European Journal of Information Systems (2007) 16, 448–459

www.palgrave-journals.com/ejis

Model-driven component adaptation in the context of Web Engineering

Nathalie Moreno Vergara[1], José M. Troya Linero[1] and Antonio Vallecillo Moreno[1]

[1]*Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Malaga, Spain*

Correspondence: Nathalie Moreno Vergara, Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, ETSI informatica, Campus de Teatinos, Malaga 29071, Spain.
Tel: +34 95 213 2846;
Fax: +34 95 213 1397;
E-mail: vergara@lcc.uma.es

Abstract
Currently, Web-based applications are no longer monolithic and isolated systems but, rather, distributed applications that need to interoperate with third-party systems, such as external Web services, LDAP repositories or legacy applications. When one component provides a service that the Web application requires, it is often not possible to bind the two systems together if they were not programmed to have compatible collaboration specifications. Modeling the adaptation between a Web application and external assets becomes therefore an essential issue in any realistic model-driven development scenario. However, most of the existing Web Engineering proposals do not take this issue into account, or they simply address it at the implementation level (in a platform-specific way). In this work, we discuss the problems involved in dealing with component adaptation within the context of Model-Driven Web Engineering and show how design patterns can help addressing it. We first identify the major interoperability problems that can happen when integrating third-party application or legacy systems into our Web systems, and then propose the mechanisms that need to be put in place at the design level to generate the appropriate specification of adapters that compensate for the possible mismatches and differences. We base our proposal on well-known design patterns as they are established solutions to recurring problems, and the generation of code from them is normally straightforward.
European Journal of Information Systems (2007) **16,** 448–459.
doi:10.1057/palgrave.ejis.3000691

Keywords: model-driven development; Web Engineering; interoperability; adaptation; patterns

Introduction

Web applications are rapidly moving from stand-alone systems to distributed applications that need to interoperate with third-party systems, such as external Web services, LDAP repositories or legacy applications. Modeling the interactions between a Web application and external assets is something that needs to be properly addressed within any model-driven development scenario, despite not being a simple issue. In previous works, we showed how this can be done using a general model-based architectural framework for Web application modeling (WEI) (Moreno & Vallecillo, 2005a, b, c). Such a framework provides the concepts and mechanisms to facilitate the high-level integration of Web applications with third-party systems, allowing the external entities of such systems to be manipulated as native elements of our models. However, these works were based on the assumption that no interoperability problems had appeared during the integration phases.

To successfully integrate legacy or external assets into a Web application, we need to guarantee that both parts are *interoperable*, that is, there are no

Received: 1 December 2006
Revised: 12 July 2007
Accepted: 27 July 2007

potential mismatches between them. In this setting, interoperability can be defined as the ability of two or more entities to communicate and cooperate despite differences in the implementation language, the execution environment, or the model abstraction (Wegner, 1996). Interoperability enables the composition of reusable heterogeneous components and systems developed by different people, at different times, and possibly with different uses in mind.

Currently, some interoperability issues have been addressed by the adoption of common standards. Many of these have been proposed in recent years (e.g., XML, XMI, MOF, RDF, SOAP, etc.) by organizations such as W3C or OMG (Object Management Group). However, this solution does not work well in practice. Most proprietary tool vendors fail to generate fully standard-compliant specifications of the models they produce, adding particular extensions to the original one, which are unable to be understood by other systems. On the other hand, most current platforms (such as CORBA, EJB or .NET) already provide the basic infrastructure for component and service interoperability at the lower levels, that is, they sort out most of the 'plumbing' issues. However, interoperability goes far beyond that; it also involves behavioral compatibility, protocol compliance, agreements on the business rules and on the semantics, etc.

Recently, the Model-Driven Architecture (MDA) (OMG, 2001) initiative from the OMG has introduced a new approach to organize the design of an application into a set of separate models; so portability, interoperability and reusability can be achieved through architectural separation of concerns. Regarding reusability, we think that MDA can be successfully combined with design patterns (Gamma *et al.*, 1995) to facilitate the reuse of adaptation designs. Since patterns describe only the core of the solution to one problem, the implementation to carry out the responsibilities and collaborations in the pattern can take place under the guidance of MDA. In addition, the generation of code from them is normally straightforward.

Here we will focus on Web Engineering, a domain in which MDSD can be successfully applied. In fact, existing Web Engineering proposals are model-driven because they address the design and development of Web applications using separate models (navigation, presentation, content, etc.). They are also supported by model compilers that produce most of the application's logic and Web pages based on these models. However, most Web Engineering proposals do not fully exploit all the potential benefits of MDSD, such as complete platform independence or model transformation advantages, when they address some specific issues such as the integration of legacy assets into Web applications, or modeling business process adaptation. In particular, integration and adaptation are not properly addressed by current Model-Driven Web Engineering proposals (MDWE), which either ignore these problems or solve them at very low level by means of *ad hoc* implementations of component adaptors that compensate for the differences. However, they are not made explicit in the system specifications or in any of the models.

In this paper, we discuss the main interoperability issues that appear during model-based design and development of Web applications, how they can be identified and how they can be addressed using design patterns that allow modeling, at the appropriate level of abstraction, the bridges that help to mitigate the interoperability conflicts.

Instead of discussing these issues in the context of any of the MDWE proposals, we will use WEI, our model-based architectural framework for WEI, for two main reasons. Firstly, because it contains a richer and more complete set of models than any other proposal and all the other proposals can be naturally embedded into it (Moreno & Vallecillo, 2007). This framework provides a more precise separation of concerns, and better discrimination when it comes to identifying those models of the application where interoperability problems may occur. It also allows our contribution to be extrapolated to any other MDWE proposal, not only ours. Secondly, WEI already permits the integration of external services and legacy systems into the models that comprise the specification of a Web application (Moreno & Vallecillo, 2005a). This allows us to concentrate exclusively on the interoperability problems, without having to worry about the (normally *ad hoc*) mechanisms required by most of the MDWE proposals for integrating such kinds of external applications.

The structure of this document is as follows. Following the introduction, the next section briefly describes WEI and its constituent models. Then, the succeeding section presents the main contribution of this paper, the identification of a set of common conflicts and problems encountered when integrating third-party systems into a Web application design. For each problem, we analyze when it can happen, how it can be detected and how it can be solved at the design level by means of the appropriate design patterns. The penultimate section relates our work to other similar proposals and finally, the last section draws some conclusions and outlines some future lines of research.

The integration process in WEI

WEI in brief

WEI is a model-driven Web architectural framework for organizing the models that address the different concerns of a Web application development. At a high architectural design level, the whole WEI concept space is captured by means of 13 metamodels, organized in three main packages (*User Interface*, *Business Logic* and *Data*); each one corresponding to a viewpoint as shown in Figure 1. In turn, each viewpoint is composed of a set of models, which specify the entities relevant to that concern; so, each model focuses on one particular concern

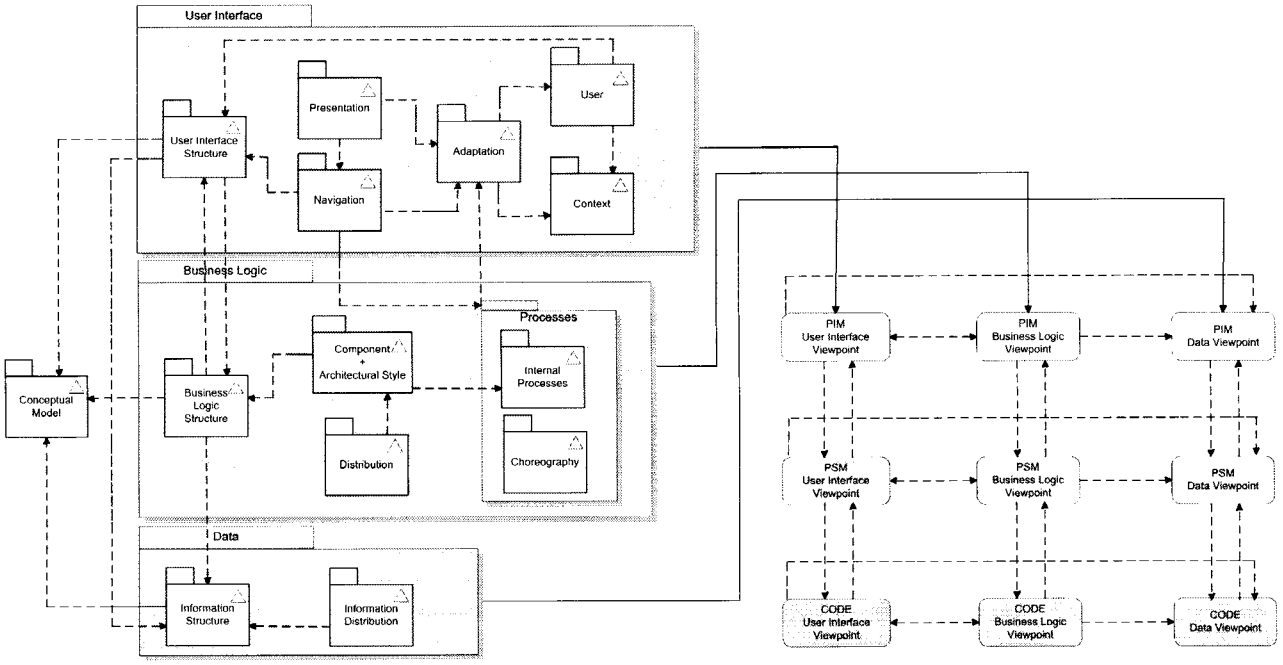

Figure 1 The WEI framework.

(navigation, presentation, architectural style, distribution) and at different levels of abstraction (platform-independent, platform-specific).

- The *Data Structure viewpoint.* It describes the organization of the information managed by the application to be stored persistently by means of, for example, a database system (the *Information Structure* model). It also describes the distribution and replication of the data being modeled (the *Information Distribution* model).
- The *User Interface viewpoint.* It focuses on the facilities provided to the end user for accessing and navigating through the information managed by the application (the *Navigation* model), and how this information is presented (the *Presentation* model) depending on device, network, location and time aspects (the *Context* model) and the user profile (the *User* model). The *User Interface* level is responsible for accepting persistent, processed or structured data from the Process and Data viewpoints (the *User Interface Structure* model), in order to interact with the end user and deliver the application contents in a suitable format.
- The *Business Logic viewpoint.* It encapsulates the business logic of the application, that is, how the information is processed in order to achieve a business objective (the *Internal Processes* model), and how the application interacts with other computerized systems (the *Choreography* model). The major classes or component types representing services in the system, their attributes, the signature of their operations and the relationships between them are described in the *Business Logic Structure* model. For a complete description of a business process, apart from the *Business Logic Structure* model, we need information related to its basic entities, which are connected by means of point-to-point connections or links. Furthermore, the *Component+Architectural Style* model defines the fundamental organization of a system in terms of its components, their relationships, and the principles guiding its design and evolution; that is, how functionality is encapsulated into business components and services.

In addition to models, the framework predefines some dependencies between the models that determine those cases in which a model definition requires the previous specification of some other models. Furthermore, these dependencies also specify correspondences between the elements from different models of the framework, especially when they may have been independently developed by different parties, or when they represent the system from different viewpoints, and therefore the same element is specified in different ways in different models (each one offering a partial view of the whole). Correspondences between model elements may also be subject to certain consistency rules, which check that the views do not impose contradictory requirements on the elements they share. In this regard, a central element of the WEI architecture is the *Conceptual* Model, which can be used both to specify the basic structure and contents in the Web application (so the rest of the views can relate to the elements of that model), and also to maintain the consistency of the model specifications establishing how

the different viewpoints merge and complement each other.

Based on previous metamodels, we have defined our own DSL associated to them as light-weight extensions of UML, that is, UML profiles that do not break the original semantics of UML metamodel, but extend it using the mechanisms provided by UML to specialize its metaclasses (stereotypes, tag definitions and constraints). The interested reader can visit the WEI Web site (*http://www.lcc.uma.es/~nathalie/WEI/*) for a complete description of the metamodels and profiles.

The WEI methodology

The integration process in WEI is supported as part of its development methodology for building Web applications. This process conforms to the MDA principles, in the sense that it is defined in terms of models and the relationships between them, so transformations can be easily formalized among the models until the final implementation is reached. As illustrated in Figure 1, WEI distinguishes three levels of abstraction (or viewpoints). Since each viewpoint is usually implemented using a different technology, WEI requires the definition of at least three PIMs to generate the implementation of a Web application: one for each layer (one PIM for the *Data* layer, one PIM for the *Business Logic* layer and one for the *User Interface* layer).

It is important to note that integration with external systems must be achieved at all required levels of abstraction. This will depend, of course, on the nature of the element to be integrated. For instance, an LDAP repository whose main functionality is storing data, requires that its integration into a Web application be carried out only at the Data level. Portlets expose functionality through a user interface component. Thus, their integration has to be considered both at the *Business Process* level and at the *User Interface* level. In contrast, Web services encapsulate functionality related to the business logic of a system, without any predefined user interface, so they will only be integrated within the *Business Logic* level.

The detailed description of the process that WEI follows for designing Web application using external systems is outside the scope of this paper, and has been reported elsewhere (we refer the interested reader to Moreno & Vallecillo, 2005a, b, c, for the description of the process and some illustrative examples). Although *a priori* there are no major problems with this approach, it is based on the assumption that no interoperability problems may occur. However, this is not always the case as we may face different kinds of incompatibility issues when trying to integrate external services or legacy applications into the system, or to interoperate with them. For instance, in the simplest case, the interface of the services required by our application (as specified in one of the PIMs) may not match the interface of the actual service, as provided by the external service provider. However other problems may also arise. The following section is dedicated to identifying them, and propose solutions at design level, that is, the solutions are incorporated into the model of the application at the highest level of abstraction possible, so they can be foreseen and their solution (if required) be part of the models of the application, and therefore be naturally represented in a platform-independent manner.

Identifying and solving integration problems at design-time

This section identifies the main problems that may happen when integrating third-party application or legacy systems into our Web designs. They are based on our experience on modeling and building Web systems that need to interoperate with external assets and applications.

Please note that our contribution is not focused on the automatic detection and correction of mismatches. Here we concentrate on their identification, and on the mechanisms that need to be put in place at the design level to generate the appropriate specification of adapters that resolve the possible mismatches and differences. We base our proposal on well-known design patterns, as they are established solutions to recurring problems and the generation of code from them is usually straightforward.

Syntactic conflicts

We cannot assume that software developed by independent parties will be assigned the same names. A syntactic conflict occurs when architectural components use different data structures and representations for identical concepts. The concept in question may be an object type, a process, a property, a relationship, or an instance of any of these. The conflict could be as simple as a disagreement on the spelling of a name (naming of methods, parameter, types or exceptions), or as complex as a different data organization and a different approach to data representation (typing of methods or parameters). Besides, when the interaction is more complex than one-way communications, the matching conversion for the reverse direction must also be constructed.

Where these problems appear. We have found this type of problem at the three levels of abstractions established by WEI. For example, between pairs of *Presentation* models at the *User Interface* level when we have to integrate external interfaces of portlets; between pairs of *Business Information Structure* models at the *Business Logic* viewpoint when we have to integrate WSDL Web services descriptions, etc.

How to detect them. At first sight, the designer can detect these conflicts when he compares the required with the provided specification. Other automatic strategies based on signature matching can also be applied here (Zaremski & Wing, 1995, 1997), although their tool support is currently quite limited. Model subtyping and type-inference are recent approaches that can also be of great help, because they allow checking that a given model is a subtype of another, that is, one can safely replace the other (Steel & Jézéquel, 2005; Romero *et al.*, 2007).

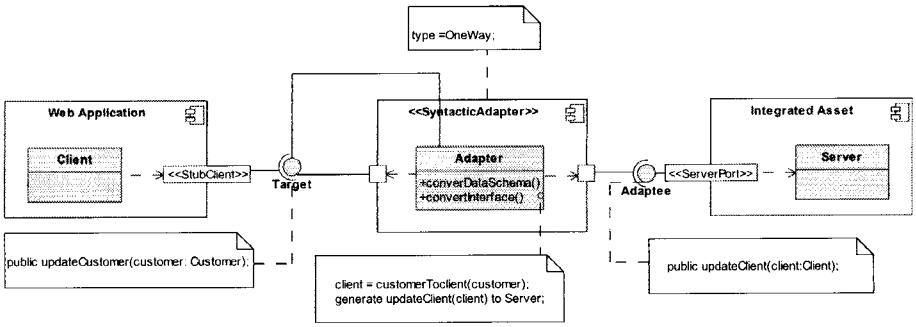

Figure 2 Solving syntactic conflicts with the adapter pattern.

How to address them. In general, what is required is an intermediary component that translates between the two representations: it converts the source data schema or interface (and its data) to an equivalent one in the language of the target model. Such a component need to be specific for the set of interfaces involved but can be constructed taking advantage of the adapter pattern. As shown in Figure 2, from the general case the pattern has to be customized to fit our needs. Its main participants are: (i) the *Target interface*, which is the interface required by a client component; (ii) the *Client*, which is a component whose interface is compatible with the *Target interface*; (iii) the *Syntactic Adapter*, which is the component responsible for making an existing interface or data schema compatible with the *Target interface* and (iv) the *Adaptee interface*, which is the *Server interface*. The code implementing the translation can also be specified at design-time using any action semantic language, such as Xion (Muller *et al.*, 2005) or OAL (BridgePoint Object Action Language) (Mellor & Balcer, 2002).

Example. Let us consider an LDAP directory that defines an entry with one object type named 'customer', while its counterpart in the Web application database is called 'client'. In this case, the adapter pattern provides a simple and elegant solution (see Figure 2). The actions shown for the adapter object have been defined using the OAL notation.

As the reader can notice, the interactions between cooperating parts are modeled in WEI in terms of *ports* that encapsulate the programming abstractions through which the client and service provider perceive and use the communication. WEI requires that certain implementation choices about the selected target platform, which are usually made implicit and therefore assumed and hard-coded in the application, are made explicit in the appropriate model(s). Thus, a ≪StubClient≫ identifies a component that should obtain a reference to the Stub before using it, which represents an instance of the server provider. In this way, both the remote interface and its implementation have to be available, so the client relies on an implementation-specific class. On the other hand, the ≪ServerPort≫ plays the other endpoint of the Web interaction. It encapsulates the programming abstractions through which the server carries out the communications with the client.

Underlying platform or communication protocol mismatches

These problems are common in Web applications that use several platform technologies, simultaneously. This kind of problem appears when the two systems disagree on the technical specifications for performing the communication, or even when they use the same mechanism but the specific protocols they use may be different.

Where these problems appear. They affect both the *Business Information Structure* and the *Choreography* models.

How to detect them. They can usually be detected by comparing the underlying platform or communication protocols used by the client and server components. Details about implementation protocols are also explicitly modeled in WEI, as tag definitions of stereotypes ≪ServerPort≫, ≪StubClient≫, ≪DynamicClient≫, etc., which allows this kind of conflict to be detected.

How to address them. Generally, the adapter pattern allows solving this kind of mismatch in a natural way. In other cases, we may require more sophisticated strategies such as those provided by the mediator pattern, by wrappers, or by combinations of both. Typically, a wrapper acts as a server object and provides a standard interface through which mediators can access heterogeneous components.

Example. Let us consider a document conversion Web application that translates between different file formats like PostScript, PDF, plain text, etc., using external services for this purpose. Suppose that the application uses the HTTP protocol to communicate with these external services, but wants to make use of a new external service that is available only as a CORBA component. In order to solve this mismatch, Figure 3 shows a mediator–wrapper pattern to transform data from the client component to the server component in the format it expects. The participants in the Mediator–Wrapper pattern are: (i) the *data-driven protocol adapter* component, which defines the interface required by a client component; (ii) the *mediator*, which is a component that coordinates wrapper objects; (iii) the *wrappers*, which

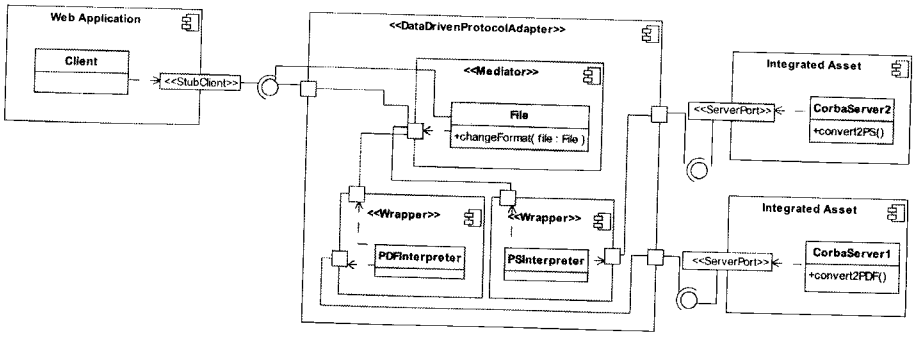

Figure 3 Solving platform mismatches with the mediator–wrapper pattern.

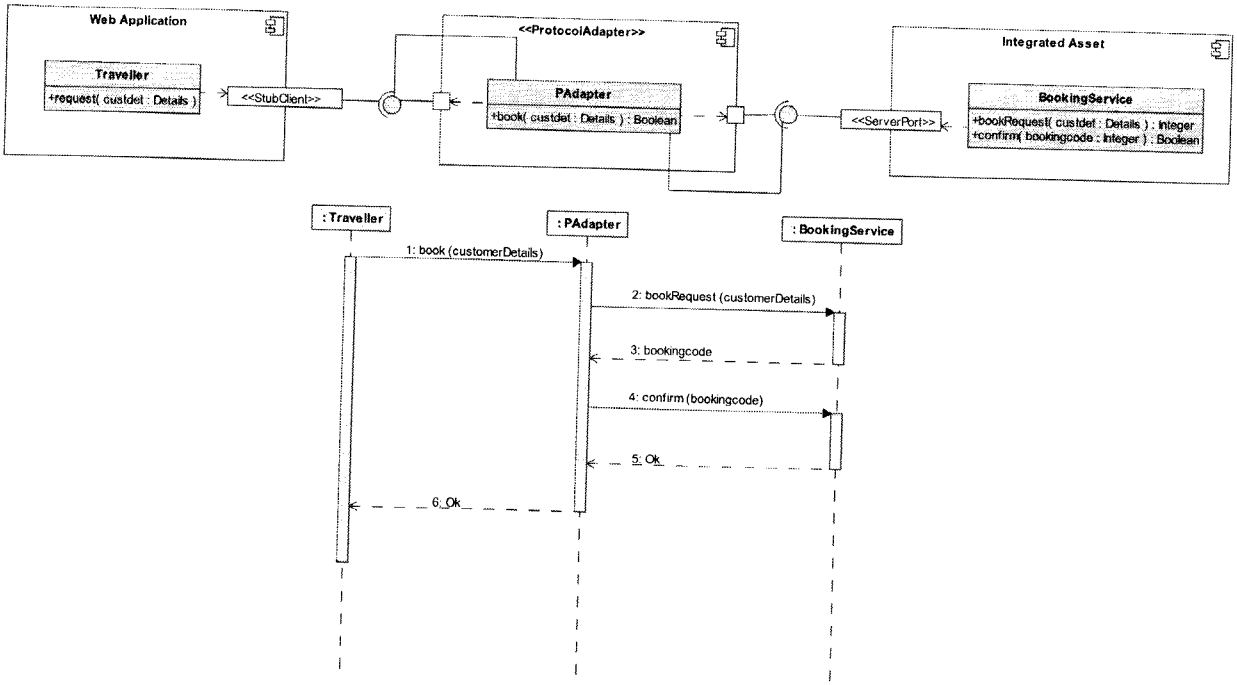

Figure 4 Solving protocol problems with the mediator–wrapper pattern.

are the components responsible for making the transformation and (iv) the *integrated assets*, which provide the corresponding server interfaces and implement the services.

Choreography mismatches

This problem refers to mismatches in the relative order in which an object expects its methods to be called, the order in which it invokes other objects' methods, and the blocking conditions and rules that govern the interactions. In this way, if a BPEL4WS specification assumes that a component behaves in a certain way, but the actual implementation of the component (which is not stated anywhere in the WSDL specifications of the component) does not behave like that, a choreography mismatch occurs.

Where these problems appear. They mainly happen in the *Choreography* model.

How to detect them. There are several proposals that address interoperability problems at this level (also called the 'protocol' level), assuming that the components have a precise specification of their choreography using some formal description techniques: based on logical and temporal rules (Lea & Marlowe, 1995); using UML's OCL pre- and post conditions on the object's method together with a simple finite state machine (Cho *et al.*, 1998); using process algebras (Bastide *et al.*, 1999; Canal *et al.*, 2003), etc. On the other hand, we can also rely on model checkers based on Promela or SDL to check this kind of conflict.

How to address them. The solution should be given depending on the extension of the conflict. In those cases in which the only changes required to enable the interoperability between the parts imply only modifications in the communication choreography, but without altering the content of the messages (i.e., only

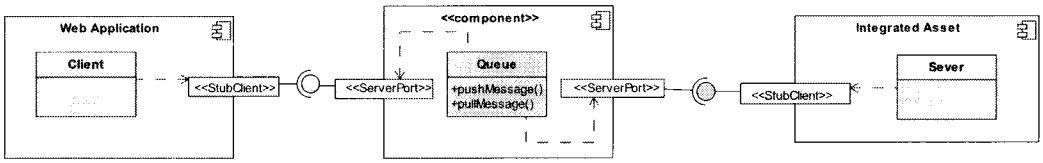

Figure 5 Solving role problems with the adapter pattern.

changes in the connectors), the adapter pattern works well. In other cases, the conflict may require combinations of sets of patterns, or it may even be impossible to solve. Some works are currently investigating the automatic generation of mediators from the formal specification of the incompatible components, under certain circumstances (Bracciali *et al.*, 2005). This is an interesting line of future research, especially in the context of the Web 2.0, where automatic mediation and adaptation seems to be required.

Example. Let us consider a Web portal implementing a booking service functionality in an airline ticketing system. The *AirlineTicketing* system is composed of three Web services: a *BookingService*, and two hypothetical *Traveller* and *Airline* components. Let us consider the UML class and sequence diagrams shown in Figure 4 specifying the provided booking service protocol and the required protocol corresponding to a potential client of the service, that is, the *Traveller*. In this simple example, an adaptor can solve the protocol mismatch capturing all the exchanged messages between cooperating parts and discarding the results of undesired messages.

Incompatibility of roles

Role conflicts occur when communicating components present incompatible assumptions about their functional roles in their interactions (e.g., both want to play the client role or, alternatively, both want to be in control). This kind of problem also happens in other environments, for example, when composing object-oriented application frameworks (Mattsson & Bosch, 1997).

Where these problems appear. They can be considered a kind of protocol problem, in which there is no agreement on who is providing what information to whom, and who is performing which function. This kind of problem affects mainly the *Choreography* and the *Component* models.

How to detect them. They can be detected comparing the 'required' component model with the 'provided' component model: each client port should have a server port satisfying its requirements. Consequently, proposals that address interoperability problems at the protocol level are also suitable here (e.g., Lea & Marlowe, 1995; Cho *et al.*, 1998; Canal *et al.*, 2003).

How to address them. What is required for solving this type of incompatibility problem is an intermediary component that plays the missing role with respect to the parts being connected. WEI solves this kind of problem applying the adapter pattern. The participants are similar to those described in 'Syntactic conflicts' section; but in this case, the adapter component acts only as a channel receiving messages and delivering them without any kind of transformation.

Example. Let us consider the example proposed in Figure 5. We can observe that both components expect to behave as 'clients', and neither expects to be the 'server' (or vice versa). In this case, the adapter component must be both a push server who accepts and queues messages from the push client, and a pull server who delivers them to the pull client on request.

Data format problems

Although the meaning or purpose of the exchanged data may be similar or identical, there may be differences in the data formats required by the various applications and platforms.

Where these problems appear. Although this conflict may appear at the three levels of abstraction established by WEI, it normally happens between pairs of *Information Structure*, *Business Logic Structure* and *User Interface Structure* models.

How to detect them. The criteria for determining the existence of data format conflicts can be assisted by subtyping techniques (Simons, 2002; McKegney & Shepard, 2003; Steel & Jézéquel, 2005; Romero *et al.*, 2007). When it is not possible to derive a relationship between two types, a data format conflict probably exists.

How to address them. The solution is based on the wrapper pattern. It translates the data considering the characteristics of the integrated data sources. In these cases, it is preferable to let a mediation component handle the transformation from one format to another. It may even be possible to define a canonical data model, although it may force service consumers to deal with it (or use other wrappers).

Example. The example illustrated in Figure 3 describes also a data format problem that can be addressed by the mediator–wrapper pattern. The proposed architecture provides a model-driven solution that allows transparent access to heterogeneous data formats.

Data consistency problems

These kind of problems arises when the two systems maintain their own copy of the information that they share, but these copies are inconsistent. There are many causes that may lead to this situation, especially when both systems update their data individually.

Where these problems appear. They are mainly present in the *Conceptual*, *Information Structure*, *User Interface Structure* and *Business Logic Structure models.*

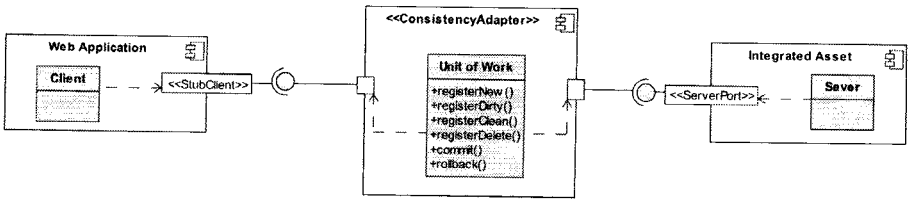

Figure 6 Solving consistency data problems with the unit of work pattern.

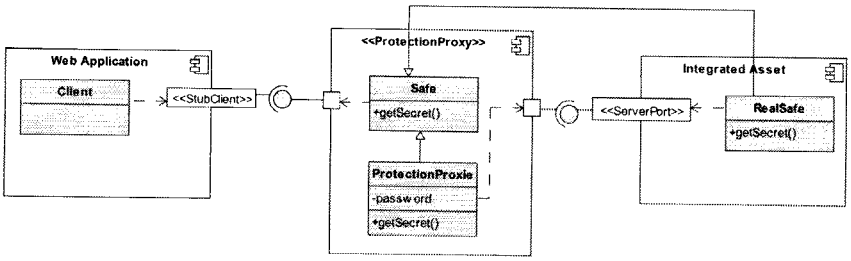

Figure 7 Solving QoS problems with the protection proxy pattern.

How to detect them. They are normally problems that happen at run-time. However, the initial design can be prepared to avoid them by using the appropriate synchronization mechanisms. Alternatively, if behavioral models are used, it is possible to detect some of them or at least some critical regions.

How to address them. Unlike most other conflicts, solutions to data consistency conflicts are particularly difficult to automate because much detailed analysis is required to determine exactly what the problem is, and the best way to solve it. In many cases, the solution consists of avoiding the problem in the first place, by synchronizing the processes by means of transactions. The unit of work pattern can be used here to keep track of everything that happens during a business transaction, which may affect the databases or the data sources that need to be kept consistent. When a transaction is completed, it figures out what needs to be done to alter the database as a result of the session, and propagates the changes to the other data sources. Alternatively, monitors could be included in the design to deal with the management of potential critical regions, also using the unit of work pattern.

Example. Let us consider a component that maintains a private file or a database with information about user logins and passwords that is not being regularly updated from the 'primary database' held in the Web application. This example may present a data consistency problem, solved as shown in Figure 6.

Quality-of-service and other service-level agreement problems

A quality-of-service (QoS) conflict occurs when the behavior of a component does not satisfy the QoS requirements imposed by the system specifications. Such requirements deal with security, timing, redundancy and similar aspects of the application.

Where these problems appear. WEI mainly models the QoS requirements in the *Component* model as properties of the interaction ports.

How to detect them. These are usually problems that happen at run-time, and need to be detected by special processes that monitor the behavior of the integrated components (especially when they are external assets). Thus, the initial design has to incorporate such kind of monitor processes. Furthermore, there are cases in which we might also want to consider the use of adaptors that allow solving the disagreements between the system requirements and the integrated components' behavior.

How to address them. There is extensive literature available on QoS monitoring and resource adaptation and allocation to meet Service-Level Agreements, mostly based on the use of the wrapper or proxy patterns.

Example. Let us consider a component that expects to use a secure communication mechanism for certain kinds of transactions, while its counterpart in the communication does not work in that way. In this case, the proxy pattern is frequently applied to implement some of the required access control mechanisms. Protection proxies are also useful when objects should have different access rights. Figure 7 shows how to apply this pattern if we have an object storing a protected information.

The main participants in this pattern are: (i) the *client* that wants to get a protected information; (2) the *protection proxy* that asks the user to authenticate itself – if the user gives the correct information to the proxy, then it calls the object and passes the protected information to the client and (3) the *RealSafe* object that represents the protected object.

Semantic mismatches

The semantic requirement for integration tries to ensure that the exchange of services and data among communicating parties make sense, that is, that requesters and

providers agree on the 'meaning' of objects, properties and processes that they share within the Web application.

Where these problems appear. Semantic conflicts result from mismatches between communicating components in the *Conceptual*, *Information Structure*, *User Interface Structure* and *Business Logic Structure* models. They may also appear in the activities in which they must interact, or in the interpretation of references to object types and instances in such activities.

How to detect them. Some proposals try to endow component interfaces with semantic (behavioral) information. In these cases, we can apply behavioral subtyping algorithms or techniques for detecting these mismatches, see for example Zaremski & Wing (1997). Other proposals use ontologies and semantic matchmaking for detecting these problems.

How to address them. Solutions to semantic problems are generally formulated in terms of ontologies that expose similarity relations among concepts of different schemas. We can simulate this approach by using the OCL constraints and QVT relations present in the WEI *Conceptual* model. Another solution to solve some basic semantic conflicts uses semantic adaptors that implement translations between the original concepts and the target elements by, for example, applying the adapter pattern as illustrated in the resolution of syntactic conflicts.

Example. Let us consider a Web application that integrates two data sources, whose data schema definition uses the same term (e.g., 'Tiger') to refer to two different things (e.g., an animal and a car). That is, they assign different meanings to the same term. A simple way of solving this problem is by using renaming techniques. More precisely, in WEI it is easy to define QVT relations that allow the renaming of a term, and the automatic propagation of the changes to the rest of the models (to maintain consistency).

```
relation Tiger2Tiger{ /*solving mismatch Tiger to Tiger*/
   checkonly domain webml.WebML_Metamodel e1:WebMLEntity{
      name = 'Tiger',
      attribute = a1:WebMLAttribute{name = a1n,
                                    type = p1:WebMLDataType{name = p1n}}};
   checkonly domain ooh.OO-H_Metamodel e2:OOHEntity{
      name = 'Tiger',
      attribute = a2:OOHAttribute{name = a2n,
                                  type = p2:OOHDataType{name = p2n}}};
   where{
      rename(e2);}
}
```

Functional conflicts

Functional conflicts arise when the behavior of one communicating resource differs from the behavior expected by its counterpart in the interaction. A functional conflict may both modify the subsequent interactions of functions of each part, and may also affect the successful performance of the joint action.

Where these problems appear. These conflicts appear in the *Internal Processes* and *Choreography* models, which describe how the business functionality is decomposed into small pieces of required behavior for the individual resources comprising the system.

How to detect them. Automatic strategies based on protocol and behavioral mismatches can be applied here (Lea & Marlowe, 1995; Cho *et al.*, 1998; Canal *et al.*, 2003).

How to address them. When the provided functionality is larger than the required one, it may be possible to construct an interceptor that captures requests and decomposes it into multiple requests to be executed. But, in general, the problem may not allow a common solution.

Example. Let us consider a *UserTrace* Web system, which interprets that a *deleteUserOnly()* operation means only deleting the row representing the information about the user currently connected, while a cooperating external system interprets that this operation implies deleting not only the information about the user but also all the other information relating to him. This may represent serious interoperability problems, which are very difficult to detect and to solve in practice. As shown in Figure 8, from the general case the adapter pattern has to be customized to fit our needs.

Related work

Closer proposals to the contribution we have presented in this paper fall into two main categories: ones that follow a model-driven approach for coping with a subset of the problems identified here, either in the Web context or in other domains, and others that follow a code-centric or implementation-driven approach, applying or not design patterns, for solving particular conflicts.

With regard to the first group of works, several authors have provided a number of interesting proposals regarding *Enterprise Application Integration* (EAI) systems (Linthicum, 2000). These systems address three major integration goals: data or information integration, process integration, underlying platform independent and common facade. To solve the integration conflicts

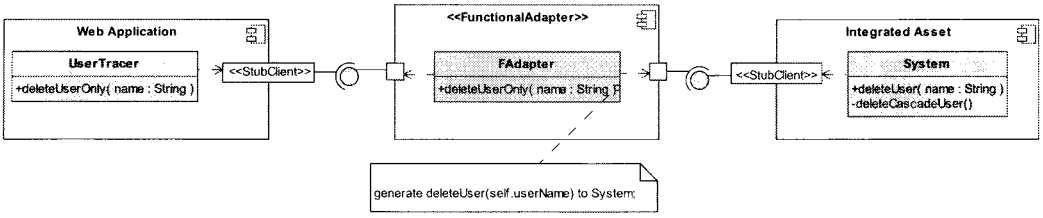

Figure 8 Solving functional conflicts with the adapter pattern.

presented, they usually implement the mediation and federation patterns. However, the integration is achieved by defining an interaction conceptual model that particularizes an existing conceptual model provided by the EAI vendor (linked in such a way to the models of a particular system). Often this conceptual model is not Web –oriented; so many of the modeling, integration and adaptation requirements of Web applications that we have identified here are not supported by EAI vendors. In addition, most EAI solutions are provided to deal with these issues at a very low level, that is, they are implementation dependent. Probably, one of the major benefits of our proposal is that it is platform- and technology independent, according to the MDA principles. This allows designs to be reused across many different platforms, and permits organizations to count of long-lived designs of their systems, which are not tied to particular technologies. And then they can be linked to the particular implementations provided by some EAI vendors, as we shall later see.

In the particular context of Web Engineering, model-driven adaptation has been neglected by most Web methodologies. In fact, modeling the integration of third-party systems into a Web application design is not yet fully supported by most MDWE approaches. And those methodologies that allow reusing external components at design level (such as WebML (Brambilla *et al.*, 2003), UML-Guide (Dolog, 2004) or MIDAS (De Castro *et al.*, 2006)) are normally based on the assumption that no interoperability problems will occur. Thus, any incompatibility conflict that may appear during the integration phase is usually hard-coded at the implementation level, and not documented by any of the Web methodology views, as previously mentioned. Consequently, the ideal objectives and advantages derived from applying model-driven development (such as platform, technology and implementation independence) are lost.

More mature results solving specific adaptation requirements can be found in the Component-based Software Engineering and data-based fields. In Schmidt & Reussner (2002), the authors address some protocol interoperability conflicts by deriving adapters for merging and splitting interface protocols. In Spitznagel & Garlan (2003), Garlan *et al.* tackle the problem of solving component interactions conflicts by using wrappers that alter the behavior of a component without modifying the component or the infrastructure itself. Yellin & Strom (1997) also develop adapters for software components that have compatible functionality but present syntactic or protocol mismatches assuming that interface mappings are provided. On the other hand, Thiran & Hainaut (2001) propose a technology for the automated production of hard-coded wrappers for data systems. They show that the code of a wrapper can be dedicated to solving syntactic and semantic mappings, and that these mappings can be modeled through semantics-preserving transformations. Finally, Reck & Koenig-Ries (1997) support the quick development of mediators by automatically deriving their specifications – which are subsequently fed to a mediator generator.

These efforts can be considered mechanisms that can be leveraged for Web application adaptation, but are not sufficient because in this context richer description models than component interfaces and protocols or database schemas are required. This is due to the fact that clients and services are typically developed by separate teams, possibly even by different companies, and service descriptions are all that client developers have to understand to know how the service behaves. Thus, the work presented here can be seen as complementary to these other works. Here we have focused on a platform-independent specification of such adaptors and wrappers, while the previously mentioned works in this section can provide the implementation details of our designs for specific platforms. More precisely, our work provides a bridge between the platform-independent models of a Web application (which are now able to capture the interoperability aspects at the design level), and the current solutions for EAI integration and component adaptation, which allow solving most of these problems – but usually at a platform-specific implementation level.

Conclusions and future directions

This paper identifies the main conflicts encountered in solving integration problems during Model-Driven Web applications design and development. A set of well-known design patterns are proposed to tackle these problems at design level, addressing them in the different models that constitute the specification of a Web system. In this way, this work tries to improve the design, documentation and maintenance of existing Web systems by filling the current gap between the design of a Web application and its implementation – achieved in most cases as a set of interacting parts where some of them are external services.

On the basis of this design–time adaptation, our proposal can be implemented applying transformation rules that determine how to derive the code from the pattern models for a target platform technology. In doing so, not only the structure but also the behavior of patterns need to be described, as we have briefly illustrated in some of the examples shown in 'Identifying and solving integration problems at design–time' section. In that sense, there is a lack of a standard syntax for specifying actions in UML. Instead, there is a variety of action languages such as the OAL (Mellor & Balcer, 2002), the Kennedy Carter Ltd. Action Specification Language (ASL) (Raistrick *et al.*, 2006), the Shlaer-Mellor Action Language (SMALL) (Shlaer & Mellor, 1992), the Xion language (Muller *et al.*, 2005), etc. Although most of the existing action languages and tools are not Web oriented, we have successfully used them to model many of the behavioral parts of the adaptors, and we have obtained good implementations, because these tools generally give explicit support for statecharts diagrams (generating Web

services or software components at the implementation level).

Our future research is directed toward developing a Web development environment capable of supporting the automatic suggestion of pattern(s), which can be applied to solve the identified integration conflicts. This tool might integrate some of the proposals and engines that we need to evaluate the distance between the models of the 'required' and the 'actual' services and also make use of proposed patterns to create the links and adapters needed to connect two or more available resources.

Finally, we want to point out that although the proposal has been addressed from the WEI viewpoint, there is nothing to prevent us from extrapolating the results to other Web Engineering methodologies. The analysis conducted here and the proposed solutions could be very useful to other model-based Web Engineering methods if they decide to extend their proposals to address adaptation requirements following a model-driven approach. Since each integration conflict has been localized in terms of the WEI models that could present that specific problem, and there is a compatibility between models and design elements coming from different methodologies and the appropriate ones of our framework, this goal can easily be achieved. For example, in the WebML context every WEI adapter would be modeled as a parameterizable operation unit being able to interact with other content and operation units in the hypertext model. In the same way, WEI assembly connectors would have their counterpart in the definition of appropriated WebML transport links allowing both the parameters passing and the specification of correspondences between required and provided requirements.

Acknowledgements

The authors like to thank the anonymous referees for their insightful and constructive comments and suggestions. Although the views in this paper are the authors' sole responsibility, they could not have been formulated without many hours of detailed discussions with MDWE experts. In particular, we like to thank Nora Koch and Piero Fraternali for sharing their expertise and knowledge with us. This work has been supported by Spanish Research Project TIN2005-09405-02-01.

About the authors

Nathalie Moreno Vergara is Assistant Professor at the Department of Computer Science of the University of Málaga where she received the M.Sc. degree in Computer Science. Her current research interest is oriented toward model-driven development of Web applications, conceptual modeling methodologies, model transformation languages and code-generation techniques in the Web context.

José M. Troya Linero is Full Professor at the Department of LCC of the University of Málaga. He received his M.Sc. and Ph.D. degrees in Computer Science from the University Complutense of Madrid in 1975 and 1980, respectively. His research interests include parallel algorithms for optimization problems and software engineering for distributed systems.

Antonio Vallecillo Moreno is Associate Professor at the Department of Computer Science of the University of Málaga where he holds the Ph.D. degree in Computer Science and the B.Sc. and M.Sc. degrees in mathematics. His research interests include model-driven software development, open distributed processing and the industrial use of formal methods.

References

BASTIDE R, SY O and PALANQUE P (1999) Formal specification and prototyping of CORBA systems. In *Proceedings of the 13th European Conference on Object-Oriented Programming ECOOP'99*. Lecture Notes in Computer Science No. 1628. Lisbon, Portugal Springer-Verlag, Heildelberg.

BRACCIALI A, BROGI A and CANAL C (2005) A formal approach to component adaptation. *Journal of Systems and Software, Special Issue on Automated Component-Based Software Engineering* **74**, 45–54.

BRAMBILLA M, CERI S, COMAI S, FRATERNALI P and MANOLESCU I (2003) Model-driven development of Web services and hypertext applications. In *Proceedings of the Systemics, Cybernetics and Informatics Multiconference (SCI'03)*. Orlando, Florida.

CANAL C, FUENTES L, PIMENTEL E, TROYA JM and VALLECILLO A (2003) Adding roles to CORBA objects. *IEEE Transactions on Software Engineering* **29(3)**, 242–260.

CHO IH, MCGREGOR JD and KRAUSE L (1998) A protocol based approach to specifying interoperability between objects, In *Proceedings of the 26th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'98)*. IEEE Computer Society, Washington, USA.

DE CASTRO V, MARCOS E and LÓPEZSANZ M (2006) A model driven method for service composition modelling: a case study. *International Journal of Web Engineering and Technology* **2(4)**, 335–353.

DOLOG P (2004) Model-driven navigation design for semantic Web applications with the UML-guide. In *Engineering Advanced Web Applications: Proceedings of Workshops in connection with the Fourth International Conference on Web Engineering (ICWE 2004)*. Rinton Press, Munich, Germany.

GAMMA E, HELM R, JOHNSON R and VLISSIDES J (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts.

LEA D and MARLOWE J (1995) Interface – based protocol specification of open systems using PSL. In *Proceedings of the Ninth European Conference on Object-Oriented Programming (ECOOP'95)*. Lecture Notes in Computer Science No. 952 Springer-Verlag, London, UK.

LINTHICUM DS (2000) *Enterprise Application Integration*. Addison-Wesley Longman Ltd., Essex, UK.

MATTSSON M and BOSCH J (1997) Framework composition: problems, causes and solutions. In *Proceedings of the TOOLS-23: Technology of Object-Oriented Languages and Systems (TOOLS '97)*. IEEE Computer Society, Santa Barbara, CA.

MCKEGNEY R and SHEPARD T (2003) Techniques for embedding executable specifications in software component interfaces. In *Proceedings of the Second International Conference on COTS-Based Software Systems (ICCBSS'03)*. Lecture Notes in Computer Science No. 2580 Springer-Verlag, London, UK.

MELLOR SJ and BALCER M (2002) *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

MORENO N and VALLECILLO A (2005a) A modelbased approach for integrating third party systems with Web applications. In *Proceedings of the Fifth International Conference on Web Engineering (ICWE'05)*. Lecture Notes in Computer Science No. 3579 Springer-Verlag, Sydney, Australia.

MORENO N and VALLECILLO A (2005b) Incorporating cooperative portlets in Web application development. In *Proceedings of the First Workshop on Model-driven Web Engineering (MDWE 2005)*. Sydney, Australia.

MORENO N and VALLECILLO A (2005c) Modeling interactions between Web applications and third party systems. In *Proceedings of the Fifth International Workshop on Web Oriented Software Technologies (IWWOST'05)*. Porto, Portugal.

MORENO N and VALLECILLO A (2007) Towards Interoperable Web Engineering Methods. Submitted for publication.

MULLER PA, STUDER P, FONDEMENT F and BÉZIVIN J (2005) Platform independent Web application modeling and development with Netsilon. *Software and System Modeling* **4(4)**, 424–442.

OMG (2001) *Model Driven Architecture. A Technical Perspective*. Object Management Group. OMG document ab/2001-01-01.

RAISTRICK C, FRANCIS P, WRIGHT J, CARTER C and WILKIE I (2006) *Model Driven Architecture with eXecutable UML*. Cambridge University Press, New York, USA.

RECK C and KOENIG-RIES B (1997) An architecture for transparent access to semantically heterogeneous information sources. In *Proceedings of the First International Workshop on Cooperative Information Agents*, Kiel, Germany.

ROMERO JR, RIVERA JE, DURÁN F and VALLECILLO A (2007) Formal and tool support for model driven engineering with Maude. *Journal of Object Technology (JOT)* **6(9)**, http://www.jot.fm/.

SCHMIDT HW and REUSSNER RH (2002) Generating adapters for concurrent component protocol synchronisation, In *Proceedings of the IFIP TC6/WG6.1, Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems V (FMOODS'02)*. Kluwer, B.V., Deventer, The Netherlands.

SHLAER S and MELLOR SJ (1992) *Object Lifecycles: Modelling the World in States*. Yourdon Press, Englewood Cliffs, New Jersey, USA.

SIMONS AJH (2002) The theory of classification, object types and subtyping. *Journal of Object Technology* **1(5)**, 27–35.

SPITZNAGEL B and GARLAN D (2003) A compositional formalization of connector wrappers. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*. IEEE Computer Society, Washington, USA.

STEEL J and JÉZÉQUEL JM (2005) Model typing for improving reuse in model-driven engineering. In *Proceedings of the 8th International Conference, MoDELS 2005*. Lecture Notes in Computer Science No. 3713 (BRIAND L and WILLIAMS C, Eds) Springer-Verlag, Montego Bay, Jamaica.

THIRAN P and HAINAUT JL (2001) Wrapper development for legacy data reuse. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*. IEEE Computer Society, Washington, USA.

WEGNER P (1996) Interoperability. *ACM Computer Survey* **28(1)**, 285–287.

YELLIN D and STROM R (1997) Protocol specification and component adaptor. *ACM Transactions on Programming Languages and Systems* **19(2)**, 292–333.

ZAREMSKI AM and WING JM (1995) Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology* **4(2)**, 146–170.

ZAREMSKI AM and WING JM (1997) Specification matching of software components. *ACM Transactions on Software Engineering and Methodology* **6(4)**, 333–369.