THE POSSIBILITIES ARE INFINITE

FUJITSU

Architecture Independence and Code Generation Produce Real Cost Savings

Model Driven Architecture™

Project Context

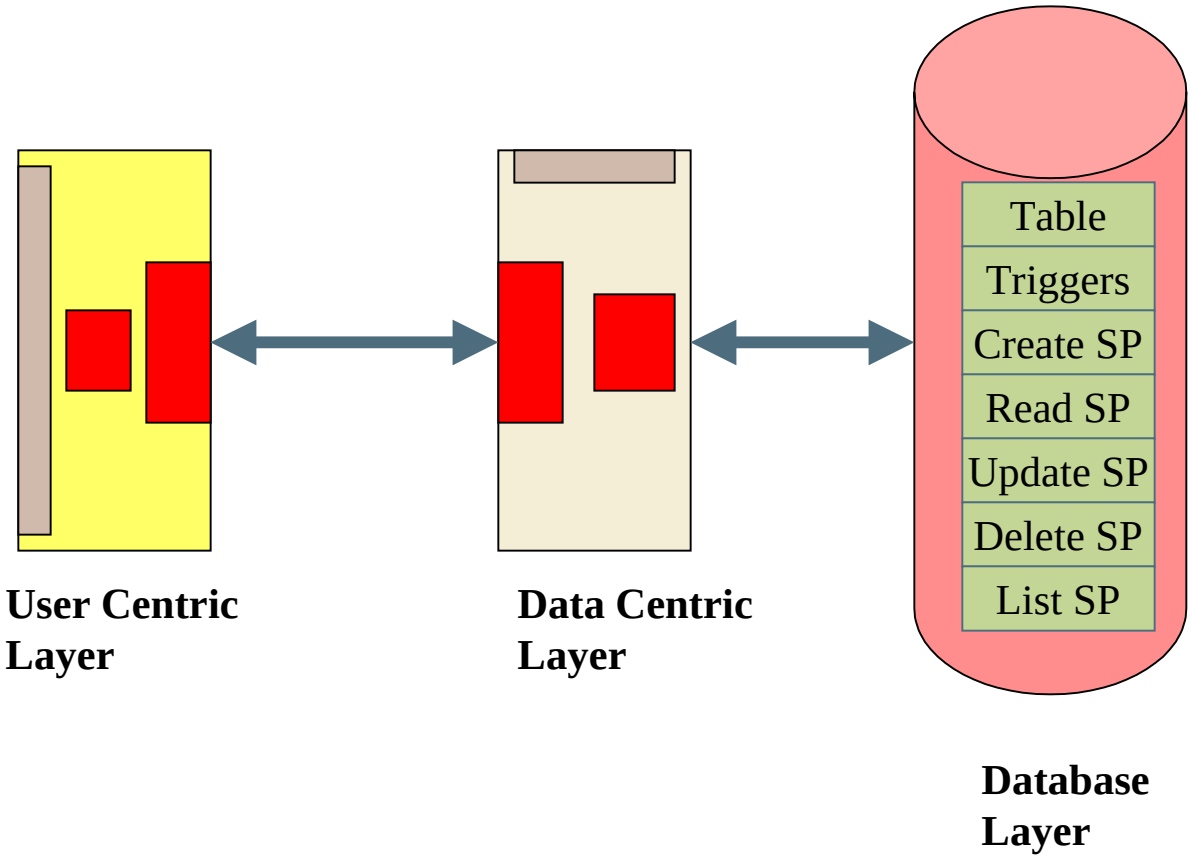
- This case study of applying Model Driven Architecture is from one of Fujitsu's Aerospace manufacturing client's projects. Fujitsu managed the development phase of this project, but had no control over the Preliminary Analysis and Architecture phases. Our influence over the direction of the project was limited. None-the-less we were able to convince the client and project team to use MDA in a limited form.
- MDA on this project was implemented on only half of the objects and was limited to first code generation only. Once code was generated, subsequent maintenance was done manually by the developers.

Technical Context

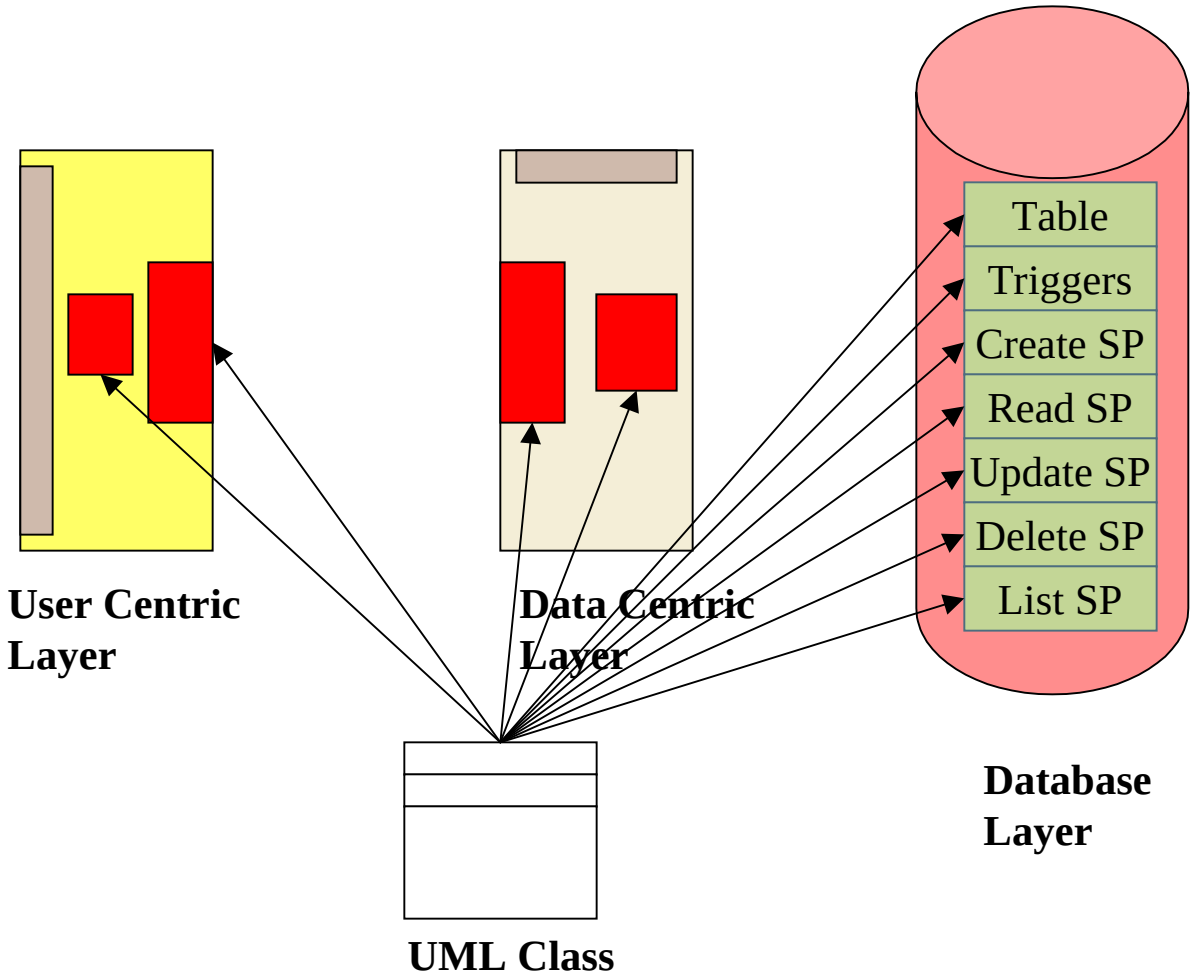
The project was implemented as a three tier client-server application.

- **Microsoft .NET**
- **Visual Basic .NET**
- **SQL Server**
- **Microsoft Word**
- **Panagon Integrated Document Management from FileNET.**

Architecture



Architecture



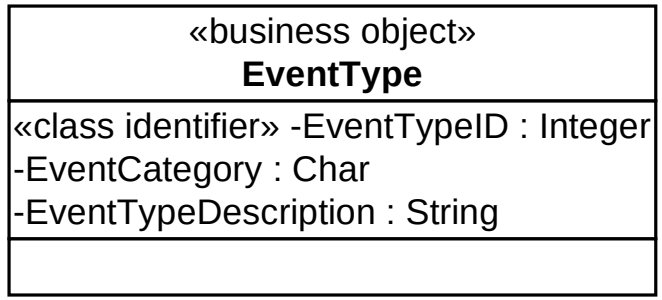
Initial Manual Coding

- **Code Generation was not used from the project start**
  - **22000 lines of code were generated manually before we started using code generation.**
    - **Synchronization issues between all eleven architecture components caused the majority of quality control issues.**

Code Generation – Codagen Architect 3.0

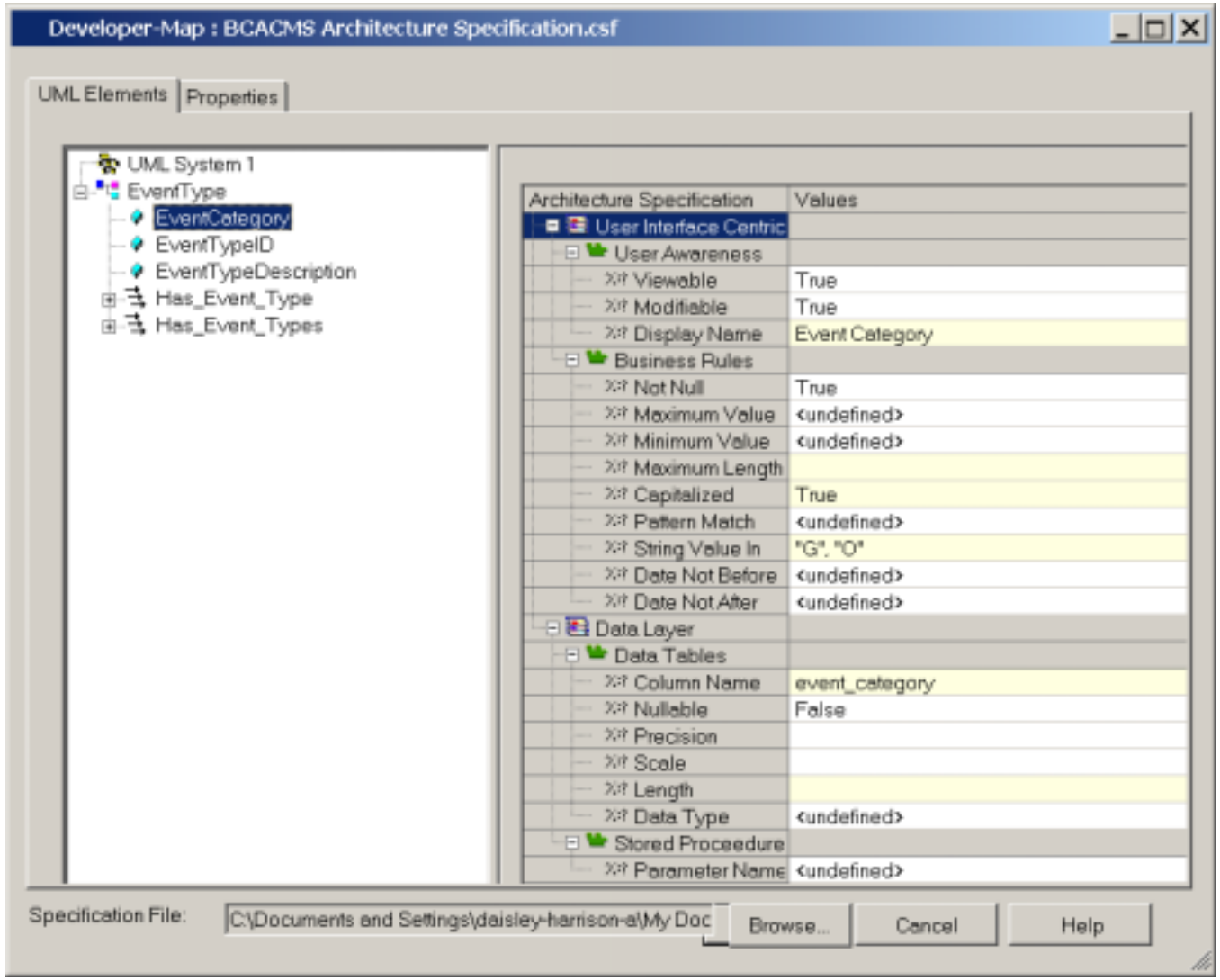
- **Microsoft Visio for Enterprise Architects**
  - **UML Modeling**
- **Codagen Architect 3.0**
  - **Architecture Specification allowed us to isolate UML logical models from the architecture**
  - **Templates allowed us to generate code in the client's architecture.**
    - **Code created in this manner was indistinguishable from the code created during architectural prototyping.**

Code Generation

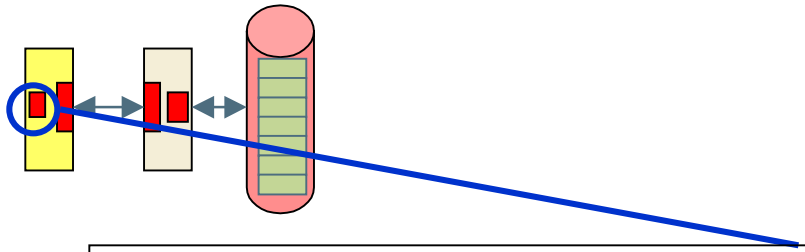


Code Generation

Codagen's Architecture Specification File



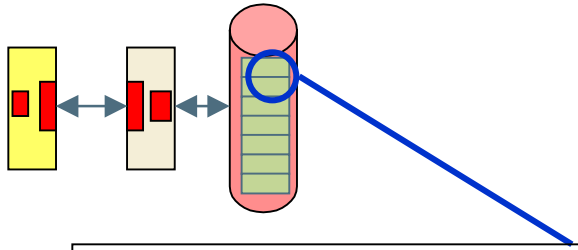
Code Generation



```
Public Class ucEventType
Inherits BaseBusiness


Private mchrEventCategory As Char
Private mintEventTypeID As Integer
Private msEventTypeMain As structEventType
Private mstrEventTypeDescription As String
<NotNull(),MaxLength(1),StringValueIn("G", "O"),DisplayName("Event Category")>Public Property EventCategory() As Char
   Get
      Return mchrEventCategory
   End Get
   Set(ByVal Value As Char)
       If mchrEventCategory <> Value Then
           mchrEventCategory = Value
           If Not Loading Then
              Validate( "EventCategory" )
               mblnDirty = True
           End If
        End If
    End Set
End Property
End Class
Public Sub Delete(ByRef objPM As ucPMgr)
Public Function GetStructure() As structEventType
Public Sub LoadRecord(ByRef objPM As ucPMgr)
Public Sub Save(ByRef objPM As ucPMgr,  OverWrite As Boolean)
Protected Sub SetupClass(ByVal blnBreak As Boolean)
```

Code Generation



```
CREATE TABLE [event_type] (
                    [event_type_id] int IDENTITY(1,1) NOT NULL,
                    [event_category] char(1) NOT NULL,
                    [event_type_description] nvarchar(30) NOT NULL
                    CONSTRAINT [PK_event_type] PRIMARY KEY NONCLUSTERED
                    (
                                        [event_type_id]
                    ) ON [PRIMARY]
) ON [PRIMARY]
```

Code Generation

- **From a single logical class definition, eleven separate components were generated.**
  - **User centric object – with all field level business rule validations**
  - **Data structure to communicate data between user centric and data centric layers**
  - **Interface definition to allow user centric objects access to data centric objects**
  - **Table create script**
  - **Trigger scripts to maintain audit trail information**
  - **Create stored procedure**
  - **Read stored procedure**
  - **Update stored procedure**
  - **Delete stored procedure**
  - **List stored procedure**

Issues With Code Generation

- **Separate data model and object model.**
  - **Maintaining the synchronization between models was time-consuming.**
    - **Synchronization was manual and error prone.**
  - **We used Erwin to maintain our table and element definitions.**
    - **Could not export the descriptions from Erwin into our object model and subsequently into code.**
- **Data Table and Element names were not the same as our Class and Attribute names.**
  - **Architectural specification used by the code generation tool recorded table-class and element-attribute pairings.**
    - **Spelling mistakes were common here and time consuming to fix.**

Code Generation Results

- Code was generated directly from Logical Model
  - Physical model was not required.
    - Physical model could be re-engineered from source code if required.
- Models used for initial code generation only
  - Pattern produced was 1406 lines of code.
  - Approximately 32000 lines of code were generated.
    - 1000 person-effort-days saved in code cutting alone.
    - Unknown effort saved in testing cycle. Since defects are related to lines of code produced, expected savings is 22 times fewer defects in generated code.

Code Generation Results

- **Architecture was not stable**
  - **Architecture templates changed fairly often**
    - **Having the architecture embodied in the templates allowed architectural changes to minimally impact the models**
  - **Architecture changes after initial code generation required direct source code modification**
    - **Once this occurred, models were no longer synchronized with code**

Code Generation Results

- Good understanding of UML is required.
  - Class relationships must be well understood and correctly documented.
- Models required a great deal of attention to details
  - While “Close” is OK for analysis models and generation of paper documentation, models used for application generation MUST be accurate. (This may seem like a draw back, however accurate and up-to-date models are useful over the entire life-cycle).
    - Took time to complete.

Overall Project Observations

- **The use of MDA techniques optimized architecture and code development. Its success brought the spotlight on other issues in the project that now appeared as the bottleneck.**

Overall Project Observations

- **Business Analysts didn't know how to read UML.**
  - **Functional requirements documents were not synchronized with models.**
  - **Business rules embodied in models were not validated by analysts. Paper documents were still produced.**
- **Maintaining the synchronization between documents and models was done manually.**
  - **This process was not successful. Paper documents were consistently out-of-date.**

Overall Project Observations

- Requirements were not stable and traceable
  - The same requirements were often gathered by multiple people in and over several meetings with the users.
  - Requirements were often contradictory.
  - Requirements needed to be validated more than once.
  - While the requirements were documented, the reason behind the them were not.
- Requirements approval did not guaranty user buy in
  - Large requirements documents signed by the user were not always understood.
- Requirement and model synchronization was done manually

Conclusions

Even though Code Generation or (MDA) was introduced late in the process, 1000 developer effort-days , a significant savings in development time was realized.

Future use of MDA techniques on Macroscope based projects could show more significant savings over the entire life-cycle if implemented earlier in the development process.

Additional savings could be achieved if documentation was normalized. I.e. Only one copy of the each requirement was used throughout the project, was traceable, and easily synchronized with models and other deliverables.

THE POSSIBILITIES ARE INFINITE

FUJITSU

Model Driven Architecture™

Presented By: Aaron G. Daisley-Harrison of
Fujitsu Consulting http://www.fujitsu.com
mailto://aaron.daisley-harrison@consulting.fujitsu.com