XU Product Architecture

User Interface Position Statement

Visual Composer, Web Dynpro and Smart Clients

Version 1.0

20.10.2005

Table of Contents

Table of Figures

Organization

Contact Persons

Name	Project Role / Comment
Hammerich, Reiner	Architect
Parann-Nissany, Gilad	Architect
Mullaseril, Paul	Architect, Project lead

History of the document

Version 1.0 STATUS: final

Changes and updates:

Date	Chapter	Name	Change/enhancement	Agreed with
Oct. 19, 2005			Draft	

1 Executive Summary

The central message of this position paper is around the interaction between Visual Composer (VC) and Web Dynpro (WD) which must be seen as “Web Dynpro provides the components and the runtime infrastructure for building coded, configurable building blocks, while VC is the environment for freely modeling front end applications using these building blocks”.

- We recommend enabling the creation of plug-in enhancements to VC/WD applications in non-SAP programming models; this would mean that even though the overall application is VC/WD, plug-ins using e.g. MS XAML or Macromedia Flex would be possible.
- Specific areas do exist (Mendocino, SBS, and so on) where the UI programming model is required to be different (MS XAML, Macromedia MXML, …) yet we still wish to use VC and the UI Services provided by Web Dynpro; see recommendation 8 below
- A “visual language” for describing the models has been created in VC; it involves a visible graphical representation of the various elements that appear in the model (e.g. a service, a UI form, and so on). This XML description of the design-time models is by now available in two releases, the VC ’04 (“VC Freestyle”) release calls it GML (Graphical Modeling Language) and the VC ’05 (“VC Patterns”) release calls it VCL (Visual Composition Language).
- The VCL & GML dialects are conceptually similar and share many details. However at this point VC ’05 and VCL have been used only for the A1S UI Pattern project, and emphasize modeling within the constraints of Patterns. A consolidation project exists that brings these together into what is sometimes called “VCL+”.
- VC’05 also supports simple extensions (plug-ins). The plug-in's are Web Dynpro components that do not need a build time plug-in or a VC Kit. These plug-in’s are configurable in VC.
- VC’05 UI Patterns implement constraints and special implicit behavior in floorplans. In the concrete implementation done for Web Dynpro Patterns and VC ’05, VC and the pattern specific kit are used to configure the behavior of the pattern but some of the behavior and constraints are implicit.
- VC ’04 also provides a so-called XGraph Language (XGL) representation that contains a blueprint of the execution semantics and is optimized for runtime. A transformation from GML to XGL is provided. This is an enabling factor for more free-style modeling, since pieces taking part in the composition have a clearly defined contract of expected behavior.
- XGL can be used in 2 flavors (marked as option 1 and 2 in figures 2, 5 and 6).
- Starting from XGL one can generate into a concrete runtime environment. This requires the implementation of a generator that covers a transformation of all of XGL into the target platform representation. The result can then be deployed to the target platform, compiled there, and treated like any other running software developed in that target environment.
- As an alternative one can implement an interpreter that covers all of XGL in a certain platform. In that case it is required to deploy XGL to the runtime environment where the interpreter is installed. The interpreter then executes the model.
- The Web Dynpro client abstraction layer enables one to implement new protocols for Web Dynpro clients. The task of the client abstraction is to translate the Web Dynpro UI model into a concrete client technology which is produced at runtime.
- The general direction SAP is currently heading towards is the usage of the Web Dynpro runtime as a general purpose runtime that will serve as a framework for the enablement of various client technologies. Web Dynpro supports VCL as a design and build time environment, and as we move forward, support for XGL will be added as well.
- We are therefore able to propose several activities for further unification of our architecture.

    1. Continue support and encourage the already existing “consolidation” project that brings GML and VCL together into “VCL+”, supports “XGL+”, and creates a Web Dynpro interpreter for XGL+.

2. Full enablement and migration of Web Dynpro Patterns to VCL+ and XGL+ (sometimes called “super patterns” or “design time patterns”)
3. Making coding Web Dynpro **plug-ins** easier, when within a VC model, a high priority.
4. Develop guidelines internally at SAP on how to model and how to mix **plug-ins** into models.
5. Alignment and consolidation of the UI **elements and controls**.
6. Unification of the overlapping portions in the **meta-models** of WD and VCL.
7. Evolution of the **programming model** to support unification.
8. Making VC, VCL+, XGL+ available and exposing general and UI services outside of the UI technology stack, to support non-UI needs in SAP. In addition support areas (Mendocino, SBS, and so on) where the UI programming model is required to be non-SAP (e.g. MS XAML, Macromedia MXML, …)
9. Development of plug-ins to support **UI rendering** (in several needed programming models; e.g. Web Dynpro, MS XAML, Macromedia MXML …) in a late binding way and in the appropriate IDE's.

2 Introduction

2.1 Goal and Scope

This paper was written in answer to a request to clarify the relationships between various pieces of our UI strategy; focusing on Visual Composer, Web Dynpro and the Macromedia Flex smart client technology. This paper is not intended to serve as a general UI strategy and hence other topics like Portal, outdated UI technologies, competitive analysis, a detailed smart client strategy, etc. are not in scope of this position paper.

We begin with some statements explaining, at a high level, what are the technologies about, and then continue with a statement of direction and position.

2.2 Target Group

Management, Architects, Developers.

3 Visual Composer

Visual Composer (VC) is a model-driven design time tool for enabling the modeling and creation of application front ends, involving user interface and services. The target group of this tool is business analysts and also developers. VC aims to offer an easy to use environment, that lets the users concentrate on the business content where programming skills or other heavy technical background is not required. Modeling capabilities allow defining how UI building blocks (*interactors* and *scenarios*) are interlinked and interact with *services* in the backend.

A “visual language” for describing the models has been created in VC; which involves a visible graphical representation of the various elements that appear in the model (e.g. a service, a UI form, and so on). Technically this graphical representation is stored as an XML-based description of the design-time model. This XML based description is really at the heart of the tool, as it is used whether the view of the model is graphical, layout, preview, or any other. It is also the storage format.

This XML description of the design-time models is now available in two releases, the VC ’04 (“VC Freestyle”) release calls it GML (Graphical Modeling Language) and the VC ’05 release (“VC Patterns”) calls it VCL (Visual Composition Language).

VCL and GML are conceptually similar and share many details, yet they have also some differences. VC ’05 (and VCL) is currently being used to create applications in the A1S Web Dynpro Pattern UI and emphasize modeling within the constraints of Patterns. VC ’04 (and GML) is being used to create applications in the mySAP Fast Track and has emphasized more “free style” modeling without (or with fewer) hard-coded constraints. A consolidation project exists that brings VCL and GML together into one language which is termed “VCL+”.

VCL as well as GML are highly declarative modeling languages, yet they allow expressions to be used within the models. These expressions introduce, for example, simple field calculations or conditional branching (a simple condition can be placed on a line in a model) during design. The goal is that complexity of these expressions should be low; for comparison – they are similar in complexity to the way MS Excel allows a non-programmer to write expressions in the cells of a spreadsheet.

“**Where is the code?”** VCL even offers the capability to write Web Dynpro based plug-ins (Web Dynpro Model plus Java Code) in case the modeling capabilities are not sufficient. The same feature is planned to be added to GML.

VC supports a quick creation of UI applications, and full lifecycle support for creating applications directly from the models is available in the tool: e.g. you can model an application, find errors in the model, save the model, and run the modeled application, all from the tool; in the New York timeframe we are also adding NWDI integration and in future capabilities such as model reuse. Several views on the models are offered, including a graphical modeling view, a WYSIWYG layout view, a preview, and so on.

VC is a browser based tool with a supporting server-side “Development Server”. The browser based tool is “rich” in its capabilities (meaning it does not do a round trip for every operation but rather has substantial client side capabilities, uses SVG and JavaScript for a richer client experience and communicates via XML with the server side when it needs data). In the Paris and New York time frame, the Development Server side provides support for NWDI integration.

3.1 Visual Composition Language and Web Dynpro Patterns

Visual Composition Language is an extensible language. The basic modeling constructs (interactors, services, links, and so on) can be extended and specialized. It is possible to write a composition unit to implement these extensions. This unit consists of a VC kit (defining the extension and its design-time behavior; written in a script-based language based on Java Script), plus optionally a build time plug-in (in Java) and a runtime component (in Java). These 3 parts together define a building block e.g. a new pattern that can be used in a floor plan.

UI Patterns implement constraints and special implicit behavior. In the concrete implementation done for Web Dynpro Patterns and VC ’05, VC and the pattern specific kit are used to configure the behavior of the pattern. However, some of the behavior and constraints are implicit. The final execution semantic is defined by the Web Dynpro runtime component and runtime services. Hence, it is not possible to understand the overall execution semantics of a pattern and the embedding floorplan from the VCL model alone. Nevertheless, due to personalization, customization and dynamic runtime meta data, this goal is difficult to achieve since the complexity of the models increases significantly.

The pattern configuration is interpreted by the pattern components and the floorplan at runtime. The floorplans are implemented using Web Dynpro Java technology. Hence all building blocks in a floorplan are also Web Dynpro components. The only Web Dynpro entities that are generated at build time are Web Dynpro configuration data. The entire infrastructure is bound to the underlying Web Dynpro technology. Serving different client platforms is achieved through the Web Dynpro Client Abstraction.

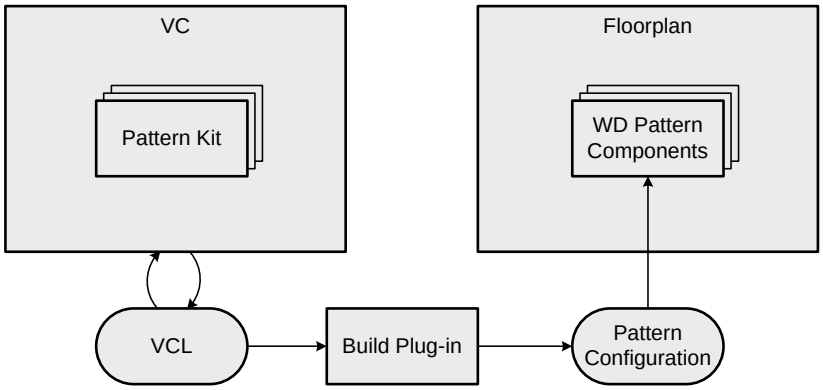

Figure 1 - VC and WD Patterns - high level overview

3.2 Graphical Modeling Language

Graphical Modeling Language (GML) in VC ’04 is currently a *non extensible* language. UI applications need to be described within the language. However a “black box” interactor allows limited extensibility (limited compared to VC ‘05) – without supporting drilling into the box and seeing its internal model.

GML is also a design-time language (as is VCL), optimized for design-time convenience. VC ’04 also provides a build time XGraph Language (XGL) representation that completely contains a blueprint of the execution semantics and is optimized for runtime. A transformation from GML to XGL is provided. The point is that XGL has much of the runtime semantics (for example the order in which events need to be fired in the runtime), regardless of the specific target implementation platform.

In this sense VC ’04 with XGL offers a capability unavailable on VC ’05 and Web Dynpro Patterns at this time. The WD Patterns and VCL configuration and implementation are private to a specific building block, while XGL gives an explicit public specification of the desired runtime behavior. This is an enabling factor for more free-style modeling, since pieces taking part in the composition have a clearly defined contract of expected behavior.

Of course *within* a coded component one may do as one likes, but one must respect the conventions of the standardized semantics and contract defined by XGL.

Constraints – in the XGL approach – are declared rather than implicit (though some code can of course support the explicit declaration). Therefore one can create in future so-called “design time patterns”. UI applications, according to this goal, will be implemented by some wizard that creates a GML model at design time; this can be done in a way guided by pattern guidelines; yet if more free style changes are needed one may begin to change the generated model.

We have mentioned before that a consolidation project is under way already, to merge VCL and GML into VCL+ with an appropriate XGL+, and thus to bring the benefits of XGL+ and “free style” together with the benefits of the WD Pattern world.

3.3 Generation and Deployment Model for XGraph Language

XGL can be used in 2 flavors:

1. Starting from XGL one can generate into a concrete runtime environment. This requires the implementation of a generator that covers a transformation of all of XGL into the target platform representation. The result can then be deployed to the target platform, compiled there, and treated as any other running software developed in that target environment. Examples for this approach are the Fast Track GML to HTMLB generator for ESS models and the Fast Track GML to Macromedia Flex generator for analytic dashboards. Certain features like runtime configuration, end user personalization, custom extensions and dynamic meta data handling are difficult to achieve with this approach. It must be assumed that at least a certain runtime library is pre-deployed on the target platform for that purpose.
2. As an alternative one can implement an interpreter that covers all of XGL in a certain platform. In that case it is required to deploy XGL to the runtime environment where the interpreter is installed. The interpreter then executes the model. An example for this approach is the Fast Track Web Dynpro XGL Interpreter.

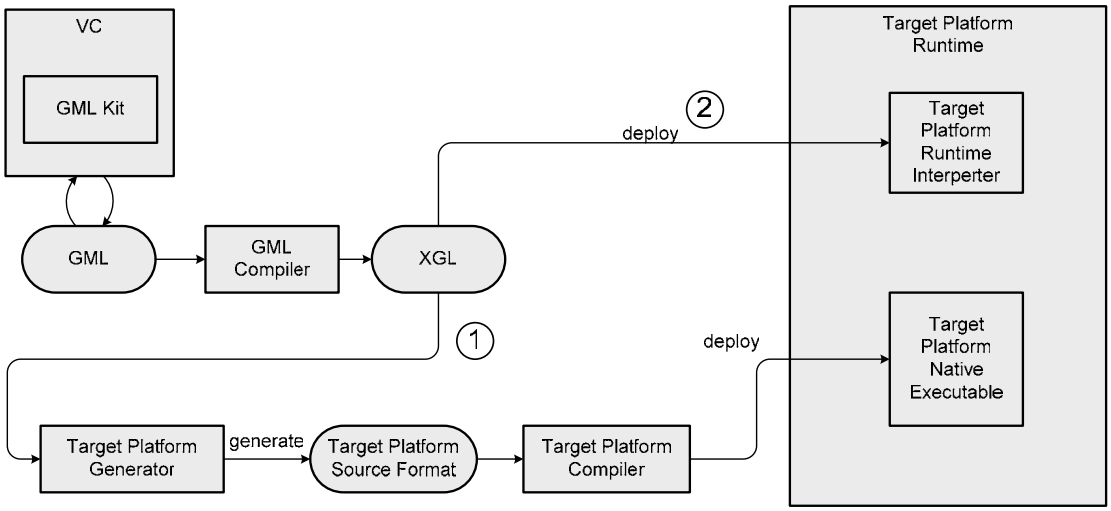

Figure 2 - Logistics of GML, XGL and the runtime

In both cases it is required to implement a piece of software that is able to translate the complete XGL specification into the target platform. Therefore mainly product definition, performance and maintenance consideration should drive the decision which approach is to be preferred.

From the diagram one can read that a runtime interpreter (2) is a more straightforward and simple solution as long as one has the entire stack under control. It requires only the deployment of XGL to the target platform interpreter and therefore streamlines the software logistics. In case where one does not want to touch the target platform runtime or one has considerations like a very lean runtime, option (1) is the right choice.

Option (2) in some situation will also mean runtime performance penalties, in case the interpretation of XGL becomes too expensive. The penalties can be avoided by an also possible hybrid solution that deploys the XGL representation to the runtime environment. The runtime itself can now judge if it is necessary to transform XGL into a much more suited runtime representation. That would mean, the runtime interpreter generates an appropriate representation on the fly and stores it in a persistent or in-memory cache. This is a solution that is very similar to the R/3 “On Demand Compilation” or JSP.

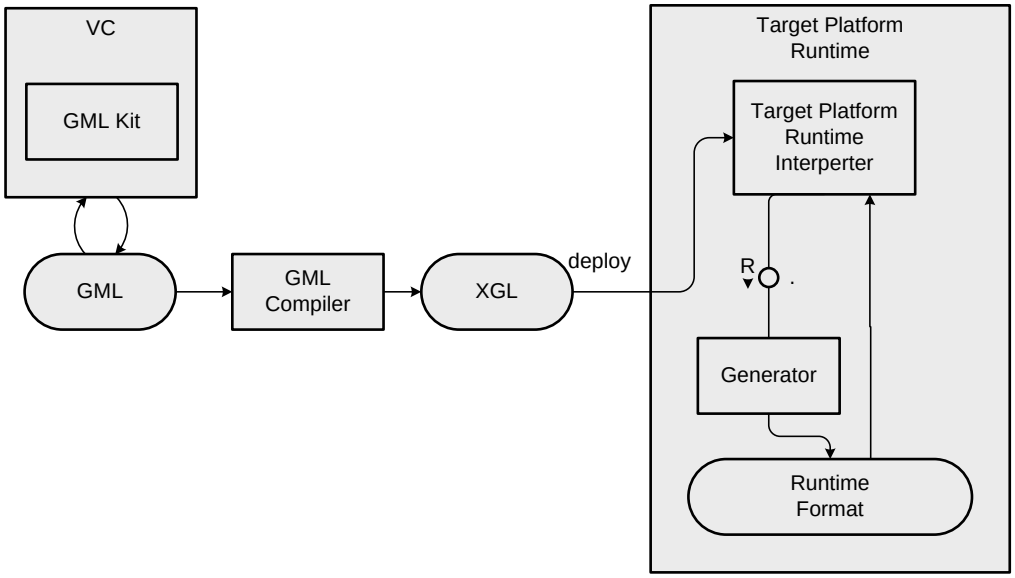

Figure 3 - Future optimization possibility - mixed XGL interpretation / generation "on the fly"

4 Web Dynpro

Web Dynpro is a comprehensive set of UI services as well as a declarative UI programming model. Compared to VCL and GML the programming model is a fine grained model for use by software engineers and enables to implement code; it offers therefore more freedom and flexibility for this audience. Web Dynpro is not only 100% pure modeling but also enables to implement code. The target audience of this tool is mainly developers and consultants.

The environment for Web Dynpro tools is the SAP NW Developer Studio (for WD Java) or the ABAP Workbench (for WD ABAP). A set of tools is available for creating and maintaining Web Dynpro models and the contained code. From there tools for the complete software development cycle management starting from development to testing and debugging are available. Web Dynpro Java is fully integrated into SAP's NWDI and J2EE deployment.

Web Dynpro supports a rich set of services like meta data handling, error handling, value help, personalization, data type support, software logistics, SAP standards, accessibility, UR based rendering, Portal integration, I18N, and more out of the box functionality. Most of these services require no programming or declarations; they work seamlessly.

Web Dynpro serves as framework for the implementation of the composed pattern runtime. By utilizing all services including the Web Dynpro model composed patterns are fully portal-aware enterprise ready applications. Web Dynpro has a history of almost 4 years and several shipments including applications built on top of Web Dynpro. Hence, one can assume that the Web Dynpro runtime has reached a certain degree of maturity. This includes a comprehensive concept for reuse including NWDI integration, sophisticated caching algorithms, mechanisms and coding practices aimed at optimizing page content transfer so as to reduce refresh speed and bandwidth usage.

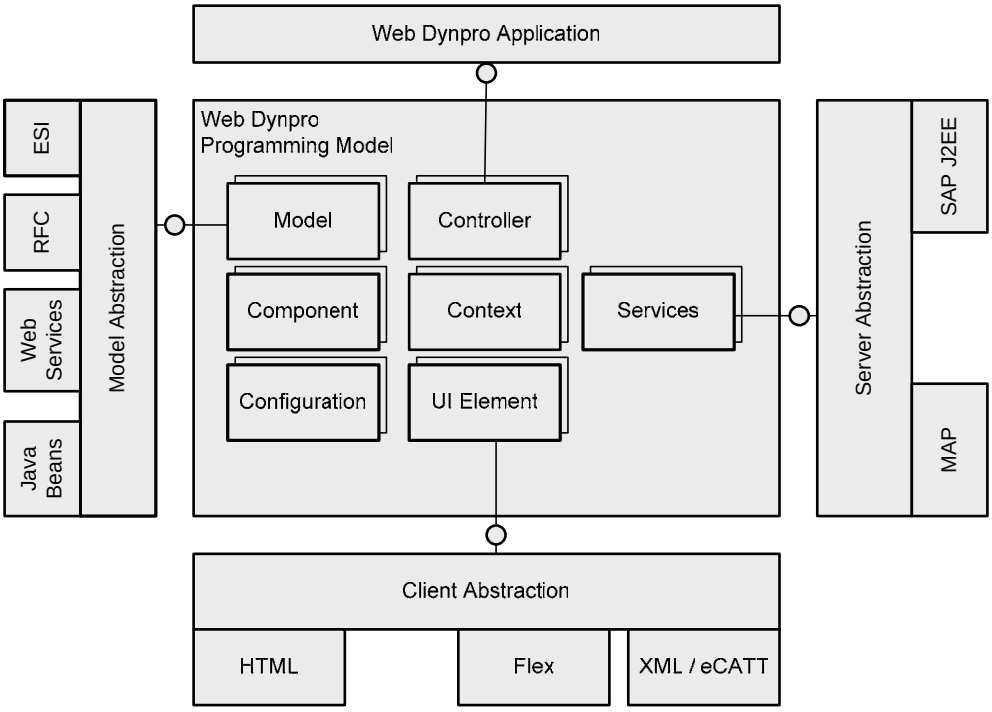

Figure 4 - Web Dynpro Programming Model

4.1 Web Dynpro Client Abstraction (CAL)

The Web Dynpro client abstraction enables one to implement new protocols for Web Dynpro clients. This abstraction shields the Web Dynpro model from client technology specific aspects. The task of the client abstraction is to translate the Web Dynpro UI model into a concrete client technology. This mapping is produced at runtime. Currently Web Dynpro supports HTML, Web Dynpro native clients, Blackberry, PocketIE, eCATT, and Flex as prototype. The CAL allows having client-side behavior through logical separation of data, layout and data-binding information.

In general the CAL allows the execution of any Web Dynpro application on a certain client technology, as long as all required Web Dynpro UI entities are available on that client. In most cases this means that the client must adapt or extend its feature set in order to match the Web Dynpro definition (meaning, for example, *the client developer must write implementations for the Web Dynpro control set required by a specific UI use case*).

5 VC and Web Dynpro: Current State

The general direction of SAP is currently the usage of the *Web Dynpro runtime as a general purpose runtime* that will serve as framework for the enablement of various client technologies. It will replace e.g. the current HTMLB. At the same time SAP is currently heading towards *VC as the modeling environment* for UI applications.

Web Dynpro supports VCL as a runtime environment, and as we move forward it is adding support for XGL as well. The overall evolution may be seen in Figure 5 - Current Architecture of Visual Composer (both GML & VCL) and Web Dynpro.

In the figure below we see the current state. We see the two options that have been already discussed (option 1, generate from XGL to a target runtime; and option 2, directly interpret XGL). We also see how these possibilities are currently being used.

Note: The state described as current state is what is already developed and delivered plus also the already designed, planned and budgeted activities in New York, Fast Track for Pattern and Fast Track ESS development.

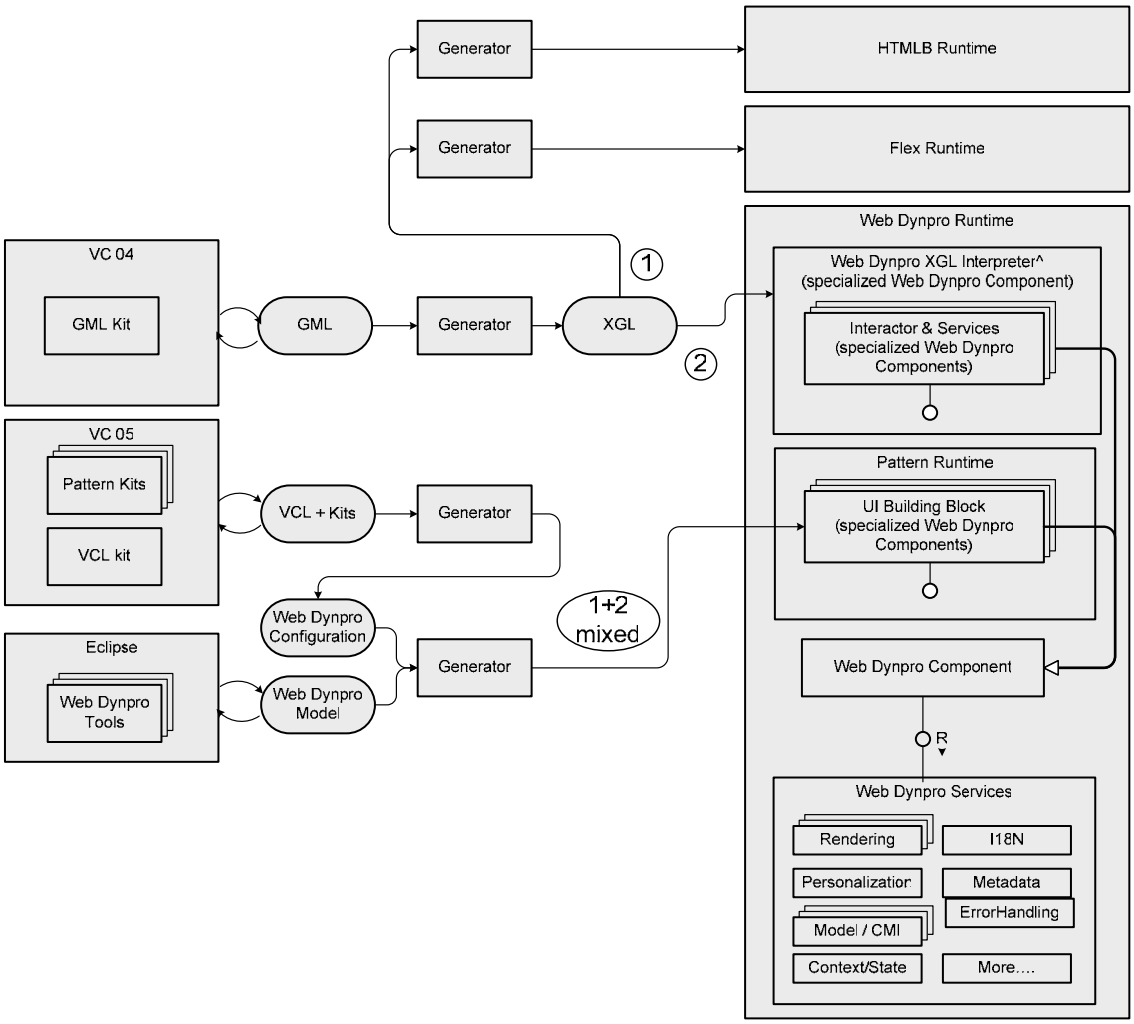

Figure 5 - Current Architecture of Visual Composer (both GML & VCL) and Web Dynpro

5.1 Challenges

The current architecture of WD Pattern with VCL has been discussed in "*Visual Composition Language and Web* Dynpro Patterns", and is summarized in the figure above as a combination of generation and interpretation.

This current state has some challenges:

- The *different models must be m*apped from VCL/GML to Web Dynpro and then to the client technology HTML or Flex.
- VCL/GML and Web Dynpro have a different model for similar concepts. This makes it *difficult to implement plug-ins*. For example constructs in the models should be more easily identified in the code and vice-versa.
- Improvements in the *integration of code plug-ins to the models* are needed, for example an ability to call plug-in methods and properties from the expressions in the model could be desired.
- Some infrastructural aspects are different for VCL, GML and Web Dynpro which creates a *difficult software cycle*. Examples are reuse and the components concepts,
- The *interoperability* between WD, VCL and GML is limited.
- There is no *reuse in VCL and GML* at this time; a concept however is available and in work
- There is no *reuse between VCL and GML* (this is conceptually not possible)
- Because of the differences mentioned above, supplying general purpose components like ALV for WD and VCL and GML involves *duplication of effort and is also error prone*.
- Generic concepts like personalization, configuration, value help and error handling require *a lot of effort to unify* and achieve across this environment.
- Web Dynpro *adoption in mySAP Business Suite is slow*. Currently Web Dynpro does not offer applications some of the functions they need (see gap list from UTC). Missing interoperability with Dynpro technology forces mySAP Business Suite to implement extensions to existing products also in Dynpro technology.
- *Complexity of VCL / WD Pattern* dev infrastructure (the "1+2 mixed" approach in the figure has implications to the complexity of the infrastructure)
- *Flexibility in VCL/ WD Pattern* is too limited. This also reflects into service modeling (e.g. ESI transformation nodes).
- *Large skill set required* for development of composition unit in VCL / WD Pattern (meaning, adding a new building block in WD Pattern requires an understanding of VCL, WD, WD Pattern specific infrastructure, Eclipse, VC, Portal, ESI, ESR…)
- *Free style modeling in GML creates complex models* today; concepts are in work to further simplify them
- We are *only beginning to gain experience* with the shipping and maintenance of models (e.g. the Analytics Dashboards is the first SAP shipment of a modeled application)
- A customer or *ISV cannot currently extend the Web Dynpro and VC* approaches by writing their own components or controls; so enhancing / extending the SAP offering is limited
- *Web Dynpro is a complex environment* that requires the understanding of many technical details of the programming model and the programming environment for development of an application. The learning curve of developers with this environment is slow today.

6 VC and Web Dynpro: Future State

What follows is the convergence of GML, VCL and WD into a single target architecture in a future release. *This architecture will bring together the benefits from both GML and VCL into a single model called VCL+ with an execution semantic called XGL+.* First activities are already undertaken. This is a project that aims the convergence of GML and VCL into a single model.

Now looking forward to the recommended future state, we see the figure below. **The option "2" – to interpret XGL+ by the WD runtime – is the mainstream for internal SAP usage and applications. Option "1" – to generate directly to a target runtime – is kept for cases like SBS (Small Business Systems) and Mendocino, where market requirements exist that limit the choice of target platform. For example, in SBS the requirement is to have a very small footprint e.g. the flash files.**

Note that the WD "rendering" box will use Flex as one of its standard rendering possibilities through an appropriate WD Client Abstraction (see section "Web Dynpro Client Abstraction (CAL)"). The abstraction of different client runtime technologies can be implemented inside the Web Dynpro framework.

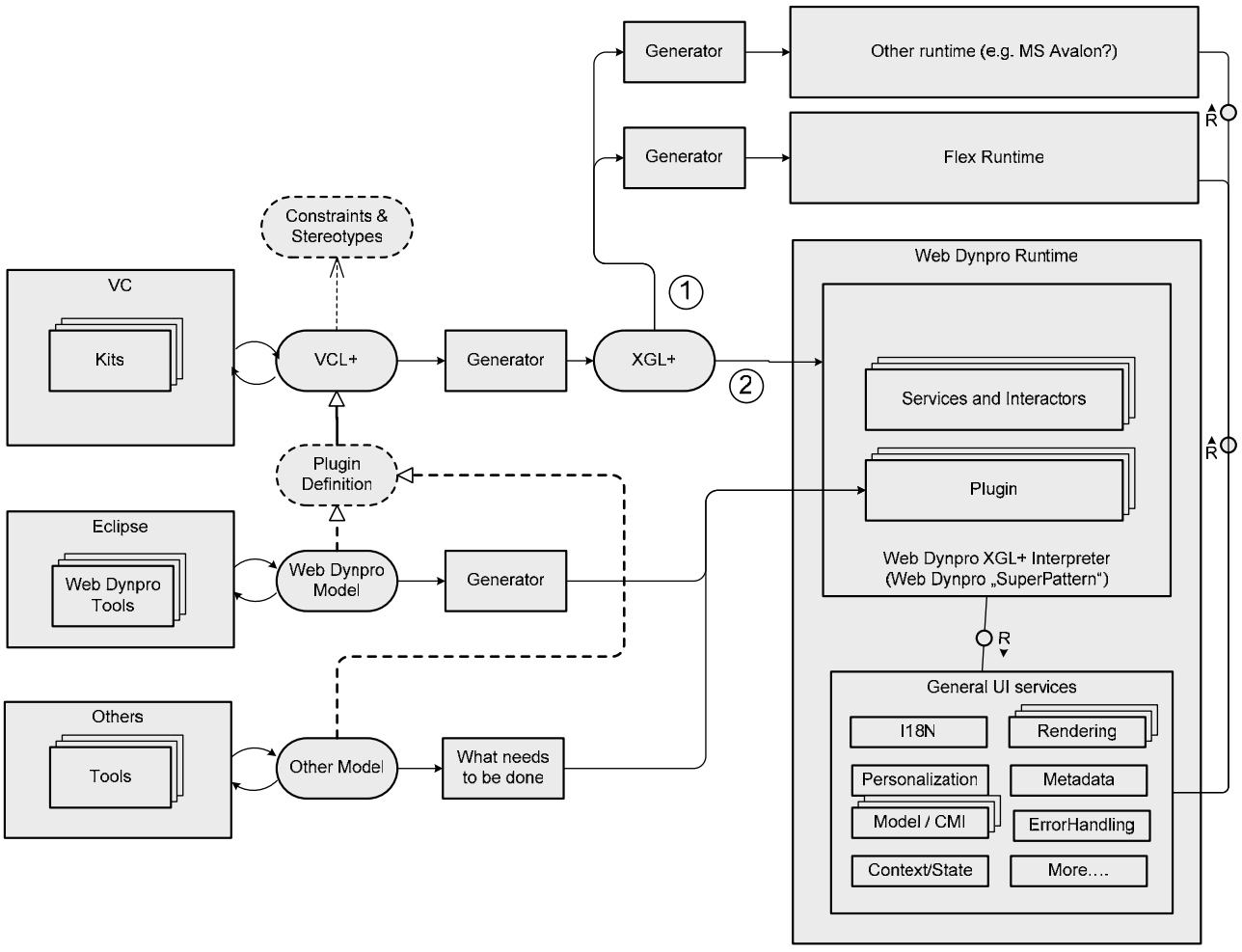

Figure 6 - Future target architecture for Visual Composer and Web Dynpro

UI Patterns will run within an environment described by the XGL+ format. Constraints will therefore be made explicit in the contract and (even if they are coded constraints) will be declared and thus enabling design time

patterns. For example if one would like to restrict the number of fields in a form within a tab strip to a maximum, this would be expressed as a constraint on the number of fields.

There are several areas where requirements exist for a direct compilation from XGL to a target platform for the foreseeable future; however they should be viewed as the exception not the rule. These currently include analytic dashboards and B1S (these use direct compilation of XGL to Flex). Therefore a direct compilation from XGL to a target platform will continue to be possible.

Internally at SAP we are utilizing Web Dynpro as a general purpose runtime environment with several advantages:

- Utilize Web Dynpro as the component technology and programming model for building components and plug-ins for VCL+
- Make use of the implementation of the rich set of services and SAP standards already implemented in the Web Dynpro runtime environment
- Make all modeled UI applications per se portal aware
- Exploit Web Dynpro infrastructure
- Exploit Web Dynpro CAL for different client technologies
- Make XGL(+) the contract between design time and runtime
- Allow “Adaptive UIs”, which adapt to the environment at runtime. This environment is typically substantially different at each customer site, where the XGL is deployed to.

The interaction between VC and Web Dynpro must be seen as “**Web Dynpro provides the components and the runtime infrastructure for building coded, configurable building blocks, whilst VC is the environment for modeling UI applications using these building blocks**”. This will enable SAP to exploit the advantages of both technologies and endow an environment that has much more value than simply the sum of the value of each technology. A precondition for success is a consolidation of overlapping concepts in VCL/GML and WD.

We are therefore able to propose several activities for further unification of our architecture.

6.1 GML and VCL Convergence Project

It was already mentioned previously that a project is underway to consolidate GML and VCL. This project is building some of the pieces of path “2” in Figure 6 - Future target architecture for Visual Composer and Web Dynpro though not yet all of them. The pieces under construction include

- A consolidated “VCL+” modeling language for the design time, combining features of VCL and GML
- A consolidated “XGL+” that brings the benefits of the XGL contract between design time and runtime to the VCL+ user.
- A Web Dynpro interpreter that is able to run XGL+

This project is therefore already delivering on important parts of the desired future architecture.

Next steps needed (but not yet addressed by a project) would include the full enablement and migration of Web Dynpro Patterns to “super patterns” (sometimes called “design time patterns”). They are completely declared with VCL+ and XGL+, including explicitly declared constraints and stereotypes where necessary, so that even if a pattern component is coded its constraints and stereotypes are explicitly declared and the runtime semantics of all model elements conform to the XGL+ runtime specification rigorously. Ensuring backward-compatibility would be a challenge such a project should address.

6.2 Making the Coding of WD Plug-ins Easier Within a VC Model

Making coding of Web Dynpro plug-ins easier, when within a VC model, is an ongoing high priority. Several important steps are already happening in New York but they are only initial ones. Below when we discuss the long term trend of a unification of the meta-models, we will see how the programming model of such plug-ins

becomes much more convenient and useful. For example event handling, data binding, or the description of data, could all be simplified if the XGL meta-model and the Web Dynpro meta-model are unified.

Guidelines on how to model and how to mix plug-ins into models internally at SAP are recommended.

6.3 Alignment and Consolidation of the UI Elements and Controls

Another immediate step must be the alignment and consolidation of the UI element / control model of all SAP UI technologies. This will result in a common control set for XGL+ and WD. This will eliminate the need to transform between the XGL+ UI model and the WD UI model. The adoption of new client rendering technology will become easier and more robust. The question of how to offer UI elements in VCL+ needs to be driven by usability. Many aspects of the underlying fine-grained WD concepts can be hidden from the business analyst. Examples are the nesting of various containers, the capabilities of the table UI element, etc. Another option would be not to offer UI elements in VC at all and simply derive the appearance from meta data, stereotypes and patterns. In that case an alignment would no longer be necessary.

In addition it becomes much easier to have a similar UI behavior at design time and runtime. Hence, a common set of UI controls is a big step towards enabling WYSIWYG design time editors or previews.

6.4 Alignment of the Meta-Models of WD and VCL+

Further architectural steps are required. The NetWeaver Architecture 3yrs UI strategy paper already paints a visions how next steps SAP modeling and development environment(s) must be based on top of a common modeling technology and shared common Meta models. Overlapping concepts for various domains could look like.

Note: Figure 8 - Draft meta-model for UI domain language illustrates a vision. Until now it is not clear how far we can achieve this and what the price is. It is neither a detailed design nor it is a class diagram that is discussed widely within NetWeaver or AP. Hence, it should be read as a statement of intent. Details will be laid out in the upcoming UI strategy paper and needs deep involvement of the development teams.

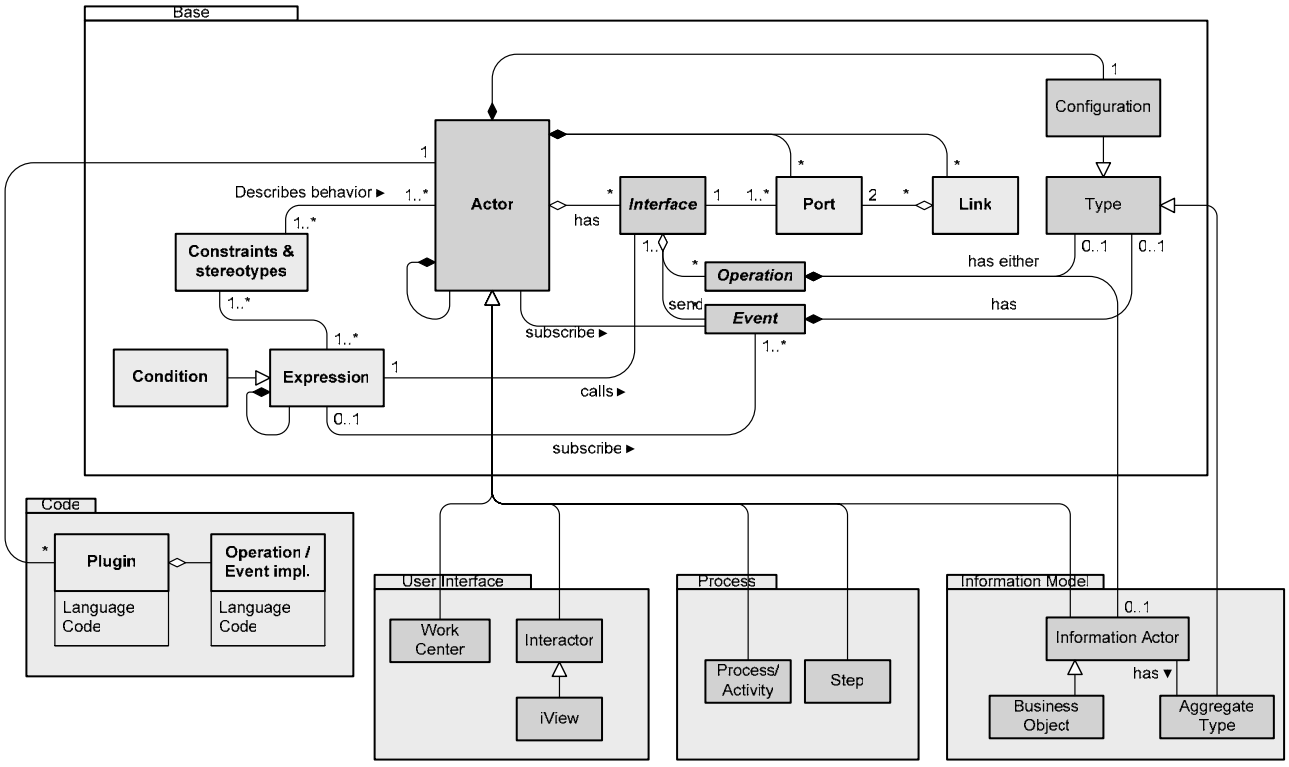

Figure 7 - Draft meta-model for UI domain language

The SAP modeling and development environment(s) must be based on top of a common modeling technology and shared common meta-models. Here, the MOIN project is key enabler. Having a common modeling technology in place that supports derivation between various packages allows the definition of common parts (base). Overlapping concepts for various domains can hence be defined in a common model. Examples for overlapping concepts found in VC/WD/ESI are as follows:

- Actors/components/services
- Infoshapes/context/BONodes,
- Dataflow and binding
- Support for code and expressions
- Component concept
- …

Different domains, packages or environments can then derive from such a common, shared meta-Metamodel and extend the definitions by domain specific requirements. Hence, one of the major investments must be an alignment of relevant parts of GML, VCL and WD concepts in overlapping areas.

From the technology side, an existing project called “MOIN” is an enabler for a common repository. This meta would give us a common modeling technology that supports derivation between various packages and allows the definition of common parts (base).

Note: Figure 8 - Draft meta-model for UI domain language illustrates a vision. Until now it is not clear how far we can achieves this and what is the price. It is neither a detailed design nor it is a class diagram that is discussed widely within NetWeaver or AP. Hence, it should be read as a statement of intent. Details will be laid out in the upcoming UI strategy paper and needs deep involvement of the development teams.

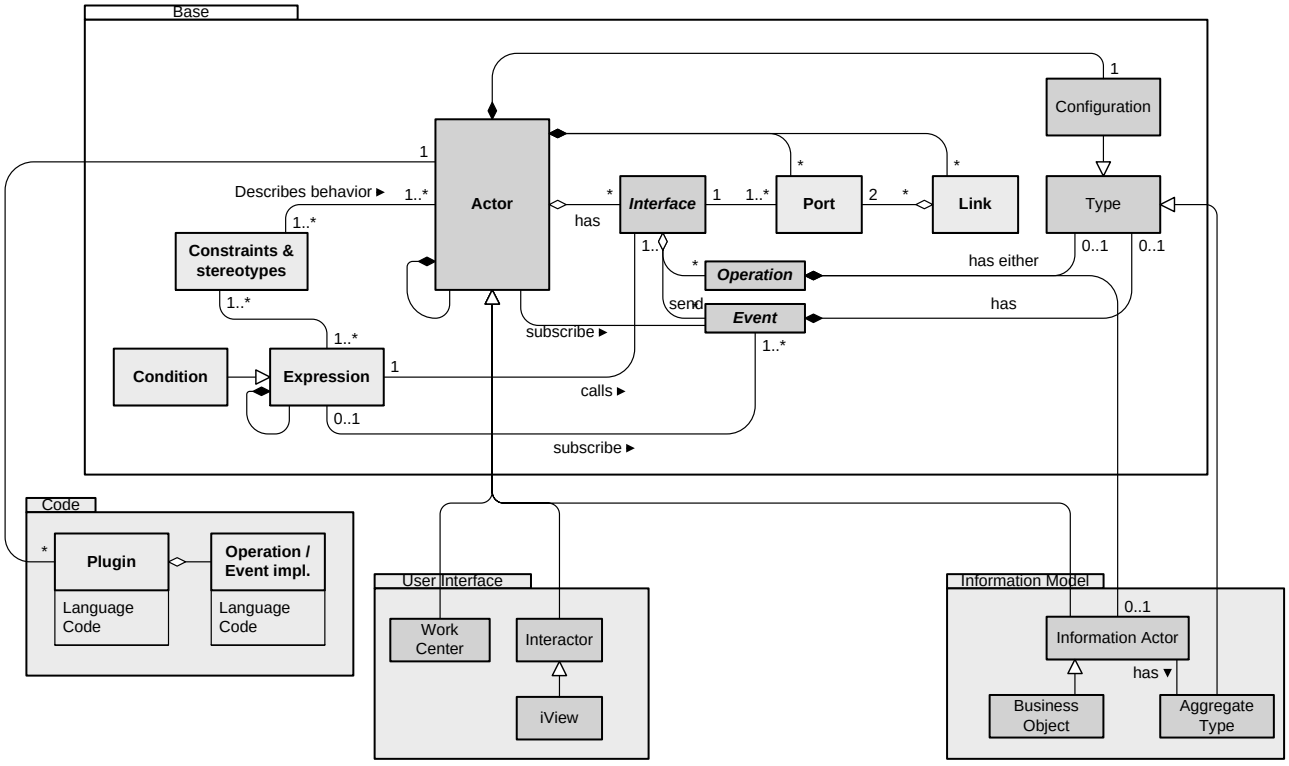

Figure 8 - Draft meta-model for UI domain language

This meta-model work should also be done in conjunction with the alignment of runtimes implied in Figure 6 - Future target architecture for Visual Composer and Web Dynpro.

The NetWeaver Architecture 3yrs UI strategy paper already paints a visions how next steps could look like. Please refer there for more details.

6.5 Exposing Services of General Interest

There exists a set of services that are required for efficient UI development. Typical examples for such services are:

- Connectivity and data access services
- Meta data handling
- Software logistics
- Components and component usages
- Support for translation, accessibility, and so on
- Data type support
- Error handling
- Transporting data sets efficiently to the front end (as XML)

We have several scenarios within the Product and Technology Group (PTG), as well as with ISVs, that would benefit from using such services. We currently have a requirement to use very lean or non-SAP UI programming models in Mendocino (where we use Microsoft UI technology), Small Business Systems (where we use Flex UI technology directly), and in relations with some ISVs (e.g. Macromedia).

The obvious question arises, how we make best use of our NetWeaver and SAP offering in these areas where the UI programming model is required to be different.

A proposal would be as follows: use as many as possible of the general services. For example one can expose SAP Web Services and meta data into the Avalon or Flex UI programming model; or use our translation or software logistic facilities to support our efforts in these cases.

This discussion is also related to “option 1” – generating directly to a target environment - above (e.g. Figure 6 - Future target architecture for Visual Composer and Web Dynpro). The services mentioned would be also exposed to this target environment to enrich it. We already offer GML or VCL support in some of these cases; indeed this is being done in the projects of Small Business Systems as well as in the efforts for the Microsoft cooperation. In these cases we generate directly from GML/VCL to the target programming model; and as this proves itself useful for many purposes we should continue to do so. Additionally, the proposal would be to use general UI servicesas much as possible, too.

6.6 Native Plug-ins

A native plug-in is a mechanism for discovering, integrating, and running modules of code in a late binding way whereby the plug-in is implemented in the programming model of a UI technology other then Web Dynpro.

We are already taking this approach of plug-ins inside SAP, where the workbench is Eclipse and the plug-in internals are Web Dynpro. This allows code to be introduced into our VCL models without breaking the modeling approach (see *Figure 5 - Current Architecture of Visual Composer (both GML & VCL) and Web Dynpro* above).

An important feature, if we could achieve it, is to allow other workbenches to be used (Visual Studio, Macromedia…) and other programming models, creating plug-ins that will be used in VCL+ and run encapsulated within the Web Dynpro runtime. In order to enable this feature, the definition of a plug-in must be covered by the meta-model (“code” package) illustrated in *Figure 8 - Draft meta-model for UI domain language*. From this package it must become possible to derive all necessary VCL+ integration aspects, such as the runtime interoperability but also the design time representation and software logistics requirements. The implementation of such a native plug-in is then done in the native workbench of a certain UI technology. For each technology that needs to be enabled, SAP will have to provide native services and even tools that support the VCL+ extension mechanism.

A VCL+ model that uses such an extension will become dependent from the client platform. Hence this is a limited solution for the general SAP strategy, but a solution that enables exceptional cases like ISVs wishing to enhance the SAP solution. The real big advantages will be the enabling of UI technologies for ISVs and customers. The plug-in's do not break the modeling domain of VCL+ and can run in a tightly coupled manner for arbitrary content types.

We have first hand experience with such an approach; Adobe integration is an example that has been actually productized. Also an initial project to check feasibility of such an idea with MS XAML has been done, with some positive results. The plug-ins in that case were MS Avalon plug-ins (in XAML) running within a Web Dynpro environment. A further prototype is a Macromedia Flex integration that follows the suggested approach. The general architecture of the XAML prototype is as follows.

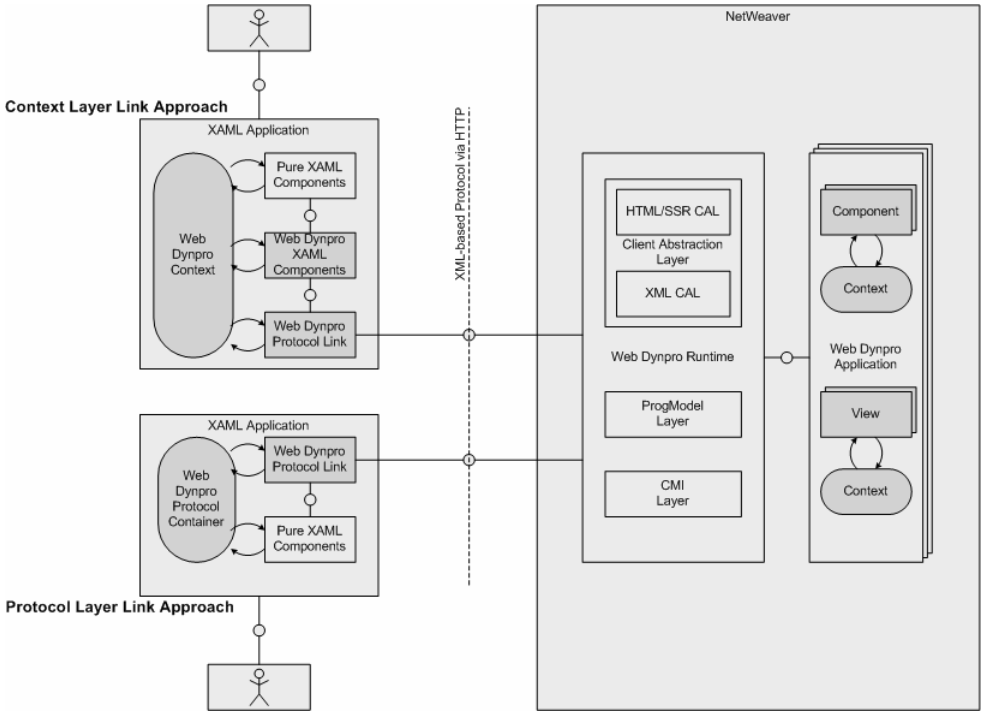

Figure 9 - Prototype of a XAML integration into Web Dynpro

The key architectural idea is to utilize the separation of data, meta data, and events from the actual layout for native plug-ins. A standardized access mechanism for data, meta data and events can be used to communicate with an arbitrary UI technology in a manner that is similar to a service interface with the decisive advantage of utilizing all of the Web Dynpro UI services. Additionally, bridging components are provided to allow the plug-in to interoperate with Web Dynpro.

7 Clients

The Web Dynpro Client Abstraction Layer (CAL) abstracts a concrete client technology and is used to translate the Web Dynpro UI composition. This is the enabling factor for accommodating different client UI technologies. Hence the Web Dynpro model as the VCL/GML model remains independent from a specific client UI technology. It is important to mention the fact that this does not necessarily mean that a single model can run on all different kind of clients. E.g. different form factors and hence a different division of the screen real estate cannot be abstracted and typically requires different models. The same applies for models that require a deep interaction with a specific client technology e.g. MS office integration in project Mendocino. Nevertheless, all of these scenarios could be covered in future using the same programming paradigm.

Depending on the capabilities of the client UI technology, Web Dynpro CAL can either render a complete UI on the server side merging data, layout and meta data or render them separately elsewhere.

7.1 Web Dynpro HTML Client

The most commonly used client for Web Dynpro is the Web Dynpro HTML client. This client implementation offers complete coverage of all Web Dynpro UI elements. It is based on UR and offers full Section 508 (accessibility) compliance. It can be easily embedded into the SAP portal and branding. Using the ACF (active component framework) it is even possible to embed active components like Adobe, MS Office, etc. The HTML client supports IE 6 and Firefox 1.0x. The Web Dynpro HTML client is representative of server side rendering.

7.2 Web Dynpro Windows and Java clients

With NW2004s, SAP ships the Web Dynpro Client for native Windows. This client offers complete coverage of all Web Dynpro UI elements. It can be integrated into the SAP portal. Currently the main advantage is a remarkable increased performance compared to the HTML browser based client.

The Windows and Java clients are sitting on top of a protocol called SCXML. This protocol is an XML language that strictly distinguishes between data, layout and meta data and opens up all kind of optimization. This protocol could even be published in order to enable others to implement a client for Web Dynpro in any arbitrary technology.

Currently it is also under discussion, how these clients could be extended to a smart client in order to create a much richer and more interactive user experience. This includes better usage of the client CPU by running parts of the UI interaction logic on the client as well as better integration of desktop features (drag & drop, clipboard, file system…) and office applications into the client.

7.3 Smart Client

Building effective applications – ones that are rich enough for the most demanding user needs - requires circumventing the shortcomings inherent in static Web browser delivery. Many companies are turning to RIA's (Rich Internet applications) to achieve that goal. A smart client is a client running such a rich internet application. The smart client utilizes as much as possible from the client CPU, operating system and environment. For that purpose all or a large amount of the UI logic resides on the client. Client typical features like drag & drop, clipboard support, local printing, office integration and desktop integration are supported by a smart client.

The previous points may also be realized by a traditional desktop client. However, a smart client goes beyond traditional clients because it uses state of the art, seamless download technologies that do not require any installation. It does not create any additional TCO compared to a zero footprint browser application.

In some discussions of Smart Clients, the full "offline" or "occasionally connected" scenarios are also discussed. We omit this in our paper, as this is not purely a UI technology issue. However the topic will come up and should be discussed in conjunction with the general topic of off-lining services and business objects.

At SAP, we desire that most of the UI logic is modeled in platform independent manner and interpreted by a smart client or, for certain cases, generated to the smart client specific environment.

Some more specific requirements from Smart Clients include the following.

- It should be possible to *invoke services* (ESI services, Web Services, WD SC-XML, and so on) from the smart client without re-rendering the UI. Data service results should be returned asynchronously and redrawn using data binding.
- The Smart Client environment should be able to *dynamically calculate simple expressions* (the platform-neutral expressions of VCL+ that we mentioned earlier) on the client. These expressions do not need to trigger a round trip, though they are allowed to do so if server processing is needed (e.g. a service must be called).
- *Data binding* is enabled and heavily used; properties and fields can be data bound meaning that they are dynamically re-evaluated at runtime whenever their data changes. Data binding can be made either directly to single fields or to complex expressions involving any number of fields. .
- *UI-only actions* should be supported (actions that do not require communication with the server; e.g., prompt, export to clipboard, print, portal events / navigation, and so on) and such actions should be carried out on the client-side.
- Additional*, non declarative UI logic* may be implemented in the smart client environment. For Flex this is ActionScript, for Avalon it is a .NET language. Hence, it is essential to maintain the dependency of this code to a certain platform in the model in order to track platform dependencies of models. A smart client must define its runtime environment in terms of KPI's (required bandwidth, required client resources like CPU, memory, etc…).
- *Client-side caching* is needed. The main advantage of a smart client is an optimal integration into the client environment and high interactivity. Since smartness and high interactivity in many cases requires also much meta data and data on the client, an efficient client side caching mechanism is essential.
- *Data operators* (e.g. joins, filters, and so on) may be used for adapting service results to UI needs. Depending on type of operator and size of dataset, the computation of the data operator should be carried out either on the server or on the client. In some cases (e.g., simulation views) operators must be computed dynamically on the client side.

7.3.1 Web Dynpro and Smart Clients

Web Dynpro is well suited for building smart clients, as it is meta data rich and has good support for data binding on both client and server. Most of the requirements mentioned in the previous section are possible. However, a few of the features mentioned in the previous section need to be completed (e.g. the ability to dynamically calculate expressions on the client).

A Web Dynpro way to support a smart client is through CAL (Client Abstraction Layer). This allows the same code base to drive Smart Clients of various technologies (e.g. both Avalon and Flex).

7.3.2 Flex as a Web Dynpro Smart Client

Flex is a prime candidate for a smart Web Dynpro client that executes UI logic on the client through Flex action scripts. Nevertheless, client side UI logic can be executed in a script like language called "Action Script", asynchronous data retrieval and a compelling UI rendering belongs to the outstanding capabilities of Flex.

7.3.3 Avalon as a Web Dynpro Smart Client

In the same way as Macromedia Flex, the upcoming MS Avalon technology can be supported through the Web Dynpro CAL layer. It is to be expected that the competitive situation in the small and midsize market will confront SAP with MS and others that rely on MS technology. MS Avalon and XAML offer a much wider range of capabilities compared to that of a Web Browser and furthermore, there are many indicators that create the impression that MS prefers office applications as the environment for business application UI's. SAP must offer an attractive solution for Avalon based on XAML. Whether adopting XAML as a UI technology or creating an offering for application level .NET programming is currently not clear.

8 Comprehensive UI Strategy

In the coming months the XUPA team will create a comprehensive UI strategy document that contains a more detailed and elaborate description of how SAP's UI should emerge in the next 3 to 5 years. Besides topics discussed here this strategy will also discuss:

- A recommendation how to evolve Web Dynpro, VCL and GML to VCL+. This will include what should be covered by VCL+ and XGL+ and how to integrate different technologies into that model.
- A recommendation on SAP's client and smart client offering. This will not only cover the different technologies but also different devices, different form factors and how to deal with them in the UI models. Also covered is the offline and occasionally connected client.
- A view on the overall development infrastructure for application development teams and the path how to migrate from the current UI models and implementation to VCL+.
- A recommendation on how to deal with old UI technologies used within SAP products. This will also include recommendations how to improve VCL and Web Dynpro in order to push adoption of the new UI technology.
- A recommendation on how Work Center and Control Center can be moved to VCL+.
- A recommendation how to deal with the exceptions where we generate directly into other UI technologies and how to deal with the consequence of having some platform dependent models.
- A discussion of the UI for "Guided Procedures"
- A discussion on the how the SAP portal needs to evolve.
- A discussion of the UI for composite applications.
- The strategy for the UI design time including "model to code" and "code to model"
- A discussion of adaptability and extensibility of UIs and how this fits into other adaptability and customizing approaches at SAP.
- Topics relating to usability and simplicity of UI technologies
- The influence of the UI strategy on TCO
- The B1S UI
- A Competitive Analyses

9 Conclusion / Summary / Outlook

The central message of this position paper is around the interaction between Visual Composer (**VC**) and Web Dynpro (**WD**) which must be seen as "**Web Dynpro provides the components and the runtime infrastructure for building coded, configurable building blocks, while VC is the environment for freely modeling front end applications using these building blocks**".

We recommend enabling the creation of plug-in enhancements to VC/WD applications in non-SAP programming models; this would mean that even though the overall application is VC/WD, plug-ins using e.g. MS XAML or Macromedia Flex would be possible.

Specific areas do exist (Mendocino, SBS, and so on) where the UI programming model is required to be different (MS XAML, Macromedia MXML, …) yet we still wish to use VC and the UI Services provided by Web Dynpro.

We are therefore able to **propose** several activities for further unification of our architecture.

- Continue support and encourage the already existing "consolidation" project that brings GML and VCL together into "VCL+", supports "XGL+", and creates a Web Dynpro interpreter for XGL+.
- Full enablement and migration of Web Dynpro Patterns to VCL+ and XGL+ (sometimes called "super patterns" or "design time patterns")
- Making coding Web Dynpro **plug-ins** easier, when within a VC model, a high priority.
- Develop guidelines internally at SAP on how to model and how to mix **plug-ins** into models.
- Alignment and consolidation of the UI **elements and controls**.
- Unification of the overlapping portions in the **meta-models** of WD and VCL.
- Evolution of the **programming model** to support unification.
- Making VC, VCL+, XGL+ available and exposing general and UI services outside of the UI technology stack, to support non-UI needs in SAP. In addition support areas (Mendocino, SBS, and so on) where the UI programming model is required to be non-SAP (e.g. MS XAML, Macromedia MXML, …)

10 Further Reading (optional)

SAP NetWeaver Architecture - Mid-Term Vision by Albert Becker, Lambert Boskamp, Rainer Brendle, Wolfgang Degenhardt (information owner), Josef Dietl, Holger Gockel, Andreas Köppel, Hartmut Körner, Prasad Kompalli, Holger Meinert, Gilad Parann-Nissany, Adolf Pleyer, Andrew Ross, Stefan Sigg, Janusz Smilek *Version 1 dated March 7, 2005*

11 Glossary

Term	Definition
UI Element	A UI building block that is not yet task-specific but occurs over and over. The most basic units of re-use in the UI includes buttons, tables, trees. Used to build UI pattern elements
UI control	Same as above
UI service	A generic executable code that is implicit meta driven or with some input from application e..g. value help, error message, accessability, personalization etc.
UI pattern	A description of a standard solution for an interaction problem in a context of a specific user task, such as searching for and editing a business object, and which is implemented by a user interface building block.
Plug-In	A plugin (or plug-in) is a computer program that can, or must, interact with another program to provide a certain, usually very specific, function.
Macromedia Flex	Macromedia Flex offers an XML- based scripting language that compiles Flash applets on the fly and can call on remote Web Services, EJBs and other Java objects to create a rich client.
XAML	XAML (short for eXtensible Application Markup Language, and pronounced "Zammel") is the user interface definition language for the next version of Microsoft Windows, code named Windows Longhorn.
GML	A “visual language” for describing the models has been created in VC; it involves a visible graphical representation of the various elements that appear in the model (e.g. a service, a UI form, and so on). VC ’05 (“VC Patterns”), the VC ’04 (“VC Freestyle”) release calls it GML (Graphical Modeling Language)
VCL	A “visual language” for describing the models has been created in VC; it involves a visible graphical representation of the various elements that appear in the model (e.g. a service, a UI form, and so on). VC ’05 (“VC Patterns”) release calls it VCL (Visual Composition Language).
XGL	XGraph Language (XGL) representation that contains a blueprint of the execution semantics and is optimized for runtime.

