A Component Oriented Domain Architecture for Fish Farming

Svein Hallsteinsen
SINTEF Telecom and Informatics
N-7465 Trondheim, Norway
+47 73593010
svein.hallsteinen@informatics.sintef.no

Øyvind Dragsten
Icon Medialab
Box 2393 Solli N-0201Oslo, Norway
+47 98820387
oyvind.dragsten@iconmedialab.no

Magne Johnsen
Superior Systems
Olav Tryvasons gt. 39-41
N-7011 Trondheim, Norway
magne.johnsen@superior.no

Jan Ove Ofstad
Superior Systems
Olav Tryvasons gt. 39-41
N-7011 Trondheim, Norway
Jan.Ove.Ofstad@superior.no

ABSTRACT

This paper presents experiences with applying the Magma model for component based software engineering for a family of applications supporting fish farming. It focuses on lessons learned from the design and evaluation of a common component based architecture as a means to cope with challenges such a rapidly changing requirements and a dynamic technology environment. It concludes that such an architecture is a good match in this case.

INTRODUCTION

The Magma model [2] is an approach to component based software engineering tailored for developing software for a market. Typical for such development is that requirements varies between different users and evolves rapidly over time. Furthermore the competition pushes for rapid exploitation of advances in underpinning technologies.

The model was developed in a joint effort between the Norwegian association of the software industry (PROFF), SINTEF and a handful pilot users.

In this paper we present experiences with the design of a component oriented domain architecture by Superior Systems, which is a small Norwegian company developing software products for the fish farming industry and that was one of the pilot users in the Magma project.

This effort was one step in in a strategic move towards full adoption of the Magma model for all software development in Superior Systems.

BUSINESS PROBLEM

Fish farming is a rapidly growing business area. In a relatively short time it has evolved from a simple craft based production to a knowledge intensive industrialized production with rapidly growing needs for software systems that support optimal planning, control and documentation of the production process as well as common business processes like accounting and budgeting.

As a result of this rapid evolution Superior experience a strong market pull for evolution of the software products they supply, both in terms of demands for new functionality and in terms of demands for distributed operation and internet based access to the system.

In addition the Superior products are marketed and sold internationally and has to cope with variations between different countries in language, traditions and governmental regulations.

It was realized that the monolithic architecture of the current system made it difficult and prohibitively expensive to accommodate the pace of evolution of the product that were foreseen as necessary to remain competitive in the marketplace.

This situation was typical of the pilot users of Magma, and both earlier experience from software reuse projects and experiences from other engineering disciplines had convinced us that a component based approach was the most promising approach to achieving the flexibility called for in order to cope with such challenges as described above.

TECHNICAL APPROACH

Figure 1 illustrates the basic assumptions of the Magma Model.

The typical user of the Magma model is a software developing company that specialises in providing software products for particular business domain. A business domain is characterised by business processes that usually involves human actors and manipulate business artifacts and that are common for businesses belonging to that domain, although considerable variation in how the business processes are carried out in detail is the norm.

A domain is supported by an ensemble of software applications provided by software product companies that specialize in applications for that particular problem domain.

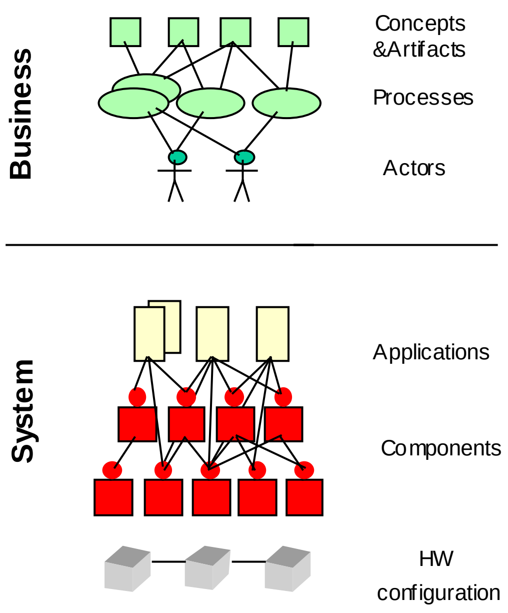

Figure 1 Reference Model

An application is a software product that is used by the actors to carry out business processes, and is typically tailored to support a particular subprocess or a particular type of actor in the business domain.

Applications use components to implement the end user functionality that they offer.

This structure of applications and components and the way they interact constitutes *the domain architecture*.

Normally components are shared in the sense that the services they offer are being used by several applications or other components. Sharing takes place both at the type level (code reuse) and at the instance level (data sharing).

Reference Architecture

The Magma reference architecture is shown in Figure 2. Basically it may be seen as a macro pattern for a component oriented domain architecture.

The reference architecture models a software system as a collection of software components collaborating through carefully designed interfaces and supported by a standardized component infrastructure (labeled Comp. MW in the figure) providing the basic mechanisms for creating and destroying and communicating between components in a distributed hardware environment.

Components are distinguished units that are at the same time units of design, construction, configuration management, substitution and distribution and that conform to and provide the realization of a set of interfaces.

The granularity of components may vary, but normally a component encapsulates a collection of collaborating programming level objects.

The component are classified into five main categories according to their role in the system: i) User Interface, ii) Application, iii) Business, iv) Data storage & retrieval and v) IT services.

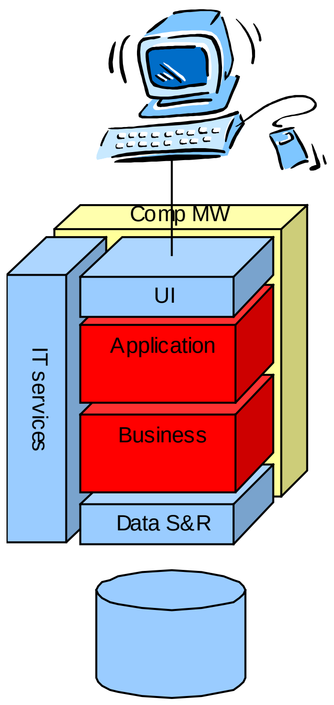

Figure 2 The Magma Reference Architecture

User Interface Components

User Interface components are responsible of presenting information to the user. These are typically graphical components based on appropriate third party user interface component libraries such as Microsoft Foundation Classes (MFC), Java Abstract Windowing Toolkit (Java AWT) and Java Swing. Examples of common user interface components are: windows, forms, menus, buttons, grids and lists.

Application Components

Application components combine services offered by the business components with suitable user interface components into useful applications.

Business Components
Business components implement business oriented concepts and services and typically encapsulate business objects.

The granularity of the business components might vary and is a design issue to be carefully considered. A *home* component offering CRUD (create, read update and delete) as well as collection operations for an entity business object type, is an example of a small grained business component. Examples are home components for entity business object types like customer, product, report and car.

A business component offering more complex, real business services such as a mail server component are large grained business components.

Data Storage and Retrieval components
Data storage and retrieval components realise persistent storage of the business data. The components provide interfaces toward the actual underlying storage mechanisms, such as relational or object oriented databases. Hiding the actual storage mechanism through the usage of defined interfaces gives a great deal of flexibility as to changing database paradigm or product.

IT Service Components
The IT service components provide a diverse set of services needed to build software systems regardless of problem domain. Examples of such services are the CORBA services like for instance transaction service, authorization service or name service.

The component categories also form architectural layers ordered as illustrated in the Figure 2, with the IT-service components forming a "vertical" layer accessible for all the others.

Process Model
The Magma process model splits the software development activity into four main processes

- the *domain engineering process*, which is concerned with understanding the domain and designing a common architecture,
- the *component engineering process* which is concerned with developing reusable components,
- the *application engineering process* which is concerned with developing individual applications,
- the *project process*, which is concerned with initiating, coordinating and controlling engineering processes to achieve short term goals.

In this paper we focus on experiences from the initial domain engineering efforts and in particular the design of a common architecture.

The work products of the domain engineering process and their purpose is illustrated in Figure 3.

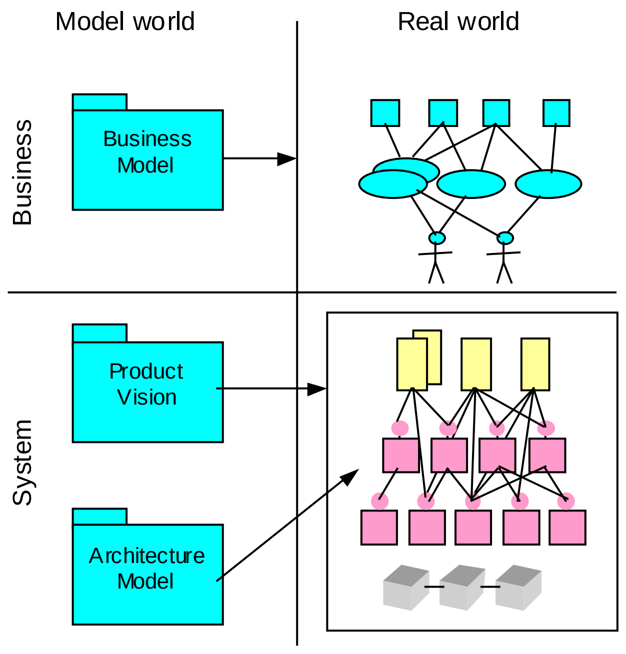

Figure 3 Work Products of the Domain Engineering Process

The purpose of the Business Model is to capture the domain knowledge of the company

The purpose of the product vision is to capture the needs and opportunities for computer based support for the business domain that represents the market of the company. Think of it as the vision of the company of the support for the business processes of their domain that they intend to offer and that they are striving to build into their products.

The purpose of the Architecture Model is to define a common architecture to be obeyed by applications and components in the domain. The Architecture Model consists of:

- The Domain User Interface Model.
- The Domain Component Model.
- The Domain Platform Model.

The *Domain User Interface Model* typically contains UI style-guides and other general guidelines for how to develop user interfaces. Its purpose is to ensure a consistent use of metaphors and idioms and a common look and feel for the applications of the domain, and to establish the foundation for reusable user interface components.

The *Domain Component Model* serves to identify application and business components and the collaboration between them.

The *Domain Platform Model* specifies the technical platform that the business components and applications need to implement the domain oriented concepts they represent and the plug-and-play style configurability that application developers need to easily assemble applications satisfying the varying needs of the domain. It covers the component infrastructure, the IT-services and data storage and retrieval layer of the reference architecture.

Seen from the business components the technical platform is a set of technically oriented services or mechanisms available to them. Seen from inside, the technical platform is a set of components collaborating to offer these services or mechanisms.

The mechanisms that constitute the technical platform are typically not domain specific. Similar mechanisms are needed in most domains. This makes the technical platform a prime area for reuse of third party components.

What vary between domains are the particular requirements to the platform mechanisms, which is derived from the non-functional needs formulated in the Product vision.

SAAM and ATAM

In addition to the guidelines provided by the Magma model the architecture design was also inspired by the Software Architecture Analysis Method (SAAM) [?] and its successor the Architecture Tradeoff Analysis Method (ATAM) [?].

Both SAAM and ATAM are scenario-based design methods that recommend a stepwise process for the design effort. However we found that neither SAAM nor ATAM directly suited our needs, so we ended up with using the following set of steps, which picks elements from both methods:

1. Describe scenarios that the architecture must support. These may be usage scenarios, system management scenarios or system evolution scenarios.
2. Derive requirements for the architecture from the scenarios.
3. Describe the architecture
4. Evaluate the architecture w.r.t. the scenarios

According to the Magma model the scenarios should be derived from the Product vision. At this point we made a shortcut, however, and compensated for the lack of a modeled product vision by involving people that had the product vision in their head in the definition of the scenarios.

The following scenarios were defined:

- Adapt the system to use a different database systems.
- Implement interoperation with a new third party system
- Adapt the system for remote use by many concurrent users
- Rapidly and cheaply adapt an application or develop a new one to cope with evolution of or national variation in business processes.
- Implement support for access by a casual user, which cannot be expected to have installed Superior software products.

From this set of scenarios the following set of requirements was derived:

- Vendor independent database system interface
- Support interoperability with third party systems
- Support multiple concurrent users
- Scalable to many concurrent users
- Support access by casual users
- Support rapid application development

Superior Architecture

The chosen architecture is an instantiation of the Magma reference architecture. This was a natural choice, partly because this effort was part of a strategic move towards adopting the Magma model, but also because the requirements fitted nicely with the properties promised by the Magma reference architecture.

Most of the effort went into evaluating and choosing between available technologies for the platform architecture and to agree on a set of business components for the business component model.

Following is a brief discussion of some of the choices

Stateless server components to achieve scalability

Adaptable user interface components (validity checking,)…

The platform model

Superior was already committed to Microsoft technology, so the COM was a natural choice for the component infrastructure

ADO was chosen as the interface for the data storage and retrieval layer. By putting an ADO interface on top of the current database system and letting all access to the database by the Business components go through this interface, one achieved the sought for database system replacability.

There was some concern about the performance of this solution and some experiments with popular database systems were carried out. This showed that the performance of the chosen solution was adequate.

In the IT-services layer, MTS was chosen for transaction management.

In addition a number of utility components that could easily be extracted from the current applications were identified, for instance user management, access control and a scheduler for recurrent business tasks.

Concerning distribution issues, other experiences within the Magma project indicated that business components had to be designed either for running on the server or in the client.

Achieving true distribution transparent components carries to much of a performance penalty with todays technology. It was decided to go for server based stateless business components.

Stateless components were chosen in order to satisfy the scalability requirement.

The Domain Component Model

Between 10 and 20 business components were identified. Since a proper Business Model and Product Vision had not been made yet, this was mainly based on analysis of existing applications and the vision residing in the heads of a few senior developers at Superior. Some of the business components were pretty obvious, like for instance brood deployment and slaughtering, and fodder management. Others were not so obvious. Therefore we feel that a reiteration of the Domain Component Model, after a more thorough job on domain analysis and scoping has been done, will be necessary.

The result of the evaluation of the architecture with respect to the scenarios is given in tabular form as recommended by SAAM in Table 1. The column labeled Intermediate Arch. shows the evaluation of an alternative architecture providing short term solutions for some of the scenarios and that is likely to be used as an intermediate step in the gradual transition to the new architecture.

Scenario	Superior Arch.	Interm. Arch.
Adapt the system to use a different database systems	yes	yes
Implement interoperation with a new third party system	yes	partly
Adapt the system for remote use by many concurrent users	yes	partly
Rapid application development	yes	no
Access by a casual user	yes	no

Table 1: Evaluation of the architecture

PROJECT DESCRIPTION

The design of the domain architecture was carried out as a master thesis at the Norwegian University of Science and Technology, advised by SINTEF Telecom and Informatics, and in close cooperation with senior developers at Superior Systems. Prior to this the company had done several step towards adopting the Magma model, like for instance acquiring object oriented modeling capabilities and implementing enhancements to their project management process.

BENEFITS AND LESSONS LEARNED

We believe that the described component oriented architecture will provide the flexibility called for by the scenarios shaping it. This was clearly indicated by the evaluation against the scenarios, although it has not yet been demonstrated in practice.

The main problem seems to be to find the time and resources to do the transition to the new architecture. Although ways have been found to do the transition gradually, it is a threat to the success of the move that it may take to long.

Both in this case and in other companies that participated in the Magma project we have experienced that it is difficult to motivate business modeling and modeling of the product vision.

The lack of these models did not appear to be a problem in the platform architecture design. The Superior developers had very clear ideas of the scenarios that were likely to impact it.

Another observation is that they are probably not very domain specific, so many business domains could share the same platform architecture.

As might be expected, the lack of a Business model and a modeled product vision it was difficult to do a good job on identifying business components.

Anyway we believe that a common documented architecture is a very fundamental artefact in component based software engineering and it is felt that having defined and documented such an architecture is a significant step towards a full adoption of the Magma model.

REFERENCES

1. Bass, Len; Clements, Paul; Kazman, Rick: Software Architecture in Practice. *Reading, Massachusetts, Addison-Wesley, 1998*
2. Hallsteinsen, Svein; Solberg, Arnor; Skylstad, Geir; Neple, Tor; Berre, Arne-Jørgen: Magma Software Engineering Handbook. *Revision 1.3, January 2000 http://www.ikt-norge.no/weboffice*
3. Kazman, Rick; Klein, Mark; Barbacci, Mario; Longstaff, Tom; Lipson, Howard; Carrière, Jeromy: The Architecture Tradeoff Analysis Method. *Proceedings of ICECCS ´98, (Monterey, CA), August 1998: 68-78*
4. Kazman, Rick; Barbacci, Mario; Klein, Mark; Carrière, Jeromy; Woods, Steven G.: Experience with Performing Architecture Tradeoff Analysis. *Proceedings of the ´99 International Conference on Software Engineering, (Los Angeles, CA), 1999: 54-63*