitous standard, for this purpose. WCML permits XML-based definition of components, associated typed attributes (name-value pairs) referred to as *properties*, and relationships between components. Components are stored in a *repository*, a dedicated data store, and referred to by a unique ID (UUID). These component assemblies provide WebComposition services or in short services – for example, an ordering service is made up of a form interface and a product database. Additional components provide business logic to control the order process, thus enabling the concept of constructing Web applications based on the services they provide. This abstract definition allows for different implementations of the

WebComposition service concept, for example using components, Web services, or even the secretary phone in the office.

WCML was developed as part of a complete approach, which defined a disciplined procedure of composing Web applications with components based on the WebComposition component model (Gaedke, 2000). It is a synthesis of a component-oriented process model with a dedicated Web application framework, reuse management, and dedicated component technology. The details of the reuse model are discussed in Gaedke and Turowski (2002).

DROWNING IN INFORMATION – AND STARVING FOR KNOWLEDGE?

Just as in R.D. Roger's popular quotation, one component in a repository resembles a small needle in a big haystack. A designer looking for an appropriate component in the repository needs a good way to find it, so that he or she can retrieve it using its unique ID. What methods exist to sort or index components within a repository, and what techniques permit components to be most usefully described and specified?

WebComposition services, and the components of which they are built, can be represented in several ways that use searchable indexes to direct the user to the UUID of likely components: controlled indexing, uncontrolled indexing, and methods containing semantic information. Controlled indexing refers to the use of controlled vocabularies or categorizations, such as hierarchical or taxonomical classification or *faceted* classification, where objects are described by characteristic. For example, facets of this encyclopedia entry include its publication date, the expected knowledge of its readers, and its subject type. Uncontrolled indexing refers to methods without the constraint of a controlled vocabulary, such as free-text keywords added to the object's description by a human or a computer, or attribute-value pairs. Examples of both are shown in Figure 1. Controlled and uncontrolled indexing methods were compared (Frakes & Pole, 1994) and both were found to be useful – but, the authors concluded, no single method is perfect. A good result often comes from mixing several methods.

Unlike most types of data, components have an additional property – behavior. As elements in a software application, they have to perform a task, and they must work reliably and accurately – and they must interoperate correctly with other components. In order to ensure this, *software contracts* were introduced in 1988 by Meyer, as part of the Eiffel programming language (Meyer, 1988). Software contracts may include information on many levels, depending on the context of the agreement; configurable values may be negotiated as part of the contract.

Software components are obligations to which a service provider (a component) and a service consumer (a client) agree. The provider guarantees that a service it offers, under certain conditions – such as the provision of appropriate data – will be performed to a guaranteed quality. Furthermore, it also provides information about certain external characteristics, such as its interfaces. The following sections will discuss the different levels for specifying a component, as shown in Figure 2.

Figure 1. Classifying an article