Exploring the Many Architectures of a Very Large Component-based Software

Jean-Marie Favre, R. Sanlaville, J. Estublier

Adele Team, Laboratoire LSR-IMAG
University of Grenoble, France
http://www-adele.imag.fr/~jmfavre

Abstract

This paper describes the OMVT, an exploration tool specifically designed to explore the architecture of CATIA, a multi-million LOC software based on a component technology. This software is developed concurrently by more than 1000 software engineers. It can be dynamically extended and changed by third parties without any recompilation. Many techniques are used to deal with these very strong requirements on software architecture. Architectural concepts are however implicit in the source code or are represented by means of very low level techniques. The OMVT enables to cope with this problem by providing stakeholders the architectural views they need at the appropriate level of abstraction. Though the views presented are specific to Dassault Systèmes, the meta-model driven approach we took can be applied in other contexts.

1. Architecture at Dassault Systèmes

Dassault Systèmes (DS) is the CAD/CAM world leader and is one of the largest software editors in Europe. CATIA is one of its best-known software product. In the mid 90s, when DS initiated the development of CATIA V5, it was rapidly discovered that OO technology has serious limitations when developing very large scale software. C++ alone did not satisfy DS' strong requirements. As a result DS developed a proprietary component technology called the OM. DS is indeed with Microsoft one of the pioneers in componen-based software engineering.

All concepts provided by the OM, are implemented in terms of C++ entities or by means of other low level techniques. The mapping from architectural concepts to implementation is not one to one. For instance the realization of a single OM entity can produce a very large set of C++ entities. Moreover, for a given conceptual entity there are many implementation choices: to improve performance and address other non-functional requirements, DS designed and tested a wide range of realization techniques. All these techniques allow to build very efficient component-based software. But at the same time developing and maintaining large amount of components is quite difficult (CATIA is based on more than 60000 classes and about 8000 OM components). A major issue was that the architectural level was implicit and that software engineers had no tool to visualize the component they develop. The problem was even more accute since many people in different teams and sometimes in different companies collaborated to the development of a single component by adding extensions. What was missing was an architectural viewpoint suited to this specific component technology. The collaboration between the ADELE team and Dassault Systèmes lasted 7 years and during this period various other kinds of architecture were identified as well. This includes for instance the physical architecture, but also the collaborative architecture and the commercial architecture[1]. It is now widely recognized that the notion of software architecture greatly depends on the perspective considered and the stakeholders needs [2]. To formalize the various stakeholder needs and the many architectural concepts used within DS, we took a meta-model driven approach for architectural reconstruction [3]. According to the terminology defined by the IEEE standard [2] each *viewpoint* is a subset of the global meta-model, while each *view* is an instance of a viewpoint for a particular portion of the software considered.

2. OMVT: a specific exploration tool

In parallel with the definition of the architectural meta-model, we provided a graphical notation for the logical architecture. This notation was cautiously designed to be compatible with existing habits used internally during informal communications [4]. This notation provided the syntax to express architetural views on CATIA. Thanks to a reverse engineering process, architectural views are automatically extracted from CATIA source code as well as many other sources of information.

The need to use many sources of information in recovering the architecture of component-based software is also described in [5]. While the tool described in [5] was applied on a toy example, the OMVT was successfully applied on the whole CATIA software which is made of more than 6 millions LOC. Though the OMVT was initially developped to explore the logical architecture and focused on DS proprietary component technology, support for many other architectural viewpoints were later added to this tool in order to explore the many other facets of software architecture at Dassault Systèmes. In total 27 viewpoints were defined to support the specific needs of various stakeholders. Some example of views derived from some viewpoints are represented in the figure below.

3. Scenario

Describing the whole features of the OMVT is impossible here, in particular because some features relie on proprietary architectural concepts such as *medias*, *solutions*, or *frameworks*. The figure below represents a typical OM component displaying some OM interfaces (1). This component is based on one "base implementation" (2) and other "extensions" (rectangles). Contrarily with the COM technology, component inheritance is supported as depicted on the top of window (1). The view depicted in window (3) provides more information about the implementation technique actually used to realize the component. For instance it is possible to distinguish extensions from single to multiple instanciation (3). As shown in (5) contextual pull-down menus are available for each entity displayed to further continue the exploration. A "troubleshooter" viewpoint was implemented. to automatically detect anti-patterns that could potentially lead to errors. In window (6) icons and colors indicate possible inconsistencies. A single click on the warning icon (7) opens window (8). Similarly a click on (9) displays window (10). These windows display only the subgraph of the component graph that reveal the existence of the anti-pattern. Inheritance relationships are shrinked to improve the reading of the figures and increase the usability for the stakeholders directly interested in correcting the error. Window (11) depicts a complement of information that improve the quality of the diagnosis. Note that all the views mentionned above correspond to the logical architecture. The stakeholder can switch very easily to alternative viewpoints on other kinds of architecture. For instance, a view on the physical architecture for the same component is displayed in window (12). This view shows that this simple component is actually implemented by more than 80 CPP classes spread in more 20 DLLs contained in more than one dozen of distinct frameworks.

4. References

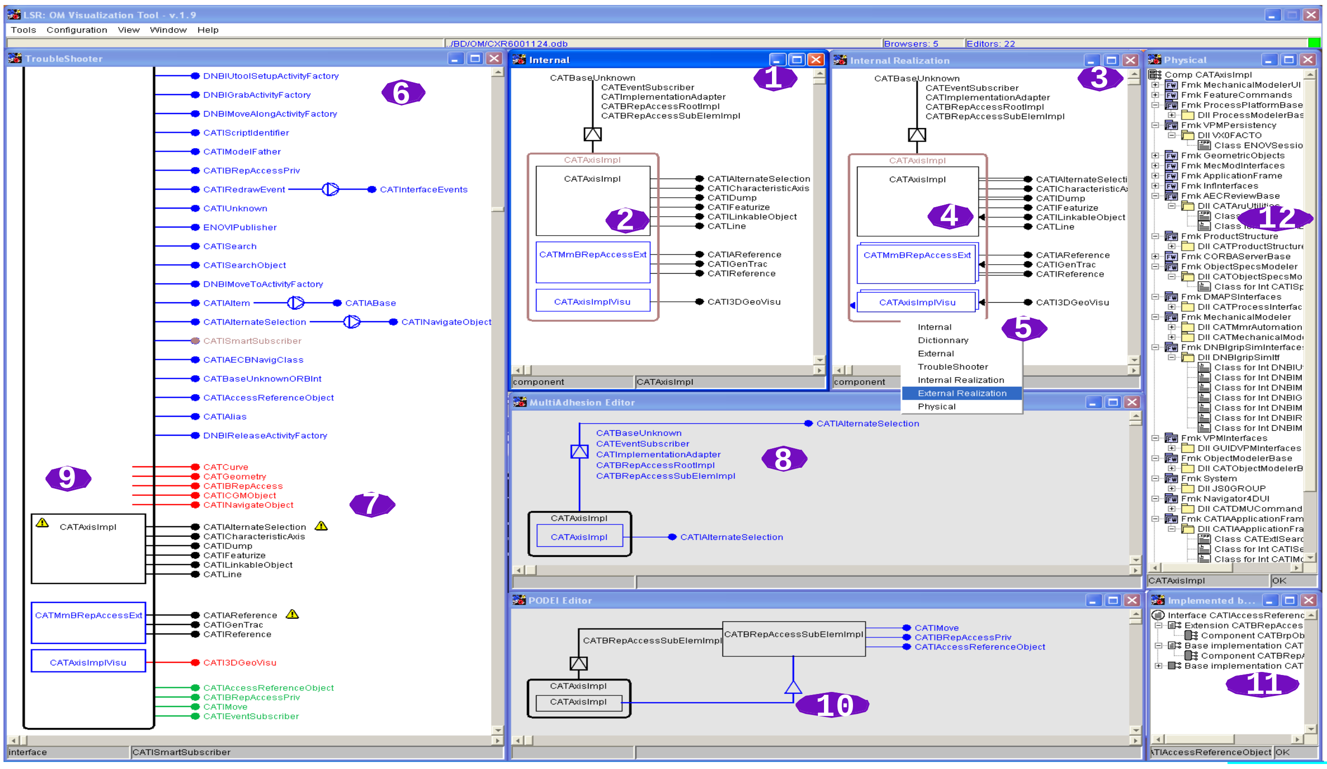
[1] J. Estublier, J.M. Favre, Y. Ledru, R. Sanlaville, "Architectural facts in the concurrent development of a Very Large Software", submitted to IEEE Software

[2] IEEE Architecture Working Group. "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems" . IEEE Std 1471-2000, October 2000.

[3] J.M. Favre, "Meta-Model Driven Reverse Engineering", submitted to WCRE 2003

[4] R. Sanlaville, "Software Architecture: An Industrial Case Study within Dassault Systèmes", PhD dissertation in french, Univeristy of Grenoble, 2002

[5] M. Pinzger, J. Oberleitner, H. Gall, "Analyzing and Understanding Architectural Characteristics of COM+ Components", IWPC 2003



LSR: OM Visualization Tool - v.1.9
Tools Configuration View Window Help
/BD/OM/CXR6001124.odb
Browsers: 5
Editors: 22
TroubleShooter
DNBIUtoolSetupActivityFactory
DNBIGrabActivityFactory
DNBIMoveAlongActivityFactory
CATIScriptIdentifier
CATIModelFather
CATIBRepAccessPriv
CATIRedrawEvent
CATIInterfaceEvents
CATIUnknown
ENOVIPublisher
CATISearch
CATISearchObject
DNBIMoveToActivityFactory
CATIAItem
CATIABase
CATIAlternateSelection
CATINavigateObject
CATISmartSubscriber
CATIAECBNavigClass
CATBaseUnknownORBInt
CATIAccessReferenceObject
CATIAlias
DNBIReleaseActivityFactory
CATICurve
CATGeometry
CATIBRepAccess
CATICGMObject
CATINavigateObject
CATAxisImpl
CATIAlternateSelection
CATICharacteristicAxis
CATIDump
CATIFeaturize
CATILinkableObject
CATILine
CATMmBRepAccessExt
CATIAReference
CATIGenTrac
CATIReference
CATAxisImplVisu
CATI3DGeoVisu
CATIAccessReferenceObject
CATIBRepAccessPriv
CATIMove
CATIEventSubscriber
Interface
CATISmartSubscriber
6
7
9
Internal
CATBaseUnknown
CATEventSubscriber
CATImplementationAdapter
CATBRepAccessRootImpl
CATBRepAccessSubElemImpl
CATAxisImpl
CATMmBRepAccessExt
CATAxisImplVisu
component
1
2
Internal Realization
Internal
Dictionnary
External
TroubleShooter
Internal Realization
External Realization
Physical
3
4
5
MultiAdhesion Editor
8
PODEI Editor
CATBRepAccessSubElemImpl
CATIMove
CATIBRepAccessPriv
CATIAccessReferenceObject
10
Physical
Comp CATAxisImpl
Fmk MechanicalModelerUI
Fmk FeatureCommands
Fmk ProcessPlatformBase
Dll ProcessModelerBas
Fmk VPMPersistency
Dll VX0FACTO
Class ENOVSessio
Fmk GeometricObjects
Fmk MecModInterfaces
Fmk ApplicationFrame
Fmk InfInterfaces
Fmk AECReviewBase
Dll CATAruUtilities
Fmk ProductStructure
Dll CATProductStructure
Fmk CORBAServerBase
Fmk ObjectSpecsModeler
Dll CATObjectSpecsMo
Fmk DMAPSInterfaces
Dll CATProcessInterfac
Fmk MechanicalModeler
Dll CATMmrAutomation
Dll CATMechanicalMode
Fmk DNBIgripSimInterfaces
Dll DNBIgripSimItf
Fmk VPMInterfaces
Dll GUIDVPMInterfaces
Fmk ObjectModelerBase
Dll CATObjectModelerB
Fmk System
Dll JS0GROUP
Fmk Navigator4DUI
Dll CATDMUCommand
Fmk CATIAApplicationFram
Dll CATIAApplicationFra
Class CATExtISearc
CATAxisImpl
OK
12
Implemented b...
Interface CATIAccessReferenc
Extension CATBRepAcces
Component CATBrpOb
Base implementation CAT
Component CATBRep
Base implementation CAT
11