The WebSA Configuration Model Profile

Santiago Meliá

Web Engineering Research Group,
Dept. of Languages and Information Systems.
Universidad de Alicante
santi@dlsi.ua.es

1 WebSA Profile

The WebSA approach is completed by the standardization which is performed, as other Web methodologies [11] have done before, by the definition of a UML Profile. The WebSA Profile comprises a profile for each model: (1) SM Profile, (2) CM Profile and (3) IM Profile. Fig. 1 depicts these profiles and the dependency between the CM and IM packages and it also shows how the WebSA profile has been applied to a web application example like Petstore.

The WebSA Profile focuses on:

- Providing modeling elements for a particular platform or domain. The WebSA domain is the set of components of the family of Web applications.
- Adding information that can be used when transforming one model to another model or to code.

In this technical report we will only focus on the Configuration Model Profile of the WebSA architectural viewpoint.
1 Configuration Model Profile

The Configuration model defines an architectural style based on the structural view of the Web application by means of a set of Web components and their connectors, where each component represents the role or the task performed by one or more common components identified in the family of Web applications. In this way, CM uses a topology of components defined in the Web application domain, and this allows us to specify the architectural configuration without knowing anything about the problem domain. At this level, we can also define architectural patterns for the Web application as a reuse mechanism.

A Configuration model is built by means of a UML 2.0 Profile of the new composite structure model, which is well-suited to specify the software architecture of applications.

The main aim of a profile is to give a simple mechanism for adapting a metamodel with constructs that are specific to a particular domain, platform, or method. Based on a metamodel like the Meta Object Facility (MOF) [8] or usually UML itself, a profile specifies new model elements called stereotypes. A stereotype customizes the UML metamodel introducing a new terminology, new syntax, new semantics and constraints, or adding further information (attributes or tag values).

In this way, the WebSA Profile has incorporated all the elements of WebSA metamodel as stereotypes, extending each one of the UML metaclasses. To do that, we match the UML 2.0 [9] metamodel classes in order to adapt them to the semantic of the WebSA classes and to allow them to define the metamodel relationships. To specify how a stereotype extends a metaclass in a profile, UML 2.0 has incorporated the relation named “extension”. An extension is a binary relation, that is, a stereotype is dependent on exactly one element of the underlying metamodel. It is depicted by an arrow with filled arrowhead (see Fig. 1). Also, the extension can be marked as \{required\}, when the stereotype is always created if an instance of the extended class is created (p.e. a WebComponent is created in place of a Class).

We have included the part of the WebSA profile that refers to the most important elements, that is, those contained in the Core package. The rest of the metamodel elements can be defined with relationships of inheritance from the stereotypes of the Core package.

Next, we will show the WebSA stereotypes and the metamodel classes of UML 2.0 which we have extended:
Once we have indicated graphically the defined stereotypes, we are going to see each one of them in detail, for which we will use a template that describes stereotypes thoroughly. It will be formed by the following sections: name, a brief description, the attributes (tag values) it incorporates, its constraints, the semantic of the concept, and finally a notation of each one of the defined stereotypes.

In order to able to go through the relationship extension of the UML 2.0 profile mechanism, the OCL operation `isStereotyped` is defined as follows:

```
isStereotyped (stereotypeName:String) : Boolean;
self.extension-> exists (x | x.ownedEnd.type = stereotypeName)
```

<table>
<thead>
<tr>
<th>Name</th>
<th>WebComponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>WebComponent extends the metaclass “CompositeStructures::StructuredClasses::Class” (see fig. 1).</td>
</tr>
<tr>
<td>Attributes</td>
<td>No additional attributes.</td>
</tr>
<tr>
<td>Constraints</td>
<td>context CompositeStructures::StructuredClasses::Class</td>
</tr>
<tr>
<td></td>
<td>inv: self.isStereotyped(“WebComponent”) implies</td>
</tr>
</tbody>
</table>
-- A WebComponent has no features (attributes and operations).
  self.attributes->isEmpty() and self.ownedBehavior->isEmpty()
--The Interfaces of a WebComponent must be instances of WebInterface
  and self.implementation->forAll(i| i.contract.oclIsTypeOf (WebInterface))
-- All connectors are WebConnectors
  and self.ownedConnector->forAll (c| c.oclIsTypeOf(WebConnector))
-- All its ports must be instances of WebPort
  and self.ownedPort->forAll (p | p.oclIsTypeOf(WebPort))
-- All its ports must be instances of WebPort
  and self.ownedPort->forAll (p | p.oclIsTypeOf(WebPort))

Semantics
A WebComponent in the Configuration Model represents an abstraction of one or
more software components with a shared functionality or role in the context of a
Web application.

Notation
WebComponent keeps the notation of UML structure class (see fig. 2.a).

Name: WebConnector

Description
WebConnector extends the UML metaclass “CompositeStructures::InternalStructures::Connector” (see fig. 1).

Attributes
No additional attributes.

Constraints
context CompositeStructures::InternalStructures::Connector
-- A Concrete Connector is related by means of WebConnectorEnds.
inv: self.isStereotyped ("WebConnector") implies
  self.end-> forAll (e | e.isStereotyped ("WebConnectorEnd"))

Semantics
WebConnector specifies a link that allows the communication in the system be-
tween two or more WebComponents or/and WebParts of the WebComponents.

Notation
WebConnector establishes the communication by means of WebConnectorEnds.
This connector is represented with the notation of a UML association (see fig. 2.b).

Name: WebConnectorEnd

Description
A WebConnectorEnd extends the UML metaclass “CompositeStructure::InternalStructures::ConnectorEnd” (see fig. 3) .

Attributes
The WebConnectorEnd has two properties: (1) lower which specifies the lower
bound of elements which could be connected with the WebConnectorEnd. (2) upper which specifies the upper bound of elements which could be connected with the WebConnectorEnd.

Constraints
context CompositeStructure::InternalStructures::ConnectorEnd
-- A WebConnectorEnd is related to a WebPort or a WebPart.
inv: self.isStereotyped("WebConnectorEnd") implies
    self.role.isStereotyped("WebPort") or self.partWithPort.isStereotyped("WebPart")

**Semantics**

WebConnectorEnd represents an endpoint of the connector that attaches the connector to a WebPort or a WebPart.

**Notation**

Adornments may be shown on the WebConnectorEnd corresponding to adornments on connector ends. The multiplicity indicates the number of instances that may be connected to each instance of the role on the other end. If no multiplicity is specified, the multiplicity matches the multiplicity of the role the end is attached to.

---

**Name:** WebInterface

**Description**

The WebInterface extends the UML metaclass “Class::Interfaces::Interface” (see fig. 1).

**Attributes**

No additional attributes.

**Constraints**

```
context Class::Interfaces::Interface
-- All operations of WebInterface are WService.
inv: self.isStereotyped("WebInterface") implies
    self.ownedOperation->forAll (o|o.isStereotyped("WService"))
```

**Semantics**

WebInterface represents the functionality the component to which it is associated offers to or requires from the rest of the system in order to be able to perform its task.

**Notation**

WebInterfaces has the lollipop notation (see fig. 2.a). There are two kinds of notations for each type of WebInterface. The required interface is depicted by an open semicircle, and the offer interface is showed by a complete circle.

---

**Name:** WebPort

**Description**

A WebPort extends the UML metaclass “CompositeStructure::Ports::Port” (see fig. 1).

**Attributes**

No additional attributes.

**Constraints**

```
context CompositeStructure::Ports::Port
-- A WebPort provides and offers WebInterfaces.
inv: self.isStereotyped(WebPort) implies
    self.provided->forAll (in | in.isStereotyped("WebInterface"))
    and self.required->forAll (in | in.isStereotyped("WebInterface"))
```
Semantics
WebPort is an interaction point between a WebComponent and its environment. It
decouples the internals of the component from the interaction with other compo-
nents, making that component reusable in any environment that conforms to the
interaction constraints imposed by its WebPorts.

Notation
A port is denoted by small squares on the boundary of the class symbol (see fig. 2.a).

![Diagram of WebComponent Notation and WebConnector relationship]

**Fig. 2.** Composition Model Profile Notation (1st Part)

Name: WebPattern

**Description**
A WebPort extends the UML metaclass “CompositeStructure::Structured-
Classes::Collaboration” (see fig. 1).

**Attributes**
No additional attributes.

**Constraints**
context CompositeStructure::Collaborations::Collaboration
inv: self.isStereotyped("WebPattern") implies
-- A WebPattern can contain WebParts or WebPorts or WebConnectors.
self.collaborationRole->forAll (cr | cr.isStereotyped("WebPart") or
  cr.isStereotyped("WebPort") | cr.isStereotyped("WebConnector"))

Semantics
WebPattern represents a Web architectural pattern, which is specified by a com-
posite element made up of a set of WebConnectors, and WebParts that corre-
sponds to Web components playing roles to accomplish a specific task or function.

**Notation**
A WebPattern is represented as a UML collaboration with a dashed ellipse icon
containing the name of a WebPattern (see fig. 3.b)

Name: WebPart

**Description**
A WebPart extends the UML metaclass “CompositeStructure::InternalStructures::Property” (see fig. 1).

**Attributes**
A WebPart has a property multiplicity, which using the notation \([x\ldots y]\) specifies
the initial instance or the amount of instances (x) when the WebComponent is
created, and the maximum amount of instances at any time (y).
Constraints

context CompositeStructure::InternalStructures::Property
inv: self.isStereotyped("WebPart") implies
-- A WPart has WebPorts.
Self.ownedPort->forAll (p | p.isStereotyped("WebPort"))

Semantics
WebPart represents a set of instances that are owned by composition belonging to a WebComponent instance.

Notation
WebPart is shown as a box inside a WebComponent or a WebPattern (see fig.3 (a and b). A multiplicity for a WebPart can be specified within the container WebComponent. (see fig. 3.a)

Fig. 3. Composition Model Profile Notation (2nd Part)

2 Topology of Configuration Model Components

The Configuration model defines an architectural style based on the structural view of the Web application by means of a set of Web components and their connectors, where each component represents the role or the task performed by one or more common components identified in the family of Web applications. As it is defined in [1], “an architectural style is independent from its realization, and does not directly refer to a concrete application problem it is intended to solve”. In this way, CM establishes a topology of components located in the web application domain which allows to specify the architecture without knowing anything about the concrete application problem.

In order to define an effective topology, WebSA has reduced the different WebComponents in the Web application to the minimum amount. Thus, it could be easier to identify the model elements.

This topology classification is used in two WebSA models: Configuration Model and Integration Model. A recollect process is carried out from the architectural elements defined in approaches such as [2, 6, 3, 4], and from the own variability study in the family of Web applications.
Next we will present the 20 subtypes of the stereotype WebComponent defined in sect. 1. Some subtypes are associated with only one logical layer, but there are others WebComponents which can be associated with any layer. In this way, we will divide the topology of WebComponents in two types, common and uncommon.

1.1 Common WebComponents

This type of components can be used in all layers, due to its task is identified in each one. In this way, it reduces the number of the different subtypes. Though, its final implementation could be very different depending on the layer. In the CM the layer associated with the component is not provided because the components are modelled independently of it. However, in the Integration Model the layer is important because the component is transformed into the implementation depending on the layer.

**EntityData:** is the data structure that represents a concept in the business domain. It is equivalent to a class in the domain model. In an EntityWeb, it represents the state of an object. In the integration Model EntityData will have OID, attributes, relationships and operations from a class of the domain model.

**LegacyView:** represents a gate of the WebApplication to the functionality which is provided or required by a legacy system. It could be associated the task of marshalling or unmarshalling the legacy invocations. It has a boolean attribute called isSynchronous in turn the invocation types that the legacy system allows.

**Store:** is a component that keeps some information in a persistent way. Also it has the mechanisms of management, and communication to provide this information to the rest of the components. It has three properties:
- **Read-Write:** it indicates which action can be performed in the store. Read or Write or Read/Write.
- **Organization:** there are three types (1) Relational: Database is composite by a set of related tables. (2) Object Oriented Database, it is more sophisticated than the relational because it doesn’t only deal with data, but it provides a language which allows defining the business logic in this layer. (3) Flat files: physical files which are specified in a proprietary format or in XML.
- **Data:** conceptual, navigational or presentation.

**Data Access Component (DAC):** it allows gaining access to the persistent data of the system. It implements a mapping between the entities and the data and vice versa. In this way, it makes independent the components from the kind of persistence of the system.

**Singleton Function Library:** it contains a set of special library functions like mathematical or financial tasks which are accessed by the rest of the system.
1.2 Uncommon or Specific WebComponents of a layer

These are Webcomponents of the Composition Model which are always related to a specific layer and/or sublayer. They are classified by the different layers of the Sub-system Model. For example, the ObjectWeb component can be related to the User-Interface layer, but it can’t be associated to the Presentation or Logic Control sublayers because the ObjectWeb integrates the functionality from both. Besides, if a component is related to a sublayer, it could be related to a more generic layer than the previous sublayer.

1.2.1 User Interface

The UserInterface layer represents the components developed to support the communication between the user and the Web Application. In this layer, we can choose between two patterns: Remote User Interface or Distributed Presentation. On the one hand, Remote User Interfaces define a thick client with a rich graphical User Interface and offline processing. On the other hand, Distributed Presentation defines a thin user interface layer that we called Presentation and it is separated from the User Control functionality. In order to select the most suitable components in this layer, we are based in Conallen’s work [2].

UserAgent: it gathers requests from the client and resends them to the server. For this reason, it becomes the ultimate recipient of the response. The most common example is a Web browser, which provides access to information services and renders service responses according to the application needs.

ObjetoWeb: This component is executed in the client providing the functionality as presentation as dialog control. Usually, it is carried out in the browser by means of plugins. Some examples are java applets, the flash components and the ActiveX.

CacheWeb: It keeps the information during a period of time into a scope. It allows to reduce the number of interactions improving the performance and the scalability. It has one attribute named scope that can have four different values: Session: it keeps the information for a session of a user. Application: The component keeps the same information for all the application since it starts. User: It keeps the information for a user. Request: It only keeps the information during a request of the client.

Resource: This component is an element used for the presentation into the user interface. Its attribute type has three values: Image, sound and video.

Next, we will show the elements that we can be associated to one of the sublayers: Presentation or User Control. In the same way, they can be used in the User Interface layer.

1.2.2 Presentation sublayer

The main function of this layer consists of showing and interacting with the final user of the application. In this layer will appear the next Web components:
**Client Page:** This component has the complete content to present the information to the user, gathering the input data, and also carrying out the request to the server (p.e. HTML, XML, flash). It can contain references to others components of the same type, or style or functional.

**Style Page:** This component only has the information is referred to the user interface (p.e CSS, XSL). It also can be reference by a Client Page.

**Functional Page:** It keeps the functionality for the interaction of the user interface of the application. Such functionality can be used for validation, presentation, navigation, etc. (p.e Javascript, ActionScript). It could be referenced by a Client Page. Its attribute type has three values: validation, presentation or invocation.

### 1.2.3 Dialog Control sublayer

This subsystem performs the processing of the functionality of the presentation about the navigation, and the requests to the business logic. The components of this sublayer are based on the elements of the MVC pattern [1] and Conallen [2].

**Server Page:** The component makes the processing of the Web page by means of scripts executed in the server side. Normally, it is make by an engine which runs the code of the page and returns the content to the Web Server. Such component contents the information about the presentation and/or logic of the server and it generates one or more Client Page components. Some examples are ASP, JSP, PHP, Coldfusion, etc.

**Controller:** This component receives the request from the presentation components, and resends them to the business logic in order to establish a reaction from the interface. It also redirects the request of navigation to the server pages, and it can configure the composition of the different pages to define the presentation.

**View:** This component obtains the information from the domain elements and it shows the information with output format and adaptability aspects. It makes independent the user interface aspects from the domain of the application.

### 1.2.4 Business Logic

It is the part of the system that resolves the business rules of the problem domain. Such layer can be or not divided in two sublayers: Process Control and Business Objects. The components of the business logic subsystem are based on the EDOC profile [6] and our experience.

### 1.2.5 Process Control sublayer

It is the server side that receives the request from the user interface, and it turns them into process which could be transactional or not, and synchronized or not.
**Process Component:** It represents an active processing unit. Such component can provide access to operations or can implements itself the operations by means of data from the entities.

It has the five attributes:

- **hasState** (Stateful or Stateless): A Boolean value that indicates if the component has an specific state through the interactions or not. The control mechanism in such session is not in the scope of this approach.
- **isTransactional**: A Boolean value that indicates if the operations must be carried out in a transactional way or not. By default its value is true.
- **isDistributed**: A Boolean value that indicates if the component is expected to be accessed over the network. By default its value is true.
- **isShared**: A Boolean value that indicates if such component can be accessed by multiples sessions.
- **isSynchronized**: A Boolean value that indicates if the component is receives synchronized requests or unsynchronized.

1.2.6 **Business Objects**

This subsystem contains the components that represents the objects are defined in the problem domain. These components can be from this type:

**Entity:** An Entity is an object representing something in the real world of the application domain. It incorporates EntityData that represents the state of the real world thing, and it provides the functionality to encapsulate the Entity Data and provide associated business logic.

Its attributes are:

- **isShared**: A boolean value that indicates if the entity can be shared by multiple, concurrent transactions or users. By default is true.
- **isManaged**: A Boolean value that indicates if the Entity type is managed. If it is managed, then the implementation provides a mechanism for accessing the extent of all instances of the type and its sub-types and may provide a mechanism for dynamically applying rules to all instances. This typically is implemented as a “home” or “type manager.”
- **isDistributed**: A Boolean value that indicates if the component is expected to be accessed over the network. By default its value is true.

1.2.7 **Persistence**

This part of the systems contains the persistence elements of the problem domain. The components of this layer are:

**ConnectionPool:** This component manages a set of data sources or connection to different store or legacy systems. It allows to reuse such resources for a set of users and reduce drastically the time to create or destroy each data source.

**DataSource:** This component provides a physical connection to a store of data or a legacy system. It has a set of properties:

- **isTransactional**: A boolean value that indicates if the data source allows transactions or not. By default the value is true.
isSecure: A boolean value that indicates if the connection is secure or not. P.e. it has a validation mechanism to establish the connection. By default is true.

isRemote: A boolean value that indicates if the data source access to a store over the network. By default is true.

3 Using the WebSA Profile in VisualWADE

VisualWADE [10] is a CASE (Computer-Aided Software Engineering) tool to develop Web applications. The goal of that tool is to simplify the design and implementation of Web-based information systems. This is addressed from a model or perspective by means of enhanced conceptual models. With VisualWADE, a Web application is modeled by three major viewpoints; Structure describes the organization of the information managed by the application, in terms of the pieces of content that constitute its information domain and their semantic relationships. Navigation concerns the facilities for accessing information and for moving across the application content. And Presentation affects the way in which application content and navigation commands are presented to the user.

For the purposes of this paper, we have incorporated to VisualWADE the architectural logical view. To do that, we have taken advantage of the extensible capabilities of VisualWADE by means of their plug-ins technology. A plug-in is a collection of some combination of the following: main menu items, shortcut menu items, custom specifications, properties (UML tagged values), data types, UML stereotypes, online help, context-sensitive help, and event handling. In this experiment, we have developed a plug-in which allows us to use the presented WebSA profile in VisualWADE. Therefore, we can use this tool to easily accomplish WebSA models.
The best way to understand that extension is to show a tangible example. Fig. 7 shows an snapshot of the VisualWADE tool were a Configuration model that specifies the presentation logic layer of an e-commerce web application according to the MVC pattern [2]. Following the Configuration Model profile, the example contains four specialized and instanciable stereotypes of the WebComponent Class. The WebComponent Control stereotype gathers the requests from the client-side (called Controller in the topology). The WebComponent Model (called EntityData in the topology) stereotype represents the data and functionality of the problem domain in this layer. The WebComponent View presents the data to the user and finally, the WebComponent Server Page represents the code of the web page that will be executed by the server-side.

Once a WebSA model has been specified, the designer can execute the WebSA plug-in to check the correctness of the model. This is done executing the menu option Plug-ins. MDA plug-ins. Check WebSA profile (UML 2.0 version), that can be reached from the main application menu. The execution of that validation process checks all the OCL constraints we have presented in the section 4. In the example an error occurs and the environment informs of such situation. The designer receives detailed information of the error in the task console. In this case, the message “Error: Context CMProfile::CM::WebComponent. Invariant Violation (self.attributes->size()=0), a WebComponent has not attributes” is showed to the designer. This means that an invariant violation in the context of a WebComponent has been produced. The profile specifies that structure features as associations are not allowed in a WebComponent. For that reason, the association relation between the Model component and the View component is highlighted. To solve the problem, the designer has to specify a relation stereotyped as WebConnector between both components.

5 Conclusions and Further Work

In this paper we have presented the Configuration Model profile to represent a architectural view for web applications. WebSA is a methodology that fosters the use of the MDA philosophy to define a set of Domain specific models to cover the Web application domain. The formalization of these models is done by means of a MOF metamodel. In order to be compliant with the MDA [5] standard, WebSA defines its metamodel as an extension of UML metamodel using the profile mechanism. This profile contains the needed stereotypes, tagged values and constraints for a powerful architectural modeling. The OCL has been used to specify the constraints attached to these new defined elements, thereby avoiding an arbitrary use of them. Furthermore, this WebSA profile has been implemented in the context of a CASE tool to design web applications by means of a plug-in technology, thus validating the correctness of WebSA models.

Currently, we are working on the set of QVT [9] transformation models to support the WebSA refinement process. It allows us to formalize the transformations model-to-
model and model-to-code and it guarantees the traceability between those models and the final implementation.

6 References

8. Object Management Group, Meta Object Facility (MOF) v1.4, OMG doc. formal/02-04-03