Towards a Framework for Collaborative Software Development of Business Application Systems*

Robert Mühlbacher

CSC Ploenzke Informatik
Consulting
A-1010 Vienna, Austria

Gustaf Neumann

Information Systems and Software Techniques
University of Essen
D-45143 Essen, Germany

Abstract

This paper discusses the current practice in business information and office automation systems in particular with respect to collaborative development. While from an architectural point of view the predominant monolithic structure of business information systems hinders collaboration, the distributed architecture of office automation systems promotes it. We will sketch an architecture where office automation systems can be extended with active components to obtain business object systems which are able to solve problems traditionally solved by business information systems. Business object systems can support collaborative development and collaborative employment of business applications using WWW technologies.

1 Introduction

How was it possible to develop an entire operating system such as Linux (which is technically superior to many commercial operating systems) in collaborative work, whereby the developers did not have much contact among each other? Linux is certainly a puzzling example of collaborative development on the Internet, but it is not unique in this respect. Several other operating systems or tools were developed similarly. Actually, a large part of the software packages forming the Internet infrastructure was developed this way.

While the Internet infrastructure helped to develop highly successful quality technical software, so far, it did not help in the development of application software. The question arises whether it is possible to develop business application software utilizing the Internet (or an Intranet) as an infrastructure in a collaborative manner where experts in their field are enabled to contribute and can benefit from others. Candidates for collaboration in this context are

- the development and maintenance process of the basic infrastructure,
- the development of the application contents, and
- the implementation of the application system.

In evolutionary business application systems the last two items fall together. We will discuss primarily these issues in the remainder of the paper. [6] showed that when agents cooperate in a distributed search problem, they can solve it faster than any agent working in isolation. We think that many aspects in business information systems are similar to search problems and that a stronger support for collaboration can improve the business performance.

Collaborative success stories such as Linux are concerning technical systems. In a technical system the problem structure is typically clear (such as file system, memory management, etc) and often determined by hardware components. These objects are deeply structured, well analyzed and understood, exist in small quantities and don't change their relationships, dependencies, and interfaces over time.

In the area of business information systems on the other hand, the structuring of the objects of concern is not very complex, but the relationships and dependencies are quite complex and change over time. These changes are induced by new regulations, organizational restructuring, new market opportunities etc. The same objects appear in various roles, many of these objects differ in only small details which makes classification very hard and makes it often impractical to model these variations in wide class trees. The dependencies between these objects are important to keep control over this richness of variances. In contrast to most technical systems the dependencies between objects change frequently – just as the business requires (see also [7]). The

*Published in Proceedings of WETICE 96, IEEE 5th Intl. Workshops on Enabling Technologies, Stanford, CA, June, 1996

criterion for business application software quality lays in its ability to follow these changes.

2 Business Application Systems

Existing business application systems can be distinguished in business information systems and office automation systems.

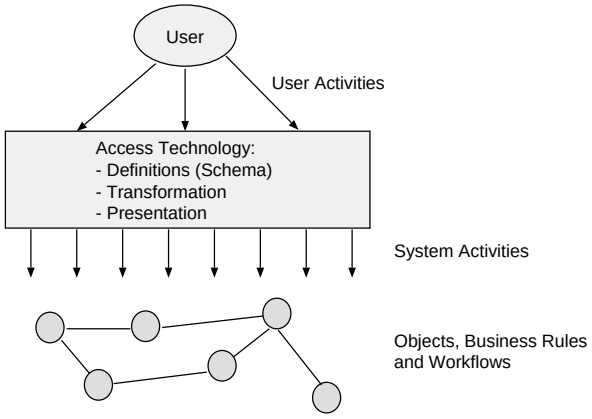

Figure 1. Indirect Manipulation of "Hidden Objects"

The key technology in classical **business information systems** is a database where typically a simple representation of business objects is kept persistently in form of tables, and the functional and organizational information is maintained in a (typically) monolithic program that controls and administers the database (Figure 1). The development of business information systems started in many cases from accounting systems as the core application, other functional business areas were added later (horizontal extension). Business information systems covering all basic functional areas are called "business solutions", the main products on the market are SAP, Oracle Financial or JDEdwards. SAP [11] for instance provides solutions in a broad bandwidth of functional business areas such as accounting, production planning, sales, marketing, distribution, etc.

One central reason for introducing standard business solution software is to stop the costly in-house software development [14]. The main advantage of business solution systems over in-house development is that the software development costs can be "shared" among the customers of the software company, especially when the number of the customers of a package is fairly high. For example the installations numbers for SAP reached 4,500 installations in September 95, the number of users is estimated as 350,000 worldwide [11]. A consequence of these high installation numbers is that business solution systems must be capable of dealing with the requirements in highly specialized business areas such as insurances, hospitals or even oil companies (SAP IS-Oil). This vertical extension is a dimension of the complexity that in-house developed software does not face.

The horizontal and vertical richness of a monolithic system requires the developer to consolidate the functional diversity and the customer to configure or parameterize the generic system according to the actual business needs. This adoption process is performed by parameterization of the standard package. Some packages offer a 4th generation programming language for making company specific enhancements. Especially the parameterization becomes increasingly difficult[1], so that knowledge based systems are suggested to aid configuration and parameter tuning [10].

When such an overall business solution is introduced it does not have only the effect of solving the basic business tasks, it tends to impose a different organizational structure as well. In many cases it appears to be easier to adapt the business organization than to adapt the software. The adaptation of a standard package is quite costly, the reintegration of these enhancements into later releases of the standard software is difficult. Consulting companies like CSC Ploenzke offer add-on packages for SAP to obtain better support for specialized business areas such as for example credit and loan applications [3]. There is no framework available to reconsolidate independently developed modification into improved future versions.

However, putting the responsibility for the business tasks primarily into the hand of a software company is some kind of outsourcing and is a capitulation in front of the complexity of the business tasks of a company. It is a threatening idea that the software packages become the "disabling technology" where a company is not able to change a business task because the software package used is not capable of dealing with this change.

The starting point of **office automation systems** was document processing and document handling (retrieval, transmission). Later office automation systems were extended to deal with more complex objects such as graphics and tables (spreadsheets), message handling became more important. The advent of products like MS-Office, Star Office and Lotus Notes turned these systems into consumer-goods like mass products. The World Wide Web can be seen as a huge office automation system supporting today primarily document processing and handling.

In office automation systems typical office objects (like documents and spreadsheets) are defined which are created and maintained over time. These objects are frequently passed to other co-workers. In a company it is very likely that several similar office objects will be used and modified

[1] SAP supplies specific utilities in version R/3 to support the user in the configuration task.

on a regular basis. In some cases it is possible to find common aspects of these objects and to define generic templates (such as styles, empty special purpose documents) for later reuse. These are typical examples for a bottom up generalization with a very weak class concept. Typically the available class hierarchies are very shallow (one level), only a few systems allow deeper hierarchies (e.g. LaTeX2e). In the case of LaTeX collaborative development was possible, various people and organizations were able to contribute for example style files.

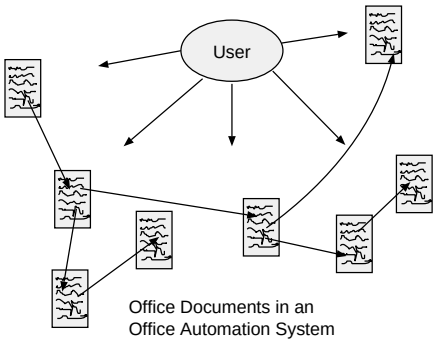

Figure 2. Direct Manipulation of "Visible Objects"

However, the current practice in office automation systems does not support collaborative business information systems sufficiently. The class models in such systems are not satisfactory, the consolidation from instances to new classes doesn't happen, at least not in a consequent manner.

3 Collaborative Application Software Development Based on Business Objects

The strength of todays office automation systems lays in the direct manipulation of objects and in its well suitedness for distributed work and collaboration. The main weaknesses are in our opinion that the objects are fairly static. We propose a system based on more general business objects which are able to describe and to implement business tasks. These business objects must be extensible to cope with business changes and problem life-cycles, and they must include active components to allow delegation of subtasks to these objects.

3.1 Basic Requirements

First we describe the technical preliminaries for the supporting infrastructure before we discuss the building blocks of the business application systems.

- The application structure should be similar to a **hypertext document**, with links between separable pieces, which might be developed and maintained by different developers in different organizations.
- Objects should have **descriptive and active components** where the descriptive component refers to informal descriptions and documentation and the active components refer to programs. These objects should be contained in single units preferably completely in text form, so that they are well suited for a transmission in a single text message to another actor. Similar examples can be found in the context of "enabled" email systems in [1], HTML documents enhanced with scripting language support are reported in [4, 16], HTTP support for agents is discussed in [8].
- The objects should have a **presentation component** which allows actors (users or agents) to introspect the objects and to allow retrieval of the objects via search engines and to ease knowledge discovery.
- The system should be based on a **dynamic object model** to allow extensibility and adaptability of the objects. The extension of an object can be performed in general by any actor (human or agent).
- For notification, task control and collaborative support a **hybrid email and web browsing tool** is needed, which is able to interpret the text based object format. The email component is for example necessary to create individual task lists. The web browsing part can be used to fetch pre-canned generic task solutions (components) which can be reused by an actor. The same tool should have a time triggering mechanisms e.g. for repeated tasks or timeouts. It accesses a persistent work item storage (mailbox) and persistent component repositories (case bases, HTTP server, databases, etc.).
- **Version management:** For collaborative development it is important to obtain information about the genesis of an object, the connection between an instance and its class together with a change history of the components. Each object in an collaborative development space should be able to answer the question: "Where do you come from and how did you become what you are now?". The version management should allow parallel asynchronous collaboration [17]; a fairly simple system for this task is cvs [15].

Every object with active components contains a program that can be executed in a certain situation (e.g. when the object is saved, fetched from a server, arrives at the client site, is read by a client, when certain time marks are reached). Mobile objects are sent in the form of messages, so every message can contain programs as well (similar to [9]

or [13]). Objects are stored in a persistent storage which can act as an active database.

3.2 Building Blocks

Conceptually business application software can be described in terms of the following building blocks.

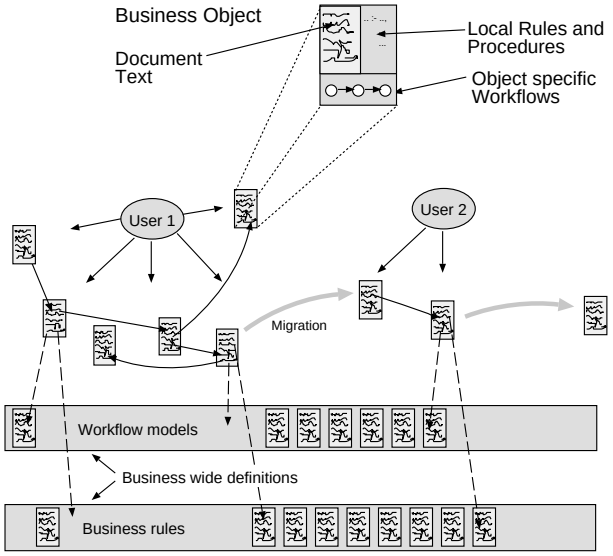

Figure 3. A System based on Business Objects

3.2.1 Business Objects

Business objects are the static descriptions of important application characteristics. Every business object has a state (attributes) and might contain methods that modify the state. All methods together define the behavior of an object. The methods are either generic methods for general object handling or they might be problem specific. Business objects are able to behave differently depending on the context. Some business objects are stationary, others are mobile.

3.2.2 Workflows

Workflows define how mobile objects (such as work items) move around in the system between actors. Actors (human or robots) modify the business objects according to their tasks in the business process. Workflows describe the lifecycle of mobile objects and their metamorphosis.

3.2.3 Business Rules

One of the main parts of a business application's description are the business rules. They describe the logic dependencies between business objects. The business rules concern primarily inter-object relationships and can be constructive (e.g. business policies, context dependent workflow or calculation rules) or restrictive (for all objects x there must be an object y with the following properties). The business rules should be usable for explanation purposes.

Business rules are the "biotope" in which business objects may "grow" or not. Business rules can be expressed for example in logic formulas which are interpreted by the object methods or might be used to control a meta object model.

3.3 Management of the Building Blocks

The basic building blocks described above can be found in more or less explicit form in practically all business information systems where the emphasis varies from application to application. When systems reach a certain size, the structures and dependencies become increasingly complex when more and more functionality and diversity is integrated into a monolithic application.

We propose an approach that should avoid the build-up of the Gordian knot of monolithic systems. The system should be able to exploit the basic Web infrastructure for finding and retrieving application building blocks and should provide means for a competent user to contribute building blocks that can be used by others. The following issues apply to all basic building blocks.

3.3.1 Genus et Differentia

All the basic building blocks (business objects, workflows, business rules) are subject to the Aristotelian principle of "genus et differentia", where "genus" is typically a well known basic item and "differentia" defines in which detail a derived item differs from the basic (parent) item. Note that this principle entails a high degree of code reuse.

A system exploiting this principle can be built in a top-down way (by specializing generic structures) or bottom up (generalizing specific structures). The most general class for the top-down approach is a universal supertype [12] as a starting point of the generalization. For example in the area of financial service providers such fundamental structures for data models are already developed (e.g. IBM's FSDM – Financial Service Data Model[2]).

A new instance (or class) is created after applying some differentia to a genus object. The resulting building blocks may be used as genus for a next development increment. A contributor to a collaborative development system should be able to contribute at each abstraction level and publish the result to make it available for others.

[2]FSDM is an IBM product sold in a price range between one and two mio US$.

The principle of genus and differentia does not stop, however, at class hierarchies. In a document oriented view, it is possible to obtain a document, to modify it locally, and when the original source document is enhanced, it is possible to transfer these changes to the locally modified document. This transfer can be done by very basic tools like the Unix "diff" and "patch" or by version maintenance systems like "cvs" [15] which allows several people to work on the same document and automatically transfers updates of the repository to the local working versions. These tools work only well for textual representations (documents, source code, etc.) since the basic "diff" tool is line oriented.

3.3.2 Treating Instances as First Class Entities

In some object oriented environments (such as C++) every object (instance) is structurally identical to its class, and every object must have a class, other environments (e.g. CLOS, OTcl [18], ObjChart) do not have these restrictions and allow for example dynamic (runtime) extension of objects. This dynamic extensibility allows to handle exceptions on the instance level without making the class model more complex: the class description can be kept small and simple, instances can be changed according to the special needs, but have to meet the business rules.

Classes describe prototypical standard cases like the examples in standard text books. Classes describe how things are supposed to work in theory. If a certain amount of instances differ in the same way (the same differentia) the theory is falsified and a new class based on the common differentia can be added to the "theory".

3.3.3 Agents and Meeting Points

An agent is somebody who acts on behalf of somebody else. Agents may be stationary or mobile. Note that our notion of a business object is more general than the notion of an agent.

Mobile objects are sent as messages between participants. The underlying interaction model is – as for every other message based system – asynchronous. Mobile objects can be either sent to a particular actor in the system (messengers) or they can roam around a set of networked servers until they are able to accomplish their task (itinerant agents [2]). The main advantages of agent based communications pointed out in [5] are that (1) agent based transactions avoid the need to preserve process state (in contrary to a client server based architecture with non-atomic transactions), (2) agents enable users to personalize server behavior, and (3) agents enable more intelligent mail handling. As noted in [9] agents reduce the number of primitive transactions on a server which can lead to significant speedups when communication latencies are high. The message size for agent based communication tends to become larger since the relevant state has to be transmitted in the message.

When a mobile object arrives at its (possibly intermediate) destination its methods can be executed in an execution environment which is frequently called Agent Meeting Point [2] where it can interact with other objects.

4 An Example

In this section we give an example (see Figure 4) how a collaborative business application system might be realized with mostly existing technology. We will use HTML documents as business object representation where OTcl with safe interpreters (as available in Tcl 7.5) is used as an embedded language within HTML pseudo comments or in special tags. The embedded OTcl commands can result in other HTML text, in addition the OTcl commands might issue side effects in the actors environment. We will not address version management here. Business objects can be sent via mail, mailboxes are used as agent meeting points, where the active components of the mails exchange their information.

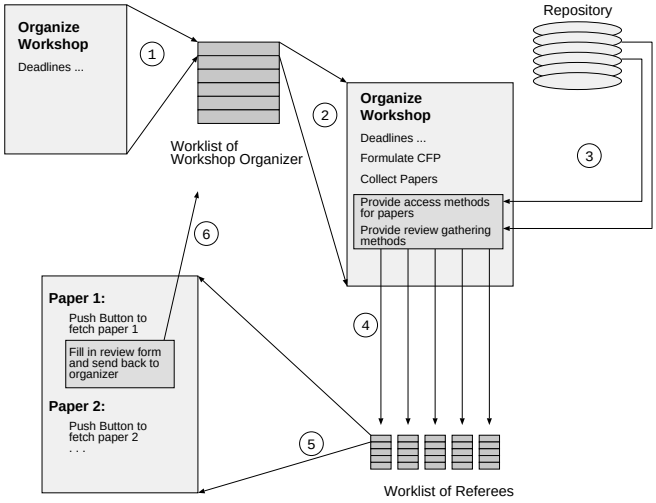

Figure 4. Workshop Organization

The task sketched in Figure 4 concerns the organization of a workshop. The initial problem at time (1) is the work description with the basic information which is added to the work-list (mailbox) of the workshop organizer. The organizer will check whether there is already a completed workshop organization document available on the network (case base repository) for example by using some search engines. We assume that the organizer does not find a suitable document and so s/he decides to develop his own. The first tasks in (2) are hard to automate anyhow, but for (3) "provide access methods for papers" and "provide review gathering

methods" he is able to locate generic tasks components on the network. For the first of these tasks he uses a simple component which accepts the title of the paper and the filename as input that adds an appropriate hypertext link to the document, and the component for the second task generates a reply form with the evaluation form, where title and author are already filled out and where a button can be used to mail the reply form back to the organizer. The organizer mails a new HTML document containing the components to the referees (4) where it is added to their work lists. At time (5) a referee opens the document, fetches the papers assigned to him and sends at time (6) the filled out evaluation form back to the organizer. The return message could contain enough information to be processed automatically by the mailbox administration program and would insert the evaluation information into the main organizing document. This document can contain methods for requiring overdue referee reports which can be triggered by time and can be sent out automatically or after interaction of the organizer. Once the conference task is over the organizer can supply the document (with or without results) to the case base repository to make it available for others.

5 Conclusion

We think that the tools and methods used in office automation systems support collaborative development much better than their counterparts in business information systems. However, active components and better access to semantic information of the documents used in office automation systems (we called them business objects) can help substantially to develop highly collaborative business applications.

References

[1] Borenstein, N.S.: "Email With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail", ULPAA 1994 Conference Proceedings.

[2] Chess D., Grosof B., Harrison C., Levine D., Parris C.: "Itinerant Agents for Mobile Computing", Research Report, IBM Research Division, RC 20010, March 95.

[3] CSC Ploenzke Projekt bei Depfa Bank, Wiesbaden, Deutschland, 1996.

[4] Kaashoek M.F., Pinckney T., and Tauber J.A.: "Dynamic Documents: Extensibility and Adaptability in the WWW", Proceedings of the "Second International World Wide Web (WWW) Conference '94: Mosaic and the Web" Chicago, Illinois, October 1994

[5] Harrison, C.G., Chess, D.M., Kershenbaum, A.: "Mobile Agents: Are they a good idea?", Research Report, IBM Research Division, T.J. Watson Research Center, 1995.

[6] Hogg T., Huberman B.A.: "Better Than the Best: The Power of Cooperation", in: Nadel L., Stein D. "1992 Lectures in Complex Systems", Addison-Wesley, Reading 1993.

[7] Lee R.M.: "Application Software and Organizational Change: Issues in the Representation of Knowledge", *Information Systems*, Vol. 8, No. 3, 1983.

[8] Lingnau A., Drobnik O. Dömel P.: "An HTTP-based Infrastructure for Mobile Agents", Proceedings of the 4th International World Wide Web Conference, Boston, MA, Dec 11-14, 1995.

[9] "Network Extensible File System Protocol Specification", (Draft) Sun Microsystems, 1990.

[10] Pietsch, M.: "PAREUS-RM – ein Tool zur Unterstützung der Konfiguration von PPS-Parametern im SAP-System R/2", *Wirtschaftsinformatik 35*, Vol. 5, 1993.

[11] SAP Ges.m.b.H, Broschüre SAP Österreich & Partner, 2.2.1996.

[12] Sowa J.F.: "Conceptual Structures: Information Processing in Mind and Machine", Addison-Wesley, Reading 1984.

[13] Tennenhouse D.L., Wetherall D.J.: "Towards an Active Network Architecture", Proceedings of Multimedia Computing and Networking 96, San Jose, California, January 1996.

[14] Thurner R.: "Innovation in Anwendungsmanagement und Softwareentwicklung", in: T. Andresen, W.-R. Hansen, N. Heydenreich, H. J. Mährländer, P.Page, H.Schiemann, R.Thurner: "Software- und Anwendungsmanagement", Oldenburg 1991.

[15] Tichy, W.F.: "RCS – A System for Version Control", *Software – Practice and Experience*, 15(7): 637-654, 1985.

[16] van Doorn M., Eliens A.: "Integrating applications and the World Wide Web", Proceedings of the Third International World-Wide Web Conference, Darmstadt, Germany, April, 1995.

[17] Vitali F., Durand D.G.: "Using versioning to support collaboration on the WWW", Proceedings of the "4th International WWW Conference" in Boston, MA 1995.

[18] Wetherall D., Lindbald C.J.: “Extending Tcl for Dynamic Object Oriented Programming”, Tcl/Tk Workshop 1995, Ontario, July 1995.