# The Magma approach to CBSE

Svein Hallsteinsen and Geir Skylstad,

SINTEF Telecom and Informatics

N-7034 Trondheim

Norway

{svein.hallsteinsen, geir.skylstad}@informatics.sintef.no

### Abstract

In order to meet the need for flexibility and rapid application development facing COTS software manufacturers, high levels of reuse are necessary. This position paper claims that this is only possible by means of CBSE and outlines the Magma approach to CBSE. This approach is based on domain specific architectures, a 3-tier software engineering process model and object-oriented analysis and design.

## Background

Off-the-shelf software vendors experience an increasingly dynamic environment for their products. Both underpinning technology and user demands are in constant movement and challenge the ability of the vendors to evolve their products accordingly.

The Magma project was initiated by the association of the Norwegian software industry (PROFF), to aid their member companies to meet this challenge. To this end a Software Engineering Handbook is being developed. The project is led by PROFF and is carried out as a cooperation between SINTEF Telecom and Informatics and a handful pilot companies. A draft version of the handbook is currently being tested in the pilot companies. The pilot companies are relatively small companies (from a few tens to a few hundreds developers) that are developing and selling software products basically off-the-shelf, although in some cases there is a modest amount of customization involved with each sale.

The Magma Software Engineering Handbook is based on the belief that extensive reuse is necessary to meet the demands for flexibility and rapid response to new needs and new technology enforced by the marketplace and that CBSE is the only way to bring about the necessary level of reuse.

## Position

In the following we briefly outline the Magma approach to CBSE.

### Problem domains

Most software systems are there to support real life processes, for instance business processes or production processes. A set of related real life processes fulfilling a given purpose we call a problem domain. Typically the business idea of an off-the-shelf software company is to provide software solutions for a particular problem domain

### Domain architectures

We envision the structure of the software system that supports a domain as consisting of applications and components.

Applications are the parts that interact with the supported real life process, and typically support a particular process or a particular type of actor.

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

The literature offers several definitions of what a component is. We sympathize with the following definition by Kruchten [BROWN98]:

*A software component is a non-trivial, nearly independent, and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces.*

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).

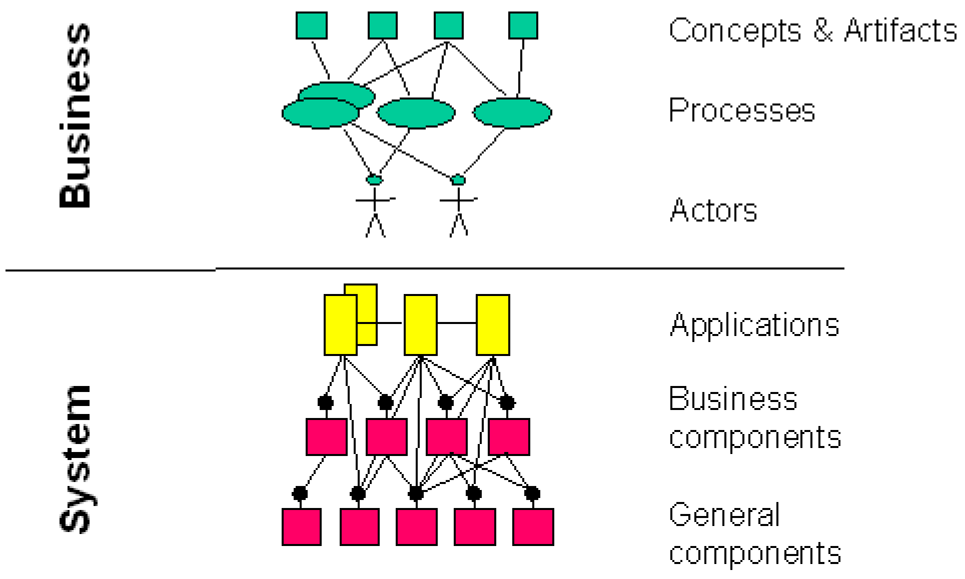

Figure 1 The fundamental structure of a domain and its supporting software systems

We distinguish two main classes of components; *business components* and *general components*. Business components implement concepts of the problem domain. General components implement the infrastructure of computational mechanisms needed by the business components to fulfill their task. This includes components dealing with UI controls, persistent storage, concurrency, transactions, communication etc., and is where component middleware like MS COM, CORBA and JEB fits in.

## Object oriented modelling with UML

Modern software systems and the problems they solve are far too complex for any human being to understand as a whole. Therefore we need to build models that allows us to view and contemplate both the problem domain and the software systems at different levels of abstraction and from different points of view. A fundamental idea is that of building a complete picture by synthesis from partial models. This is recommended in various forms throughout the handbook.

We have adopted the object oriented paradigm for modelling, which means that we think of and model the real world as well as software systems as consisting of a set of collaborating objects. We do this primarily because we think it is a more powerful modelling paradigm, and not because we think CBSE is impossible without it. The preferred notation for object oriented models is UML [FOWLER97] [AMIGOS98].

## Object oriented components

Our adoption of the object oriented modelling paradigm also has an impact on our view of components, which we generally conceive as objects, although on a coarser level of granularity the objects typically found in analysis models and in source code. Normally a component encapsulates a cluster of such finer granularity components of the same or of different classes.

## Software engineering process architecture

In accordance with the picture drawn above, the Magma process architecture defines three main software engineering processes: domain engineering, component engineering and application engineering This is illustrated in Figure 2. This three-tier process architecture is similar to the one proposed by Jacobson et al. in [JACOBSON97]

Component based systems are built through the concerted effort of such processes, where the domain engineering process provides the domain knowledge and the architectural context for component and application engineering.

### Domain Engineering

The domain engineering process is a collection of activities aiming to establish and maintain the domain knowledge and technical infrastructure necessary to effectively develop and maintain competitive software products for a given problem domain.

To this end the domain engineering process produces a set of domain models that transcends transcend individual applications and components and focus on different aspects of the domain.

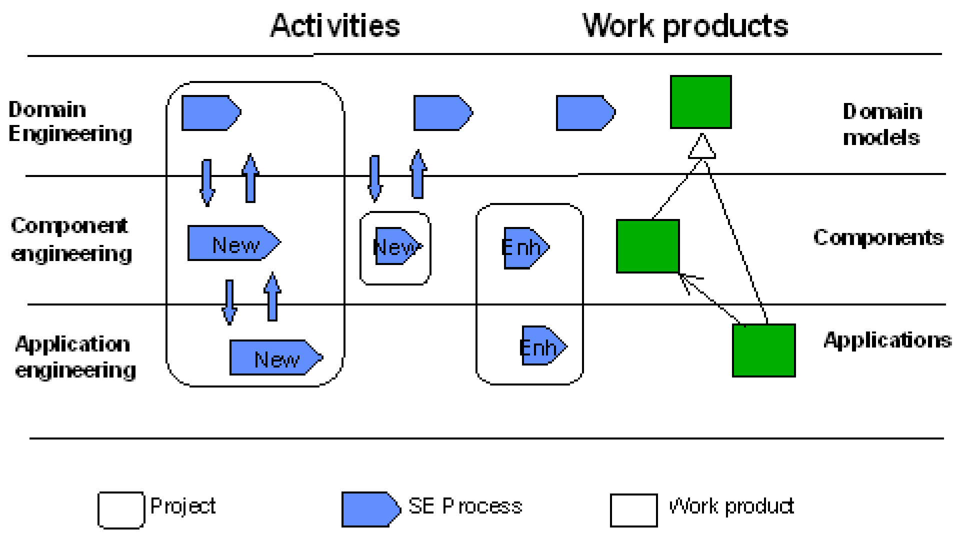

Figure 2 The Magma process architecture

The *business model* focuses on the business as such and seeks to model the processes, actors, concepts and artifacts of the business and the relation between them. The *needs model* focus on the needs for software support in the domain. The *architecture model* defines a common domain architecture, that is it identifies applications and components and the ways they collaborate to satisfy the needs specified by the needs model. The domain architecture is a key work product as it provides the common architectural context for components that we think is instrumental to CBSE.

Domain engineering is not something you do once and then is done with. Continuous activity is necessary to keep track of the evolution of the domain and keep the models up to date.

An important aspect of the domain engineering activity is to identify and accommodate the anticipated variation in the domain, both the variation caused by different needs of different users or user groups, and the variation caused by evolution.

### Component Engineering

The *component engineering* process develops the reusable components from which applications are built. Each instance of this process develops one component. The starting point is the interfaces and responsibilities assigned to the component in the architecture model.

Ideally components generalize over the anticipated variation in requirements as defined by the domain models, and this is a major concern in the implementation of components. This is not always feasible, however, and then one can build a generic component that can be instantiated with different properties depending on the specific requirements of the context in which the components is being used.

In addition to the implemented component, the component engineering process also produces a set of abstract models. The *interface model* defines collaborator roles and collaborations into which the component may engage and object and relation types visible in the interface. The *customization model* defines how the component can be customized to meet varying requirements. The *design model* focuses on the internal design of the component.

### Application Engineering

The *application engineering* process develops individual applications, that is the programs that the users see. By definition the components implement the functionality that the applications need to offer to the users. Therefore the main challenge of the application engineering process is to select the specific functionality adequate for the particular type of business process or role the application is meant to support and to design an appropriate user interface.

## Development projects

A development project has clear objectives to be achieved in a given time and with given resources. To this end the project instantiates and coordinates software engineering processes as necessary. For instance a project, whose primary objective is to develop an application and thus instantiates an application engineering process, may also need to instantiate domain engineering activities to refine the domain models and component engineering processes to develop not yet available components. Typically in the early stages of transition to CBSE, where domain models and components have to be established, there will be an overweight of domain and component engineering, while in the more mature stages, relatively mature domain models and component implementations will exist and application engineering will dominate.

An incremental project process model with use case based increments has been adopted, which is similar to RUP [KRUCHTEN99]. This type of project process seems best suited to match the need for flexible development and is also well suited for coordinating the parallel engineering processes.

## References

|   |   |
|---|---|
| [JACOBSON97] | I Jacobson, M Griss, P Jonsson,<br>Software Reuse – Architecture, Process and Organisation for Business Success<br>Addison Wesley Longman / ACM Press, Object Technology Series, 1997. |
| [FOWLER97] | Martin Fowler, Kendall Scott: UML Distilled<br>Addison-Wesley, 1997. |
| [AMIGOS98] | Grady Booch, Jim Rumbaugh, Ivar Jacobson,<br>Unified Modeling Language User Guide<br>Addison-Wesley, October 1998. |
| [BROWN98] | Alan W. Brown, Kurt C. Wallnau: The Current State of CBSE.<br>IEEE Software, September/October 1998. |
| [KRUCHTEN99] | Philippe Kruchten: Rational Unified Process – An Introduction.<br>Addison-Wesley, 1998 |