Adaptive Object Modelling using the .NET Framework

Theo Crous, Theo Danzfuss
Computer Science Department
University of Pretoria
Pretoria 0002
South Africa
{tcrous, tdanzfuss}@cs.up.ac.za

Andreas Liebenberg, Alwyn Moolman
E-Logics (Pty) Ltd
Unit L12 Enterprise Building
Innovation Hub
Pretoria 0002
South Africa
{andreas.liebenberg, alwyn.moolman}@elogics.co.za

ABSTRACT

In an ever-changing business environment, business models and rules have migrated from compiled source code to external metadata. This paradigm better known as adaptive object modelling (AOM) empowers domain experts to take control over application implementations, and allows them to change an application's business model as the business evolves. The problem with the adaptive object modelling approach is that it only caters for an evolving business model and ignores the effects of expanding functional requirements. This paper presents the Expandable Software Infrastructure (ESI), an amalgamation of adaptive object modelling and component-based software development. Unlike other adaptive object modelling implementations where metadata have only been used to describe the data and the executing domain, the ESI takes metadata further and utilizes it to describe the data, domain, behaviour and components - providing us with a truly expandable AOM. We highlight how the relatively complex task of adaptive object modelling can be executed simply and elegantly using the Microsoft .NET Framework and further describe how core .NET technologies such as ADO.NET, .NET Compact Framework, reflection and remoting sculpted the architecture of the ESI. We conclude with the notion of moving towards a standardized, intelligent architecture that executes on multiple platforms.

Keywords
Adaptive Object-Model, Adaptive Systems, Dynamic Object-Model, Reflection, Reflective Systems Meta-Modelling, Meta-Architectures, Metadata, Domain Specific Language, Generative Programming.

1. INTRODUCTION

Business needs have developed beyond the capacity of statically structured systems that are unable or unwilling to adapt to changing business requirements.

These requirements for flexible systems can briefly be described as:

- Runtime configurability
- Adaptability
- Extendibility
- Intuitive configuration

Existing approaches to flexible systems have all excelled in at least one of the above mentioned objectives, but none have successfully adhered to all 4 requirements.

We present the Expandable Software Infrastructure (ESI) developed by E-Logics (Pty) Ltd: an adaptive object modelling system that makes use of various techniques found in configurable and/or flexible systems and component-based software development. The ESI's goal is to realize all 4 requirements through the use of metadata and can be briefly described as a metadata-driven component-based framework.

The main contribution of our work is to make an effective use of the .NET Framework to successfully design and develop a flexible system, the Expandable Software Infrastructure (ESI) that conforms to all four above mentioned requirements. We also demonstrate how the ESI was influenced by the .NET framework and focus on the role of .NET Technologies such as ADO.NET, .NET Compact Framework, reflection and remoting in the ESI.

This paper is structured as follows: Section 2 describes the ESI and gives an overview of the high level architecture and metadata structure and a layered view of the ESI. Section 3 presents an in-depth look at the physical architecture of the ESI and how .NET sculpted the architecture. Section 4 scrutinizes existing approaches to flexible systems while Section 5 details some future work draws conclusions.

2. THE ESI

The Expandable Software Infrastructure (ESI) is both a software component infrastructure and an adaptive object model interpreter. Development of the ESI was driven by various business requirements. These requirements are to:

- Develop changeable systems
- Reduce development time and cost
- Intuitively develop systems
- Develop flexible systems
- Develop vendor independent systems
- Reuse common software components

Essentially the ESI is an interpretive layer wrapped around traditional relational database systems, which allows domain experts to build, configure and deploy systems without the need to rewrite or recompile code. The ESI allows domain experts to concentrate on domain modelling, system configuration and maintenance while software developers concentrate on technical issues.

The ESI owes its flexibility to the extensive use of metadata. Metadata is used to describe the domain model, software components, component variability and behaviour. This implies that most changes in the business environment can be catered for by making changes to metadata. Should the need for new functionalities arise, a component that sufficiently fulfils the requirements must be purchased or developed and then described in the metadata. The component's variability refers to those parameters of the component that will be variable for different domains. It is then the responsibility of a domain expert to populate the variability for the executing domain.

The ESI provides a range of tools to assist users with the tedious task of populating metadata. The most notable of these tools is the ESI management console. The management console provides an UML [13] modelling tool that users can use to describe the domain. The management console also enables users to extend the ESI by describing new components and their variability.

ESI Metadata

The ESI metadata is a self-describing object model that can be divided into three layers, as illustrated in figure 1.

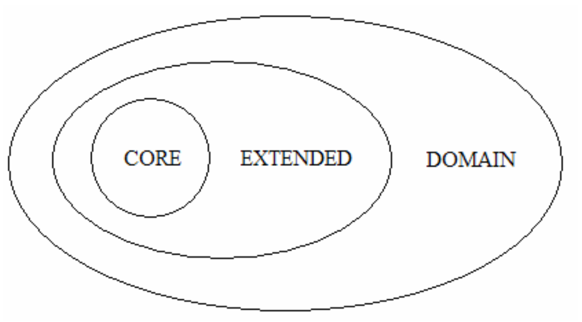

Figure 1. ESI metadata

The Core is used to describe those entities that are critical to the execution of any ESI implementation. Extended metadata are those data that describe the pluggable components while domain metadata is specific for a given implementation.

The core ESI object model is loosely based on design patterns found in classic AOM implementations [2] namely:

- TypeObject Pattern
- Entity and EntityType Pattern
- Property Pattern
- Strategy Pattern

The main differences between the core ESI object model and these classic AOM patterns are that the ESI architecture is split into a functional and a physical level and the ESI metadata is self-describing.

The advantages gained by this architecture are:

- The physical relational database model can differ from the functional object model.

- Technical details stored in the physical layer can be hidden from domain experts, providing a more intuitive model.
- One functional model can easily be migrated to a different physical implementation.
- The core of the ESI can be extended.

Figure 2 presents a graphical representation of the core ESI object model.

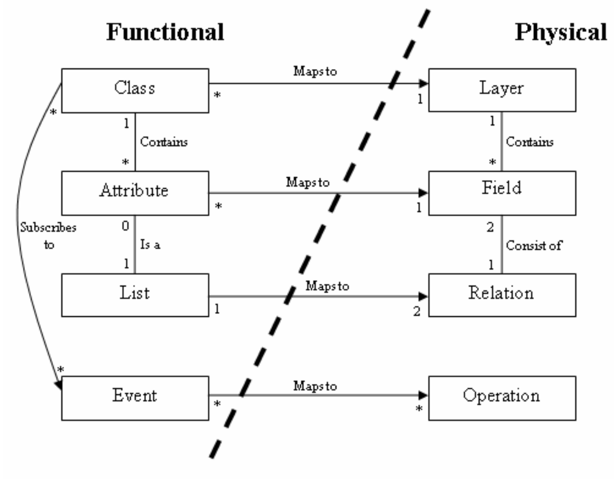

Figure 2. Core ESI Architecture

Changing core metadata results in a new ESI assembly to be built. This assembly is generated by interpreting the stored metadata and generating a new dynamic link library (dll) using the reflection and emit libraries found in .NET. The newly built assembly now forms the base of all ESI systems.

3. ESI AND THE .NET FRAMEWORK

Before the acceptance of component-based frameworks such as J2EE and .NET, implementing a system such as the ESI was an extremely daunting and often impractical task. The following advantages of the .NET Framework [10, 14] made it the perfect candidate for the ESI:

- Low learning curve
- Ease of application deployment and maintenance
- Comprehensive class library
- Managed Code
- Framework support

The decision to choose the .NET framework was not only based on technical merit, but also on non-technical factors such as available resources and user expectations.

The architecture of the ESI was sculpted by the .NET Framework. ADO.NET, remoting, reflection and the .NET Compact Framework were the defining technologies in the structure of the ESI.

ADO.NET and especially datasets enabled the implementation of a data abstraction layer that is vendor-independent and can also treat text-based data stores such as XML and CSV files similar to relational databases. It also provided the ability to create an efficient client-side data cache that reduces network traffic and improves overall system performance.

The .NET remoting infrastructure enables the ESI to execute in a distributed environment over either TCP or HTTP. This permits the ESI to provide rich client interfaces that can retrieve data over the internet and even through firewalls.

.NET Reflection is used to extend the ESI at run time. New types and operations can be added to the ESI by defining them in the metadata. The ESI then uses reflection to load the type at runtime. The ESI also makes use of the .NET emit library to allow for the core ESI to be extended and recompiled by simply altering the metadata.

The .NET Compact Framework allows the ESI to execute on mobile devices such as PDA's. This extends the range of applications that can be executed using the ESI.

The ESI allows multiple deployment scenarios of which the most common is essentially a distributed client-server architecture as highlighted in figure 3.

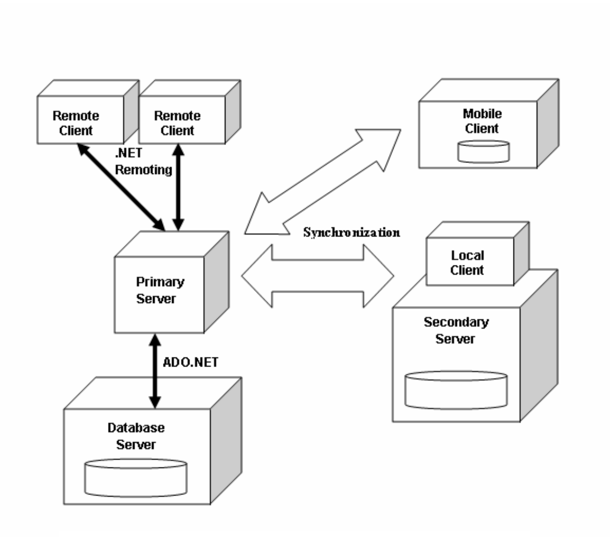

Figure 3. ESI deployment scenario

As seen in Figure 4 the ESI can be broken into nine distinct components. Each of these components leverages the .NET framework to reach its goal.

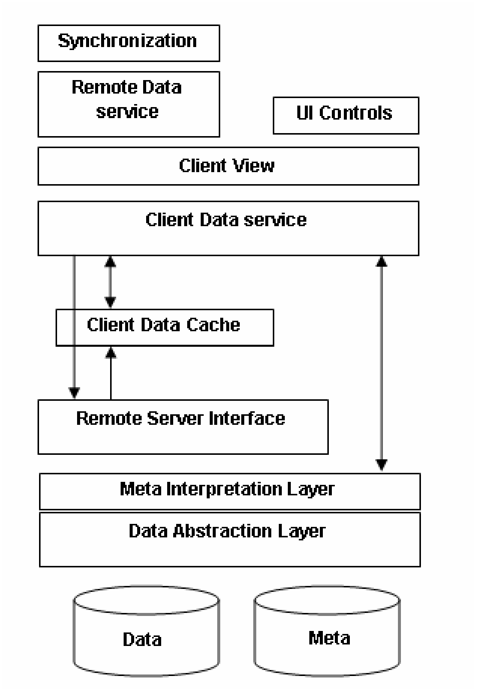

Figure 4. ESI layered architecture

1. The Data Abstraction Layer: The data abstraction layer is responsible for performing basic Create, Read, Update and Delete commands (CRUD) on all the supported data sources.
2. Meta Interpretation Layer: The metadata interpretation layer uses the data abstraction layer to load and save the metadata. Metadata are converted into runtime classes through the reflection API, and all classes built on top of the interpretation layer will use these classes as if they were compiled at design time.
3. Remote Server Interface: The remote server interface is responsible for managing remote client connections and executing all server side operations such as data retrieval. The Remote Server Interface uses the .NET remoting infrastructure to provide basic remoting functions such as object serialization.
4. Client Data Cache: The client data cache reduces network traffic and improves response time, by caching results in a disconnected data set.
5. Client Data Service: The client data service is responsible for executing all client-side operations and managing access to the local data cache.
6. Client View: The client view is a thin wrapper around a .NET dataset that presents users (typically GUI components) with a meta interpreted view on the data. Without a client view user interface components only see a dataset, with the client view user interface components see a collection of metadata objects.
7. Remote Data Service: The remote data service is used by data services to communicate remotely with each other.
8. UI Controls: User interface controls provide users a view on the data and a mechanism to interact with ESI clients. Currently the ESI contains two sets of UI controls; Windows Forms controls and Mobile Controls. Windows Forms controls are extensions to .NET provided controls and allow for ESI-specific functionalities. Mobile controls are UI controls that execute on the .NET Compact Framework and often implement a subset of the functionalities provided by the Windows Forms version of the controls.
9. Synchronization: Synchronization is used to keep secondary and mobile servers in sync with the primary ESI server.

4. COMPARISON WITH EXISTING APROACHES

We categorize existing flexible system approaches into the following categories:

- Configurable Systems
- Adaptive Object Modelling
- Component-based Software Development

Configurable Systems

A configurable system extends the traditional notion of a system by introducing a fixed set of parameters external to the system. These parameters can be modified to alter some runtime attributes or properties of a system. The Gandiva software development system [11] can be seen as an example of a configurable system.

Configurable systems are limited by a fixed set of parameters which are defined at compile time. Therefore the dimensions of configurability are fixed and the scope for adapting is limited.

The ESI relates to configurable systems in that it allows users to configure the system using external attributes. ESI differs from configurable systems by allowing the definition of variability in metadata – enabling the extension of configurable parameters.

Adaptive Object Modelling

An adaptive object model (AOM) [14] is an object model where the domain representation is interpreted at runtime and can be altered or changed with immediate effect [1]. The adaptive model defines mechanisms to describe entities, attributes and relationships, as well as mechanisms to interpret the domain model and execute business rules. Browsersoft's eQ! Foundation [15] is a good example of an industry stable AOM implementation written in Java.

The biggest shortfall of the AOM approach is its internal structures are difficult to extend and maintain. This results in the situation where business requirements can easily be adapted although the functional requirements of the system cannot change easily. We can say AOM systems are adaptive although not adaptable [4, 5].

In addition to using metadata to describe the domain, the ESI also utilizes metadata to define software components, their variability and behavior. This provides the ESI with information that can be used to expand the system on a functional level.

Component-based Software Development

In component-based software development, software products are built on top of component infrastructures [9]. The component infrastructure provides a mechanism for business components to be plugged in and configured to produce a final software product or system. A software system can be extended by plugging in new components or replacing old components. The best known example of a component infrastructure is probably Enterprise Java Beans [16].

Although component infrastructures can be easily extended to provide new functionality, they often requires writing "glue" code to make the new functionalities available.

The ESI provides a pluggable component infrastructure that enables it to expand on a functional level. Instead of having to write code to plug the new components into the framework, the ESI requires the component to be described in metadata.

Table 1 summarizes which objectives are successfully met by each flexible system approach.

	Runtime configurable	Adaptive	Extendible	Intuitive
Configurable Systems	✓	✗	✗	✗
Adaptable Object Modelling	✓	✓	✗	✓
Component-based Software Development	✗	✓	✓	✗
Expandable Software Infrastructure	✓	✓	✓	✓

Table 1. ESI comparison

The ESI is an ideal solution when implementing systems in a constantly changing environment, which requires flexible, configurable, intuitive and adaptable systems.

These systems may span any number of domains, including: asset management, data warehousing, geographical information, decision support and supply chain optimization systems.

5. CONCLUSION AND FUTURE WORK

Developing an adaptive object modelling system is not an easy task. Choosing the correct technology is critical to simplifying this undertaking. The .NET Framework enabled a small team of software developers to conquer this mammoth task within reasonable time. This success can be broadly credited to .NET's low learning curve, the comprehensive class library, ease of deployment, managed code and excellent support.

The ESI overcomes the shortcomings of classic adaptive object modelling systems by introducing aspects from component-based software development. Although the infrastructure is currently being used by a number of industry applications there are a few shortcomings:

- It is limited to the Microsoft Windows and Windows CE platform.
- No web or thin client interface exists.
- Does not conform to standards, therefore it is difficult to extend the ESI with a component that was not developed for the ESI.

- The ESI currently lacks version control and change management.

Apart from the shortcomings mentioned above we would like to see the ESI move towards an intelligent or adaptable architecture [9]. The simplest example of resource adaptation is that of network bandwidth. The system must detect low bandwidths and modify caching settings and request processing accordingly. Another goal for the ESI would be to make it platform independent. With recent developments in the ROTOR and MONO projects, we would like to see the ESI execute on one of these frameworks, thus enabling cross-platform execution.

6. ACKNOWLEDGMENTS

The authors would like to thanks E-Logics (Pty) Ltd for allowing them to work on the ESI and for funding the project. Also a word of thanks to Prof. Judith Bishop, Department Computer Science, University of Pretoria, for her guidance and support.

7. REFERENCES

[1] J.W. Yoder, B. Foote, *Metadata and Active Object-Models.* 1998.

[2] J.W. Yoder et al. *Architecture and Design of Adaptive Object-Models.* ACM SIG-PLAN Notices 36, No.12, pp.50-60, 2001.

[3] R. Reza et al. *Language support for Adaptive Object-Models using Metaclasses.* ESUG Conference, 2004.

[4] A. Dantas et al. *Using Aspects to Make Adaptive Object-Models Adaptive.* ECOOP '04 Workshop on Reflection, AOP, and Meta-Data for software evolution (RAM-SE), pp.9-20, 2004.

[5] K. Lieberherr, *Workshop on Adaptable and Adaptive Software*. Addendum to the Proceedings of the 10th annual OOPSLA, ACM Press, pp.149-154, 1995.

[6] J. van Gurp, J. Bosch, M. Svahnberg. *On the Notion of Variability in Software Product Lines.* In Proceedings of the working IEEE/IFIP conference on Software Architecture (WICSA'01), 2001.

[7] L. Baum, M. Becker. *Generic Components to Foster Reuse.* System Software Research Group, University of Kaiserslautern, 2001.

[8] C.W. Young, M. Young, *Deploying solutions with .NET Enterprise Servers.* ISBN: 0-471-23594-6. Wiley Publishers, 2003.

[9] R. Allen , R. Douence, D. Garlan, *Specifying and Analyzing Dynamic Software Architectures.* Lecture Notes in Computer Science, Vol. 1382, pp.21, 1998.

[10]G. Heineman, W. Councill, *Component-Based software engineering. Putting the pieces together.* ISBN: 0-201-70485-4. Addison Wesley, 2001.

[11] M. Stuart, C. Wheather, C. Mark, *The Design and Implementation of a Framework for Configurable Software*. Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDS '96)

8. WEB REFERENCES

[12] S. Pratschner. *Simplifying Deployment and Solving DLL Hell with the .NET Framework.* http://msdn.microsoft.com, November 2001.

[13] Unified Modelling Language. http://www.uml.org, January 2005.

[14] MetaData and Adaptive Object-Model Pages. http://www.adaptiveobjectmodel.com, January 2005.

[15] The eQ! Foundation. http://www.browsersoft.com/, December 2004

[16] SunMicrosystems. J2EE 1.3 specification. URL:http://java.sun.com/j2ee/download.html, July 2001.