EMBEDDING CULTURE IN E-BUSINESS SYSTEMS *

Gregory (Grzegorz) E. Kersten

DSMIS, John Molson School of Business
Concordia University, Montreal, Canada
(gregory@jmsb.concordia.ca)

Abstract. Cultural consideration of software products has been limited to interfaces through internationalization architectures and localization practices. The incorporation of cultural values and practices in e-business systems may lead to modification of cultures and cause their uniformity. To avoid it culture needs be embedded in software core. Two complementary methods are proposed: one is based on the extension of software localization beyond the interface; the other utilizes emerging software design paradigms based on the meta-object orientation.

1. E-BUSINESS SYSTEMS AND CULTURE

E-business systems are the latest generation of information systems that extend beyond the enterprise allowing for communication among organizations and between companies and consumers. The enterprise may be a business or any other organization that is involved in activities such as procuring products and services, exchanging information, and brokering. E-business systems (EBS) differ from other systems in several key aspects. They are network-centric and rely on ever-present Internet connectivity. Business-to-business EBSs provide tight integration of intra-enterprise business processes (e.g., supply chain management). Business-to-consumer EBSs allow for a very large number of consumers to access the enterprise via the Internet. Their user interface is provided by the web browsers, it is easy to understand and common to many different applications. In addition, the availability and popularity of the Internet brings numerous other opportunities for transformation of business processes and creation of new forms of interaction (Buffam 2000).

Almost all the EBSs that are being introduced everywhere in the World are developed in Silicon Valley. Even if they are developed elsewhere, they tend to embed the Silicon Valley-centric view of business and economy. As Zysman notes, the belief is that "policy driven by market transformation rooted somewhere south of Palo Alto will overcome and sweep away national economic models and the state's capacity to regulate, forcing the development of a single international market" (1999).

The dominant perspective on the development of EBSs is rooted in neo-liberal, computer scientist world-view. According to this view, the world largely comprises of leaders and followers. The leaders are visionaries and they define and redefine the rules; they, and those who work with them, comprise the new class and create the "third culture" (Brockman 1995). The followers, in this new third (technocratic) culture, are burdened by policies that retard their own progress and often try to slow the ad-

* This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada and the FRDP of the Concordia University.

vance of the leaders.

This new third culture is an offspring of science; it is based on the respect and peerage that network technologies made possible. Kelly notes that it is a streetwise science culture, one where working scientists communicate directly with lay people, and the laymen challenge them back (1998).

This view implies a single model of the economy on which all societies and policies must converge. Because the U.S. has the lead, the rules governing the new economy should reflect U.S. regulatory and economic institutions and practices (Zysman 1999). Further, the understanding is that technologies should dominate the policy and social choices, and that they eventually produce a single global market, similar business models, organizations and—ultimately—beliefs and values.

The third culture is technocratic and as such it does not reject regional and local cultures, it merely ignores them. It is expansive not because of the values and believes it espouses but because of the ubiquitous and pervasive technology. It is neutral and flexible hence able to use every solution and approach to strengthen its reach and effectiveness.

This paper draws on some observations and concepts developed in studies of technology and culture. It continues the discussion on culture and software that has started recently (Carmel 1997; Cioffi 1999; Hall 1999; Kersten, Matwin et al. 2000; Kersten, Kersten et al. 2001). It builds on the foundations software technology, its development methodology, and its use.

There is no solid theory that links software and culture, or the way ideas and values are implemented in software. Such a theory is needed if the EBSs are to reflect national and regional cultures that are different from the culture in which the software was developed. It is also needed if EBSs are to support and represent organizational cultures that are different from IBM's or Microsoft's. We argue that many cultural aspects and characteristics are missing in the EBSs core but they should be embedded in these systems in order to preserve existing cultures.

According to the instrumental perspective software is culturally neutral. This appears to be a predominant perspective in the US-dominated software industry (Carmel 1997). Cultural adaptation is then limited to the interface discussed in Section 2. The instrumental perspective on software technology is contrasted, in Section 3, with rich understanding of both culture and technology.

Further, studies the EBS models and architectures are necessary, which we discuss in Section 4. Assumption that culture is present at the deeper levels than the interface lead us to propose in Section 5, a *software culturalization architecture* which is an extension of well established software internationalization architecture. The concluding implementation issues discussed are based on the concepts of aspect-oriented and feature-oriented programming paradigms.

2. CULTURES AND SOFTWARE INTERFACE

The wide spread use of software packages such as word processors, spreadsheets and browsers has lead software companies to recognize that a significant portion of their revenues are coming from outside of the English speaking world. This recognition, together with the saturation of the US market, triggered efforts to adapt the companies' products to the requirements of local markets. Software localization methods were constructed to modify software written in one language for members of one culture to another language and for members of another culture (Keniston 1999).

The requirement that software must fit the cultural context of the user has been widely accepted. However, this context has been defined solely in terms of the requirements regarding the user interface. In his answer to the question "What then needs to be encapsulated in this concept of cultural context?"

Taylor (1992) lists the following locales, i.e., the collections of all the conventions that characterize a particular culture or user community: transliteration, hyphenation, spelling, collation, national conventions (numbers, currency, time and date), and colour (op. cit.). Hall adds such elements as messages, terminology, and positioning of windows, tables and graphs (Hall 1999).

The premise behind the external perspective is that "all the culturally and linguistically sensitive software components need to be separated from the core of the application" (Hall 1999, p. 298). Following this premise, software internationalization architectures have been proposed in which the locale sensitive and independent elements are separated (Hall and Hudson 1997). Hence, the software comprises of the culturally dependent *interface* that, therefore, must be localized and the culturally independent *core*.

The separation of the core and the interface led to *software internationalization architectures* in which the locale sensitive elements are separated from the locale independent core (del Galdo and Nielsen 1996; Hall 1999). Design methods for translating software from their source market to the target markets have been developed and implemented in many products. These methods are based on three topics: (1) the choice of character codes; (2) the use of locales; and (3) the use of resource files. High-level software internationalization architecture is presented in Figure 1.

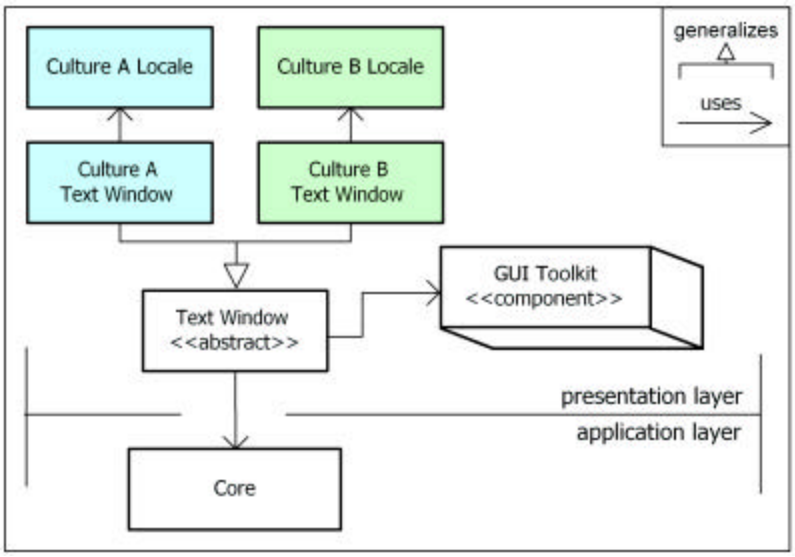

Figure 1. Interface-driven software internationalization (adapted from Kersten, 2001).

The software internationalization architecture illustrated in Figure 1, allows for localization. Keniston notes that “localization, or more generally language, has rarely been treated as an important topic in the literature on the impact of the so-called Computer Age” (1999). Individuals, organizations and governments have been, however, aware of this problem and have made attempts to address it. For example, the government of the republic of Iceland asked Microsoft to develop an Icelandic version of MS Windows so that young Icelanders would not loose fluency in their native tongue. France is one of the most active opponents of Anglo software and insists on software localization. Together with Canada’s French-speaking Quebec they have made efforts to use French and restrain the "Anglophonic tide" (op. cit.).

Governments and other organizations make significant efforts to protect their languages. The question is, however, whether or not they in fact address the deeper issue of the impact of software, and especially the EBSs, on the society and its values. If we accept the need for software localization, or even the fit to a particular culture, then the interface is an aspect that is obvious but perhaps not the most important. As we argue in the next section, there is more to culture than language just as there is more to software than interface.

3. METHODOLOGICAL CONCEPTIONS

3.1 Surface culture and deep culture

The third or the technocratic culture is empiricist and utilitarian. It is outcome oriented, and promotes individual endeavours and risk-taking. It espouses reductionist and generalizing perspective seeking universal laws and objective explanations (Kuper 1999). This culture, as all other cultures, has values, beliefs and norms, but they espouse the economic and technological progress rather than history, social interaction, faith and tradition. This led to the surface perspective on culture, one that is limited to language and other symbols that can be interpreted like language (Kersten, Kersten et al. 2001).

There is another understanding of culture. According to it culture is a structure of meanings, beliefs and values that condition human behaviour allowing while for its interpretations and purposefulness. The set of values is unique for a given culture and it cannot be detached and interpreted as an instance of a culture-free biological and economic basis. Culture is, as Hofstede put it succinctly, the software of the mind (Hofstede 1997); and "if the cultural faith is eroded, life loses all meaning." (Kuper 1999, p. 6).

This latter perspective distinguishes between *deep culture* and *surface culture*. Deep culture includes beliefs, values ideas, knowledge, procedures and norms. It manifests itself in language, symbols, artefacts and objects ranging from art to organizational structures to clothing and cars, all of which are elements of the surface culture. The meaning of the symbols and artefacts is defined by the deep culture and so any separation of the surface causes loss of the intended meaning. Thus, learning the values and other constructs underlying any particular cultural manifestation is required to interpret this manifestation within the culture that it was created by. Similarly, creating a manifestation within one culture for another culture requires understanding of the other deep culture.

Both deep and surface cultures are present in business and other organization. The surface culture is the set of symbols (e.g., logos, dress code) and it is the manifestation of the organization's values, rituals and norms.

3.2 Technology, Software and Culture

Technology is knowledge embedded in products and processes created by people to meet their needs, as well as these products and processes themselves. Several perspectives on the role and value of technology have been proposed within the philosophy of technology. According to the instrumental theory technology is neutral. The four tenets of this perspective are: (1) indifference of technology to the ends it is used to achieve; (2) indifference to politics; (3) universally rational character; and (4) possession of a common standard of measurement, typically efficiency, which is independent of producers, situations, users, etc. (Feenberg 1991). This instrumental perspective fits well with the technocratic culture and its values.

In this view technology is deterministic and proceeds separately from the demands of culture and soci-

ety (Pacey 1983, p. 83). Product design and development can be done in isolation of the users and their situation. This means that in software development interface localization is sufficient. Pacey argues that this technocratic value system is single-minded and insistent on an unambiguous and neutral view of progress, collaboration, problem solving, and values.

Hart-Davidson (1997) defines technology as the set of artefacts and the sets of cultural beliefs, practices, and texts that surround the production, use, distribution, and conceptualizations of those artefacts, designed to produce some cultural condition. In this perspective, which Feenberg calls the *critical theory of technology*, technologies are used to advance and enrich social objectives, and it cannot be seen as separate from people (1991). Software, developed by an organization, is a technology in which values and ideas of the organization and the individuals are embedded.

There are three forms in which the cultural embedding in technology occurs (op. cit.): There are three forms in which the cultural embedding in technology occurs (op. cit.):

1. It happens unconsciously being inherited via the cultural programming of its human designers and developers;

2. It is implemented intentionally in design requirements which are specified for the target markets; and

3. It emerges through the interaction of group and organizational cultures reflecting their development processes, structures, and incentives to control the environment.

The Linux and Windows operating systems are striking examples of the latter, reflecting the importance given to cultural values such as openness and flexibility versus ease of use and stability.

The cultural differences in the software core can be easily seen with three decision support systems developed in France, Great Britain and US (Kersten, Kersten et al. 2001). We may note here that the Electre family of systems, which were developed in France, have been widely studied and used in Quebec but rarely in the other provinces. Demeester gives another example provides in his overview of the European project of telematics and its introduction to medical practice (1998):

> "The way culture interferes with it is through its influence on the decisions and the decision-making processes. Dramatic conflicts between developers and users, suppliers and purchasers, or integrators of telematics applications are revealed: they often lead to failures and rejection of a priori sound solutions. … The objectives of both projects are to trigger awareness of the influence of cultural diversity on telematics and to provide a methodology and methods to use culture as an explicit component when deciding for a telematic application and implementing it."

3.3 Culture's past and future

Technocratic culture and instrumental perspective on technology, and the culture and critical theory of technology, reinforce each other. In addition to the examples given above, there are many other cases indicating the relationship between software and deep culture. This undermines the instrumental perspective, however, this limitation concerns past and present. That is, one cannot reject that the technocratic or technocratic (third) culture becomes both universal and accepted.

The possible universal acceptance of the technocratic culture does not imply that the concept of deep culture is invalid. To the contrary, it is because the values and laws of the technocratic culture are embedded in technology, and foremost in software, this culture may effectively modify or perhaps even

replace other cultures. If this happens the core (deep culture) cannot be differentiated because its values, norms and laws are universal and neutral. Differentiation would be possible but only by surface, that is, language and other symbols.

From the above follows that technology needs to be judged on the basis of its relationship with deep culture. Software, we think, plays critical and much different role than other technologies. There are at least three reasons for this: (1) software is used directly for the collection and manipulation of information and knowledge, both of which are culture dependent; (2) software, contrary do other technological artefacts, does not have long history and no competitors in the existing cultures; and (3) there is, as discussed in section 1, a strong imbalance in the software development efforts. Based on the above, two alternatives can be suggested:

1. Software characteristics are unique and they cannot be separated from the technocratic culture. Software is so powerful because its use modifies various cultures making them more uniform.

2. The power of software lies in its ability to embed different cultures and to support their manifestations. It can help to maintain and enrich language but it can also help to maintain and enrich values, beliefs and norms.

The second alternative can only be implemented if both surface culture and deep culture are embedded in software. Restricting software design and development to interface localization will eventually lead to the adoption of the deep culture of the original designers. For this reason further discussion concentrates on software core rather than its interface.

4. EBS MODELS AND ARCHITECTURES

4.1 Business models and architectures

Organizations operate in a culture. They also create their own cultures through interactions of their employees and interactions with other organizations. An EBS, from this perspective, does more than any system of the earlier generations because it directly and autonomously interacts with the environment. For customers, an organization may be only its EBS; the enterprise is represented with its EBS and all the interactions with its customers are conducted via the system. If we take a narrow IS viewpoint according to which for users "the interface is the system" then all the organizational values, rituals and norms have to be disregarded. If, on the other hand, an organization wishes to continue projecting its culture then this has to be reflected at the application level in addition to the interface.

E-business requires information systems to operate autonomously and act on behalf of businesses. The autonomy, the scope of the operations which EBSs perform, and their strategic role requires a comprehensive approach to their design and assessment. In addition, new types of businesses that Internet made possible are largely virtual, made piece-meal approach to EBS development impossible. To build a software system that operates as a complete business requires an architectural approach (Buffam 2000). First, a business model which specifies business strategies describing specifying business structure, functions and high-level operations is constructed (Treese and Steward 1998). This model provides a blueprint for the definition of the corresponding EBS model and other interacting entities (e.g., employees and legacy systems). The EBS model has functional and resource layers (Abou-Zeid 2000).

The EBS model provides the basis for the EBS architecture which comprises particular technological solutions used to implement the business model. The three entities, that is, the business model, EBS model and the high level tiered architecture are depicted in Figure 2.

The functional layer of the EBS model specifies the ways of doing business; production, marketing, fulfillment and other business processes are defined here. The resource layer comprises specialized services that enable and support business transactions including catalogues, e-money, integrated logistic systems, and contracting tools. The elements of the two layers are then mapped onto the e-business architecture.

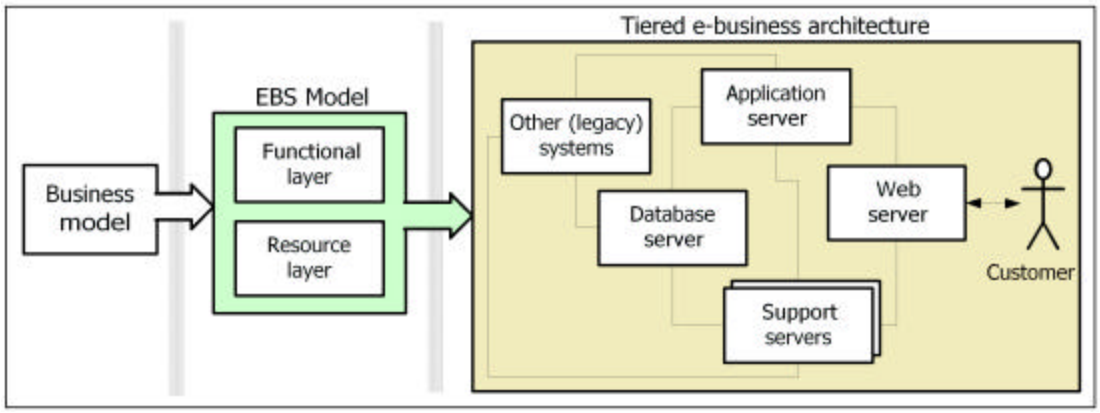

Figure 2. Business model, EBS model and e-business architecture

The e-business architecture has several distinct servers, including the database server, application server (also known as e-commerce server) and a web server. There are also other servers, which we call here support servers, for example the knowledge-base server used for personalization, server for site management or for translation.

The functional layer elements are implemented in the application server and provide the business logic components (e.g. as Enterprise Java Beans). Some of these elements may be considered as support business processes and separated from the application server (e.g., security, search, and personalization). These elements are implemented, as depicted in Figure 2, in the support servers.

The resource layer element provides design specification for specialized processes, which can carry business functions, and for the e-business infrastructure. They are mostly implemented in the database server and support servers, some, however, may be implemented in the application server.

4.2 Objects

The modelling approach to system design and the extensive use of servers is heavily based on the object-oriented approach to system analysis and design. The system (enterprise) is represented with object models, which are abstractions of the real-world objects. The separation of the object's behaviour from their implementation allows for context-dependent action rather than the necessity for a higher level of control program. Objects are initiated by a particular state and they modify this state, thus they indirectly activate other objects. This can be efficiently implemented in a server configuration, for example, an application server.

Separation of the object behaviour from its implementation makes design and development easier and more flexible. Each object can be designed separately and its implementation may change without affecting its behaviour. The object-oriented approach has had an enormous impact on the efficiency and effectiveness of software development. However many real-world objects may have something in common, the same objects in one configuration may behave differently than in another configuration,

and objects' behaviour may be affected when they share certain features.

A real-world object, for example a real estate agent provides different services to customers in one culture than in another. In some cases the difference may be due to a different configuration of objects in which the agent operates, in other however, the set of objects may be the same. The difference may be due to a similarity in the attribute value assignment to objects. The object-oriented approach does not preclude modelling of similar systems immersed in different cultures or systems which behaviour is influenced by the some similarity the objects share. These, often implied, relationships must be explicitly represented and additional objects may be needed to provide expected behaviours. Several software design approaches, which address these issues, have been proposed and we discuss them in Section 5.

4.3 Example

Consider two small real estate agencies conducting business on the Internet. One agency, called REA1, is located in a "third culture" country with individualistic, efficiency-oriented, time-conscious and mobile people. The other agency (REA2) is located in a traditional culture where "time is slow", people are community-oriented and they live in one place for generations. The real estate business can be represented with an electronic broker model. In Figure 3 we present a simplified model of two real estate agencies, based on the electronic broker model proposed by Julta, Bodorik et al. (1999).

There are both similarities and differences between the two models presented in Figure 3. They reflect the agencies' organization and functions. One difference is the involvement of the community in the transactions; both customers and the community are interested in establishing contacts prior to the purchase. This requires the agency to collect data about the community and to provide a discussion forum for the customers and community. Other differences may include different roles of the agency in its interaction with the bank or other financing company, scope of collaboration with other agencies, and so on.

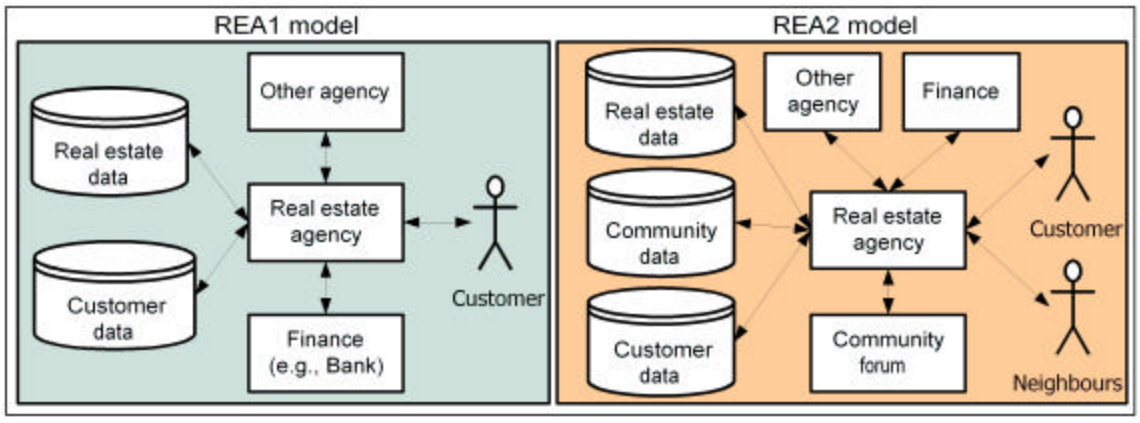

Figure 3. Two real estate models

The differences would be manifested not only in the interface of each agency EBS but also in its core. The information about customers and the properties collected by each agency is different as well as certain transactions may be different. The additional elements in one business model will be implemented as applications in one EBS and they absent in the other EBS.

5. EBS CULTURALIZATION ARCHITECTURE

The existing approach to software development is based on selection of the underlying decomposition methods with two most prominent approaches being object orientated and functional decomposition. In order to obtain consistency in defining the hierarchy and its elements decomposition methods require the use of a single criterion. The use of multiple criteria would cause that some elements could not be separated and classified leading to the "tyranny of the dominant decomposition" by object or by function (Ossher and Tarr 1999).

Decomposition of a real-world object allows obtaining elements that are represented with software components. These components are then placed together so that they represent some or all functionality of the real-world object, can support its functioning, or undertake activities on its behalf. In the current tiered EBS architectures these components are placed in the application, services and other servers.

We propose two complementary steps for software culturalization architecture:

1. Specific consideration of the culture and its characteristics at the business model level and then at the EBS model. This means that the culturally sensitive aspects of EBS are "pushed down" the interface and include the elements of the application server.

2. The behaviour of the element (object) is linked with characteristics of the level higher than the object's attributes. Similarity, aspects, and concerns are used to define the "accepted behavioural norms" for objects.

The first step requires the recognition of elements in the business model (both constructs and activities) that are generic for every type of business as well as every business of the particular type. For example, products and services, customers, business transactions, and security are present in every business. Other elements, like inventory, production and warranty services are typical for some businesses. These elements are present in the functional layer of the EBS model, however the distinction between the functional and resource layers.

A deepening of the "locale" concept is one way to proceed, capturing those cultural attributes that can be precisely measured and represented. Such an extended locale can be treated synonymously with "cultural profile", a subcomponent of the "user profile".

The second step involves new software design paradigms like aspect-oriented programming and subject-oriented programming which specifically recognise the cross-cutting and similarity of objects (Kiczales, Lamping et al. 1997; Ossher and Tarr 1999). Concerns and aspects may be viewed as application attributes which determine which software components need to be selected and configured to correspond to a particular business logic model. In this sense aspects can be used to represent shared cultural traits and the specific realizations of these traits on the software implementation-level.

An example of EBS culturalization architecture, which is based on the first step, is presented in Figure 4. It involves the internationalization of the interface and the core. Internationalization of the software core distinguishes culture-dependent components from other components of the core. This is not to say that there are components that are completely independent of any culture; this would contradict the critical theory of technology. Different cultures share certain values and beliefs; therefore there are certain mechanisms which can be used to represent business processes across many cultures.

The use of meta-object methodologies (e.g., aspect- and subject-oriented programming) requires the identification of the factors and phenomena that are involved in the influence of software on culture, and in the influence of culture on software. This could result in "cultural testing tools", i.e. rulesets

characteristic for specific cultures against which some types of software (e.g. groupware) could be run to detect potential conflicts and/or inconsistencies. These methodologies can facilitate the incorporation of culturally specific attributes into the software core because they aim at capturing concerns that affect multiple parts of a software system.

One of the outcomes of such a program may be a framework in which culturally sensitive features are organized into a "best practices" collection of guidelines/recommendations for software design. Such a framework would be very interesting from the research viewpoint. It would be also useful from a business viewpoint if it speeds the process of internationalizing products (or more generally, expanding from one market to another, even within the same country).

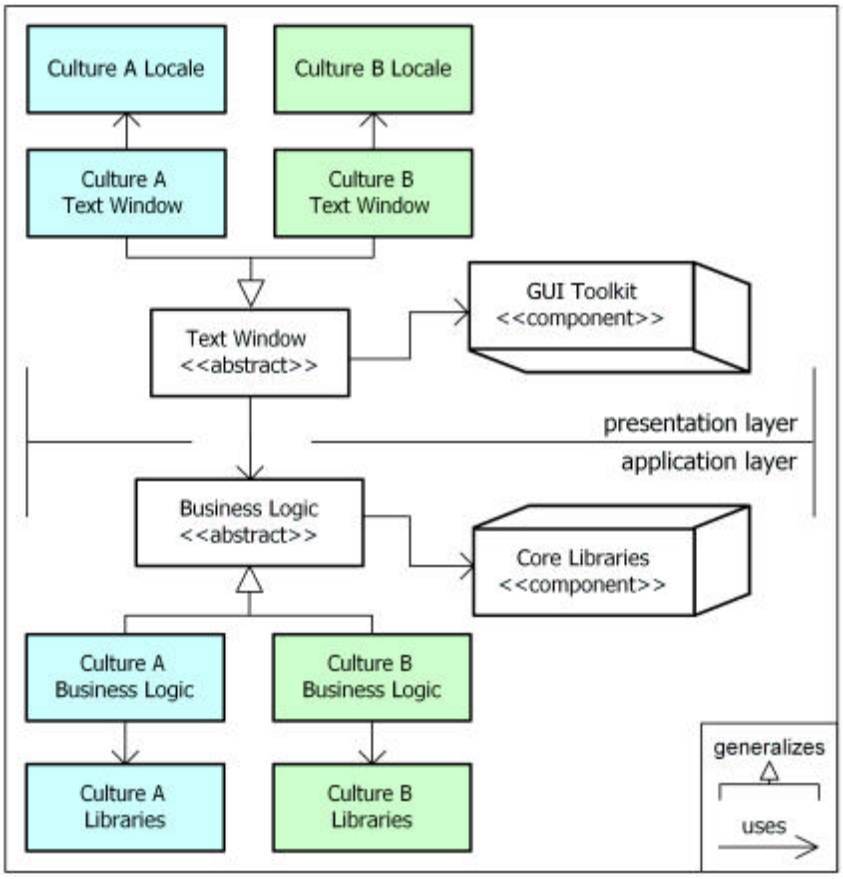

Figure 4. EBS culturalization architecture.

That is purely a call for a cultural research framework. Basically, the requirement is to provide the "intelligence" that diagnoses a user as belonging to a given culture (profiling), and the business knowledge that determines how what actions should be taken by the software in order to adapt to the culture.

REFERENCES

Abou-Zeid, S. (2000). "Situation-based Approach for E-Business Functional Modelling". Proceddings of the Object-Oriented Information Systems.

Brockman, J. (1995). The Third Culture: Beyond The Scientific Revolution. New York, Simon & Schuster.

Buffam, W. J. (2000). E-Business and IS Solutions. An Architectural Approach to Business Problems and Opportunities. Boston, MA, Addison-Wesley.

Carmel, E. (1997). "American Software Hegemony." The Information Society **13**(1).

Cioffi, J. W. (1999). The Digital Economy in International Perspective: Common Construction or Regional Rivarly. E-conomy Project, University of California, Berkeley, http://e-conomy.berkeley.edu/events/deip/summary.html. Accessed: December 2000.

del Galdo, E. M. and J. Nielsen, Eds. (1996). International User Interfaces. New York, Wiley.

Demeester, M. (1998). "L'implication des différences culturelles en télématique médicale : l'expérience des projets européens VICO et Babel." Santé et Réseaux Informatiques **10**: 10-20.

Feenberg, A. (1991). Critical Theory of Technology. New York, Oxford.

Hall, P. (1999). "Software Internationalization Architectures". Decision Support Systems for Sustainable Development in Developing Countries. G. E. Kersten, Z. Mikolajuk and A. Yeh, (Eds.). Boston, Kluwer**:** 291-304.

Hall, P. and R. Hudson (1997). Software without Frontiers. New York, Wiley.

Hart-Davidson, B. (1997). Locating the Techno-Discourse. Purdue University, http://omni.cc.purdue.edu/~davidswf/tds.begin.html. Accessed: December 2000.

Hofstede, G. (1997). Cultures and Organizations: Software of the Mind. New York, McGraw-Hill.

Julta, D., P. Bodorik, et al. (1999). "Making Business Sense of Electronic Commerce." IEEE Computer, (March): 67-75.

Kelly, K. (1998). "The Third Culture." Science **279**(53): 992-993.

Keniston, K. (1999). Language, Power, and Software. MIT Program in Science, Technology, and Society, http://web.mit.edu/kken/Public/papers3.htm.

Kersten, G.E., M.A. Kersten, W. Rakowski (2001). Application Software and Culture: Beyond the Surface of Software Interface. InterNeg.org, http://interneg.org/interneg/rese-arch/papers/. Accessed: January 2001.

Kersten, G. E., S. Matwin, et al. (2000). "The Software for Cultures and the Cultures in Software". Proceedings of the 8th European Conference on Information System. ECIS2000, H. R. Hansen, M. Bichler and H. Harald, (Eds.). Vienna University of Economics and Business Administration. **1:** 509-514.

Kuper, A. (1999). Culture. The Anthropologists' Account. Cambridge, MA, Harvard University Press.

Ossher, H. and P. Tarr (1999). Multi-dimensional Separation of Concerns in Hyperspace. IBM T.J. Watson Center, http://www.research.ibm.com/hyperspace/Papers/sac2000.pdf. Accessed: January 2001.

Pacey, A. (1983). The Culture of Technology. Cambridge, MIT Press.

Taylor, D. (1992). Global Software. Developing Applications for the International Market. New York, Springer Verlag.

Treese, G. W. and L. C. Steward (1998). Designing Systems for Internet Commerce. Reading, MA, Addison-Wesley.

Zysman, J. (1999). "Introduction and Overview: Common Stakes in the E-conomy". Proceddings of the Digital Economy in International Perspective: Common Construction or Regional Rivarly, J. W. Cioffi, (Ed.). E-conomy Project, University of California.