Component Based Software Engineering – a new resurgence?

Prabha Anantaram

The concept of "components" and "component-based software engineering" has been in existence for a long time. When it first appeared in the form of design patterns and later transformed into components and component-libraries it was projected as "the technology" to reduce core development-time issues. Has it been successful? What is its current status?

Introduction

The idea that software should be componentized, had its origins in the processes followed by manufacturing industries, wherein large machines or systems are assembled using many off-the-shelf components. Douglas McIlroy's address at the NATO conference on software engineering in 1968, first discussed the concept of *mass produced software components*. Subsequently, many attempts have been made to create off-the-shelf software components that are readily pluggable and usable in the development of new applications. Initially, component-based software engineering did not take off. However, with component architectures like Microsoft's Object Linking and Embedding (OLE) and Component Object Model (COM), IBM's System Object Model (SOM) and Sun's JavaBeans technology appearing in the marketplace, component technology received a fillip.

On the face of it, Component-Based Software Engineering (CBSE) seems to have gone through the first four stages of the new technology "hype cycle" defined by Gartner Group – a curve depicting "Technology Trigger", "Peak of Inflated Expectations", "Trough of Disillusionment", "Slope of Enlightenment" and "Plateau of Productivity". Some of the other less-fortunate technologies like the "dot-com based e-business" to the more exotic "artificial intelligence" did not make it beyond the third or fourth stage. The highly appealing concept of assembling applications from pre-fabricated software components, provided the components are well-defined and properly-packaged, is driving the investment of efforts in componentization.

The Software Engineering Institute of CMU has broadly defined *software component* as "an implementation, in software, of some functionality that is reused as-is in different applications, and accessed via an application-programming interface". By this definition, large-grained components, that are typically 'subsystems' (like a customer-problem-handling subsystem), do not qualify to be considered as a component. As organizations build new applications, they use techniques and processes to identify, customize and reuse components in that application. *Component-Based Software Engineering* (CBSE) deals with the development of software systems from reusable components, the development of reusable components, and system maintenance by means of component replacement and customization. With IT becoming more critical to business performance, new applications of IT in business processes are emerging daily. The complexity and variety of business processes are also undergoing a rapid change due to the changing patterns of global businesses. For example, nowadays one finds Insurance companies offering regular insurance together with money-investment services, or Travel management companies offering travel planning together with hospitality services. This in-turn makes systems more complex and also demands more and better quality software in the shortest possible time.

Component technology aims to address the above aspect in various ways. Firstly, it offers pre-fabricated set of components that can be used to provide ready functionality to parts of an application. For example, let us say that we want to build a travel application. In such an application, the COUNTRY-STATE-CITY code is an important component. Now instead of writing new code to implement COUNTRY as a drop-down-box that would list all countries one could possibly visit, followed by a drop-down-box for the respective STATES in that country and then the CITIES in that state, one could use an *off-the-shelf component* that provides such functionality. Secondly, component technology raises the level of abstraction of system building by allowing the developer to demarcate the application into levels of parts that could be used to compose the application. It should be noted that it is not necessary to have the data packaged into the component. Data could always be retrieved from a database. For example, in the COUNTRY-STATE-CITY component, the data on countries, their states and cities could come from a database, while the component could encapsulate the functionality of 'querying' the database, displaying the data, allowing selection, search on that data etc. To cater to specific requirements, components usually allow customization of their functionality. Users may also be able extend the functionality for additional business requirements.

Master Data Management and Business Components

Master Data Management (MDM) ***is a new focus area*** for many enterprises across the world. As enterprise systems grow in size and volume, the 'core business information' consisting of business entities such as customers, products, locations, currencies, payment terms and so on, have been observed to increase rapidly, and in the process, they get scattered and go out-of-sync. Over time, the master data and its access become inconsistent, inaccurate, and buried in data transaction structures, databases, etc. Inconsistencies arise in the way critical business calculations related to master data are carried out across the enterprise systems. Organizations are unable to perform analysis of entities such as customer, vendor or item across all relevant enterprise systems including sales, services, manufacturing and regional enterprise resource planning (ERP).

The lack of a single consistent enterprise mechanism to manage comprehensive core information results in incorrect answers in business processes. The cost of this inconsistency can be enormous for the business enterprise. MDM provides mechanisms to achieve consistent, comprehensive and core information across an enterprise.

Business components can provide the MDM infrastructure with a repository that contains metadata about enterprise information, and common business processes and workflows for maintaining the information. For example, a *payment methods component* would encapsulate various payment methods and their processes, or a *currency component* would encapsulate the data and processes to do currency conversions, etc. This reduces the need for each application to have its own business logic and process to operate on the core data. Moving core business logic and processes out of applications into the infrastructure components corrects the process of creating and maintaining master data. For example, in the sales, marketing and customer service functions, master data can consist of customer numbers, service codes, warranty information, distribution information and partner information. In the supply chain function, master data might include information on products, item codes and suppliers. In the finance function, master data might include information on cost centers, department codes and company hierarchies. *Thus, reusable business components form a key aspect for effective MDM.*

Designing Business Components

The major issues in designing business components are:

1) What part of the application domain should be made into components?
2) What are the reusable units that can be packaged as components?
3) How should it be packaged? How can the components be customized?
4) How can components be extended for new requirements of data and functionality?

We discuss the above issues with examples from **BitSlice**, a suite of business components built with the vision of 'Composing business applications from business components'.

1) What part of an application domain should be made into components?

To determine parts of an application domain that should be made into components, we should analyze design patterns across typical business applications, and find out portions that show less variance in functionality, are repeated in almost all business systems, yet critical to systems building. These should be the ideal 'first' candidates for componentization. 'Master Data', as has been stated earlier, forms the 'Core business entities' of the Organization. They are present in almost all business systems and form the backbone of the enterprise system. They represent the business rules and policies. They are used throughout the system, critical calculations are performed on business transactions and reports using this data, and so their criticality cannot be overstated. From a component point of view, the design and code patterns that deal with the management and usage of Master Data is so similar across business applications that they lend themselves to easy componentization.

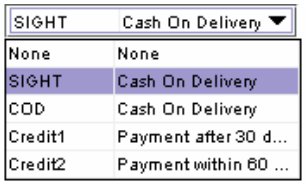
The **BitSlice-*Core*** encapsulates design patterns for processing and maintenance of Master Data into a set of 'abstract classes' with well defined 'APIs'. The core contains generic components that can be used across a variety of application systems. These components provide a uniform and consistent service layer for processing and maintenance of 'Core Business Entities'. The BitSlice suite of components extends this core and has Master Data components for typical business functions such as sales, purchase, order processing, deliveries and payments. The Core can be extended to create component sets in the Master Data domain for verticals such as Banking, Insurance, Customer Care and many other business domains.

2) What are the reusable units that can be packaged as components?

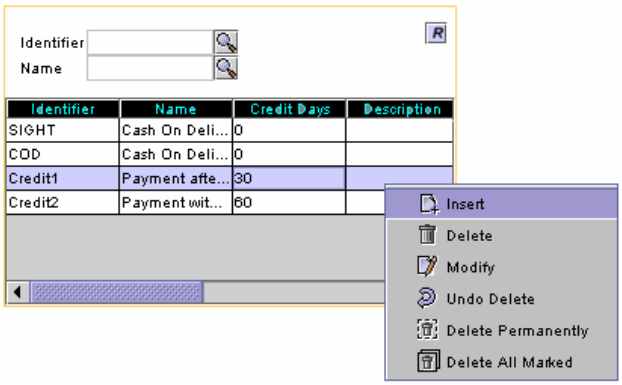
To find out the reusable units, we need to partition the domain into the typical architectural layers of UI, business object and database. Next, we need to build classes in each layer using the O-O methodology for analysis, modeling, design and implementation. Once this is done, we then need to identify small, self-contained

functional units that are frequently reused both within and across applications. For each such unit, we pick out relevant classes from within and across the architectural layers and package them as 'abstract components'. We then extend these 'abstract components' to make concrete business components for 'Master Data' pertaining to a particular business domain. BitSlice identifies seven types of key reusable components for providing Master Data Services, viz. Data Selection Lists, Business Lists, Business Object Components, Business Object Lists, Maintenance Components, Database Components, and Higher level Component Assemblies. Given below are some examples of these types from the domain of *Sales*.

Data Selection Lists These are multi-column combo boxes packaged with business data obtained at runtime from a database of "Core Business Information". A *Payment Terms* choice list is shown.

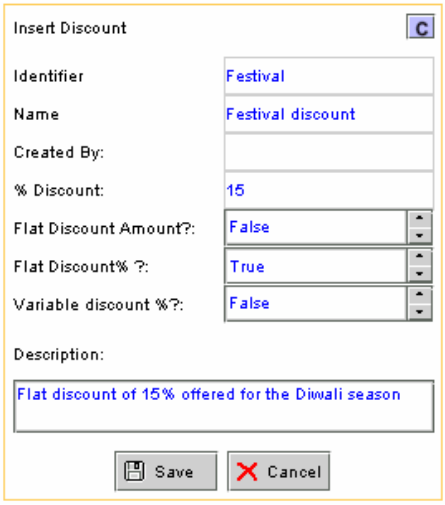


Business Lists Business lists provide a tabular view of Business Information and can be used to view and/or operate on core information such as credit policies, currency rates, delivery terms, discounts, payment terms, etc. in accordance with business validations and rules. An example is shown below.

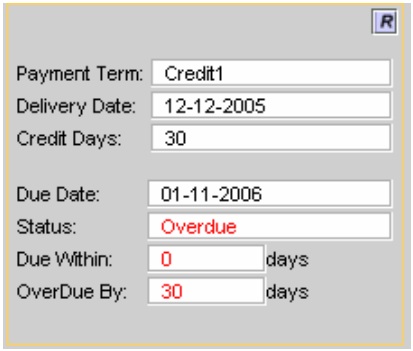


Business Object Components Business object components provide crucial services like business validations, maintenance of information structure and business computations. For e.g., the *Payment Term Business Object* does typical computations such as calculation of payment due date on a given sale/purchase, calculation of the status of a sale/purchase whether due/overdue/just due, calculation of number of days after which payment is due/overdue, calculation of interest on overdue payments, etc. Likewise, the *Discount Business Object* does discount calculations based on typical discount schemes. These non-visual components form the backbone of the master data system and usage of such components will ensure consistent master data structure and processing throughout the enterprise. This is bound to impact quality levels in all IT systems of the organization.

Data Maintenance Components These components can be used to make changes on Business data consistent with business rules and policies. They make use of underlying business object components to deliver data, perform maintenance operations, business validations and define/maintain information structure. A sample is shown.



Component Assemblies Assemblies composed from lower level components address a larger business functionality. However, since they are composed from lower level components, they have a flexible structure and can be detached and reassembled according to user needs. A *Payment Card Assembly* is shown above.



3) Packaging, Usablility and Customization

Packaging involves providing well-defined and adequately documented APIs, adhering to a technical component standard and ensuring that the components integrate easily into popular IDEs (Integerated Development Environments) used in Industry. The BitSlice components have been packaged as Java Beans and integrate into popular Java IDEs such as 'Sun Studio', 'Eclipse', etc. Extensive sets of high and low level APIs are provided to cater to different forms and levels of usage. For e.g., a PaymentTermChoiceList component can

be created with a single line of code using the following high level API:

```
PaymentTermChoiceList = new PaymentTermChoiceList();
```

This single line of code creates a drop down list that is filled in with the latest list of payment terms from a database of master data, the list positioned on the most commonly used Payment Term. This is ready to be added to (or plugged into) any form with no further programming.

The following code will create a PaymentCard showing details of payment status, due/overdue days, given a Payment Term and the 'Delivery date' which is the date on which goods were delivered:

```
PaymentCard p1 = new PaymentCard("COD", "21/01/06");
```

A component set should cater to three types of usage:

a) Components used as-is
b) Components customized for various in-built properties and used
c) Components extended using a well-defined extension process

Good usability requires all the three usage mechanisms.

Customization could mean cosmetic changes to the UI, or typical functional changes, or building in a large chunk of additional requirements of data and functionality. Customizable properties fall into two categories: a) User Interface properties (this is relevant for visual components) and b) Functional properties. BitSlice provides an online customizer through which a component's properties can be customized and the customized version of the component can then be stored and used in a particular application development. Commercial IDEs too provide online interfaces through which third party components can be customized during the design process and the customized version used during development. Components can also be customized at runtime by using the component's APIs to set the value of properties.

4) Extending the Component

Requirements within and across Organizations are bound to change with time, and components must provide a mechanism whereby the additional requirements can be built in easily. Typically additional requirements may translate into additional tables, additional columns in existing tables, changes to / additions to existing functionality. The BitSlice Core can be extended to build additional component sets for new business domains. The concrete components can also be extended for additional requirements. Both types of extensions can be done using a well defined extension process. BitSlice provides mechanisms for extending different types of components for different requirements scenarios. It also comes with test suites for different types of components. The extended components can be retested using these test suites.

Having factored out components addressing the 'Core Business Information' domain, the next step would be to factor out components in the 'Business process' and 'Business transaction' domain using a similar approach. Proceeding in this way for the entire application domain, one can visualize a component set that would make the goal of 'assembling software' become a reality. Extending the core for a number of verticals will create a market place that is rich in components for Master Data Management.

Conclusions

The use of software components is well-established in diverse business application domains, and Component-based software engineering has become a realistic and viable proposition. CBSE is adding a lot of value to rapid application development and is actively contributing to better quality software systems. With MDM gaining wide focus, component-based software engineering is undergoing resurgence. *Software component technology is widely seen as the best means of achieving the gains in programmer productivity, system flexibility, and overall system quality required by the IT revolution.*

References

1. Len Bass, Charles Buhman, Santiago Comella-Dorda, Fred Long, John Robert, Robert Seacord, Kurt Wallnau, *Volume I: Market Assessment of Component-Based Software Engineering*, Technical Note, CMU/SEI-2001-TN-007, Carnegie Mellon University, 2001.
2. Felix Bachmann, Len Bass, Charles Buhman, Santiago Comella-Dorda, Fred Long, John Robert, Robert Seacord, Kurt Wallnau, *Volume II: Technical Concepts of Component-Based Software Engineering*, 2nd Edition, Technical Report, CMU/SEI-2000-TR-008, ESC-TR-2000-007, Carnegie Mellon University, 2000.
3. Kurt C. Wallnau, *Software Component Certification: 10 Useful Distinctions*, CMU/SEI-2004-TN-031, Carnegie Mellon University, 2004.
4. Alexander Stuckenholz, *Component Evolution and Versioning: State of the Art*, Forschungsberichte des Fachbereichs Elektrotechnik & Informationstechnik, ISSN 0945-0130, 2/2004.
5. *http://www.compositesoft.com/BitSlice.html* Composite Software Systems, 2006.