Proceedings

3rd International Workshop on Net-Centric Computing

Migrating to the Web

May 14, 2001

Toronto, Canada

www.cs.ucr.edu/~stilley/ncc2001

Edited by Scott Tilley
Table of Contents

Message from the Organizers .......................................................... 5

Web-Centric Business Process Reengineering ................................. 7
L. Aversano, A. Cimitile, A. De Lucia, and P. Gallucci

Mechanically Transforming a Legacy Application into a
Network Application ................................................................. 12
I. Baxter

Net Centric Computing: Now and in the Future – Some Fictional Insights ... 17
C. Boldyreff

VARLET/BABEL: Toolkit for Net-Centric Legacy Data Integration ....... 22
Y. Bychkov and J. Jahnke

microSynergy: Generative Tool Support for Networking Embedded
Controllers ............................................................................... 26
M. d’Entremont and J. Jahnke

Call Control Component Implementing Converged Telephony Internet
Networks .................................................................................. 32
P. Falcarin and P. Lago

Reengineering for Evolution of Distributed Information Systems ........ 36
H. Giese and J. Wadsack

An Integrated Approach for Software Documentation Based on
MSR Standards ..................................................................... 40
J. Hartmann

Performance Optimization Appraisal on Web-enabling
Legacy Applications .................................................................. 44
R. Rajagopal and E. Xavier

Experience in Migrating Legacy Systems to the Web ........................ 48
W. Scacchi
Towards A Reference Architecture for Service Integration on the WEB ............................. 51
E. Stroulia

The State of Net-Centric Computing in Early 2001 ......................................................... 57
S. Tilley, S. Huang, and K. Wong

IT-based service as solution for workgroup/cooperations .............................................. 62
M. Wiedeler

E. Xavier

Towards a Web-centric Legacy System Migration Framework .............................. 70
Y. Zou and K. Kontogiannis
Welcome to NCC 2001! The underlying principle of Net-Centric Computing (NCC) is a distributed environment where applications and data are exchanged among peers across a network on an as-needed basis. NCC relies on portable applications running on multiple platforms, mobile code and data accessed via high-speed network connections, and low-cost appliances for local processing.

In keeping with the theme of “Migrating to the Web,” NCC 2001 focuses on issues related to reengineering legacy systems for use in an NCC environment. Of particular interest are holistic techniques for Web-enabling existing applications that integrate various reengineering aspects (e.g., code, data, and user interface reengineering) into a “whole system” modernization process. The workshop is structured around three central issues: decomposing legacy systems to identify logical components representing essential functionality, developing a new Web-enabled system using these components, and deploying the new system in an NCC environment.

NCC 2001 provides an opportunity for the exchange on information related to exciting new research and empirical results in areas including (but not limited to):

- Business issues in migrating to the Web
- Integration strategies based on XML and nascent standards such as SOAP
- Mining assets from existing systems
- Concerns in deploying NCC applications in fixed-link and wireless settings
- Using distributed component technology in developing Web-enabled systems

We are very pleased that the third meeting of this emerging research community is part of the 23rd International Conference on Software Engineering (ICSE 2001). ICSE is the flagship event focused on all aspects of software engineering that is sponsored by both the Association for Computing Machinery and the IEEE Computer Society. Hopefully NCC 2001 will follow the example set by the previous NCC workshops in Toronto and London to provide a stimulating and enjoyable event.

Organizers
Jens Jahnke, University of Victoria, Canada
Kostas Kontogiannis, University of Waterloo, Canada
Eleni Stroulia, University of Alberta, Canada
Scott Tilley, University of California, Riverside, USA
Kenny Wong, University of Alberta, Canada

www.cs.ucr.edu/~stilley/ncc2001
Web-centric Business Process Reengineering

Lerina Aversano, Aniello Cimitile, Andrea De Lucia, Pierpaolo Gallucci

Faculty of Engineering - University of Sannio
Palazzo Bosco Lucarelli, Piazza Roma - 82100 Benevento, Italy
(aversano/cimitile/delucia/gallucci@unisannio.it)

ABSTRACT
Migrating legacy systems towards Web-enabled, client-server architectures is only an aspect and cannot be considered as the solution to a web-centric reengineering of the business processes supported by these systems. We present an approach to reengineering business processes towards web-centric architectures that exploits a web-based workflow management system as the technological basis for the integration of legacy systems and the coordination and the communication of the different actors involved in the process.

1 INTRODUCTION
Moving to Web-enabled, client-server architectures is widely recognized as a must for keeping competitive in the dynamic business world [7, 13]. The continuous and increasing business changes and the consequent need for organizations to be flexible and adaptable impose that information systems have networked architectures, as these are intrinsically more adaptable than centralized architectures. Migrating core legacy applications towards Web-enabled, client-server architectures is also recognized as a key to successfully moving to the Internet while salvaging the past investments in centralized, mainframe-oriented software development. As a matter of fact, the increasing demand by large and small organizations around the world for integrating their information systems with the Web has determined a tremendous need for methods and tools to adapt legacy applications to the Web [3, 20, 27].

However, migrating legacy systems to the web is only an aspect and cannot be considered as the solution to a web-centric reengineering of the business processes supported by these systems. In particular, this is not enough for managing the coordination and the communication between the different actors involved in the business processes of companies that aim to move to virtual organization models. Workflow and computer supported cooperative work technologies are primary enablers for virtual organizations, as people and institutions in a network make substantially more use of computer mediated channels than physical presence to interact and cooperate in order to achieve their objectives. In particular, Workflow Management Systems (WfMSs) enable process automation through the integration, the coordination, and the communication of both human and automatic tasks of a process [15, 25]. In addition, most WfMSs of the last generation are web based [1] and this makes them a viable enabling technology for remodeling both the organization structure and its processes in order to move towards a virtual organization model.

This paper presents an approach to reengineering business processes towards web-centric architectures. Reverse engineering activities are required to analyze and model the flow of the activities, the actors, the documents, and the information systems involved in a business process. This issue is discussed in Section 2. A web-based WfMS is used as the core technological platform for the reengineering of the business process and the integration of the legacy systems. Workflow automation and legacy system integration is presented in section 3. Section 4 reports on our experience, while concluding remarks are given in Section 5.

2 PROCESS AND SYSTEM REVERSE ENGINEERING
Migrating a business process to the web requires an initial step consisting of understanding and modeling the business processes together with the involved documents and software systems. In this section we outline some guidelines for business process understanding and modeling. There are five main phases, each consisting of several steps; phases 1 and 2 define the context and collect the data needed to describe the existing processes, with related activities, documents, and software systems; phases 3, 4, and 5 focus on workflow modeling, document modeling, and legacy system analysis and assessment, respectively.

2.1 Context definition
The first phase consists of two steps: scope definition and process map definition. The goal of the scope definition step is the identification of two key elements: the context of the analyzed organization (i.e. the units it comprises and the other entities it interacts with) and the products (i.e. the objects handled within the processes of the organization). In the process map definition step a description of the process at different levels of abstraction is produced [16] according to a top down approach.
Defining a process map helps to identify the key processes, the involved documents and software systems.

2.2 Data collection and process description
The goal of this phase is to collect all information needed to describe and assess the existing processes and the involved documents and systems. This information can be achieved in different ways, for example through observations, questionnaires, and interviews conducted with process owners and key users. The collected information about workflows, documents, and software systems are included in a structured process description document.

2.3 Workflow modeling
Workflow modeling involves two separate steps: activity map definition and activity documentation. The first step aims to produce a semi-formal graphical model of the analyzed process. Different approaches to workflow modeling have been proposed in the literature [8, 9, 24, 26]. We use UML activity diagrams to model the flow of the process activities, including decisions and synchronizations, use-cases to model organizational aspects, i.e., which actors (roles) participates to which use-case (activity or group of activities), and interaction (sequence and collaboration) diagrams to depict dynamic aspects within a use case [5]. In the activity documentation step the workflow model produced in the previous step is completed with detailed documentation.

2.4 Document modeling
This phase produces a model of the content, structure, and mutual relationships of the documents involved in the workflow. It is based on two steps: document class definition and document life cycle modeling. Initially, the documents are partitioned into classes on the basis of their content and the relationships existing between document classes are identified. The result is a document-relationships model modelled through a UML class diagram, where nodes represent document classes and edges depict mutual relationships. The second step consists of describing the life cycle for each document class, i.e. its dynamic behavior, through UML state diagram.

2.5 Legacy systems analysis and assessment
This phase aims to assess the existing software systems involved in the business process to identify the most feasible web migration strategy. Usually, a legacy system is assessed from two points of views: a business dimension and a technical dimension [4, 23]. Most of the information for the evaluation of the business value is collected in phase 2, mainly through interviews and questionnaires. The technical value of a legacy system can be assessed through different quality attributes, such as the obsolescence of the hardware/software platforms, the level of decomposability, the maintainability, and the deterioration. This information can be achieved by combining interviews and questionnaires with the analysis of the legacy source code and of the available documentation.

3 MIGRATING BUSINESS PROCESSES AND LEGACY SYSTEMS TO THE WEB
Migrating a business process to web-based workflow technologies entails two steps: implementing the activities and document workflow using a web-based workflow management system and migrating the legacy systems to the new process enactment platform.

3.1 Web-based workflow automation of business processes
A Workflow Management System allows the definition and enactment of business processes. In particular, it includes a process definition tool used to design a business process and the organization structure and a workflow engine used to enact and control the execution of a process instances and coordinate the actor workload. A database contains predefined workflow procedures, status information for procedures underway, and complete histories of previously executed procedures. A web-based WfMS also includes a Common Gateway Interface (CGI) front-end allowing users to interact with the system via a browser (the WfMS client) [1]. The web browser uses the Hyper Text Transfer Protocol (HTTP) to send requests and to receive documents from a HTTP daemon process. The HTTP daemon helps the user to interact with the workflow engine by calling the CGI program responsible for workflow activities. The CGI program communicates both with the workflow engine and with the workflow database to retrieve instructions for the user and to notify the workflow engine of the activities the user has performed. Documents involved in the business process and modelled in the previous step can be automatically produced from the information exchanged through HTML forms and stored into the database; they can also be automatically printed, faxed, and e-mailed.
Implementing the workflow of business processes entails a preliminary identification of the workflow platform more suitable for the subject organization. This entails the selection and evaluation of different platforms according to the organization requirements. Reference [2] presents a method for the evaluation of workflow technologies. It represents a customized and simplified version of the DESMET method [17] for the evaluation of software engineering technologies and exploits quality characteristics derived from the Workflow Management Coalition reference model [25].

3.2 Web-based legacy system migration
Migrating a legacy system to the Web mainly entails decoupling the software system from its user interface; the latter has to be reengineered or re-developed using Web technologies. In the short term, the server part of the legacy system can be wrapped and accessed through the Web server [3]. Different strategies can then be used to incrementally migrate the encapsulated legacy component towards modern technologies [6, 7, 11, 14]. Migrating a legacy system to the Web is a complex process that depends on the decomposability of the system [6]. The decomposability level of a legacy system identified during the assessment is the basis to make decision about which migration strategy is the more
appropriate. In particular, non-decomposable systems\(^1\) are the most difficult to be migrated, because decoupling the user interface from the application logic and database components might require complex restructuring interventions [10]. Whenever the restructuring effort required to decompose the legacy system (or a subsystem) is too high, or whenever the restructuring of the system results in a degradation of the performances it might be more convenient to completely reengineer or redevelop the system using Web based technology [3]. Reference [3] presents an approach to short term migration of semi-decomposable systems\(^2\) to the web. User interface reengineering consists of a reverse engineering phase to abstract a user interface conceptual model [19, 18] and a forward engineering phase to re-implement the user interface using Web based technologies. In particular, the WfMS client should be used as basis for the user interface reengineering to enable the integration of the legacy system into the workflow prototype. On the other hand, the server part of the legacy system is wrapped to enable its access through the new user interface [21, 12]. Data-flow analysis is necessary to identify the parameters exchanged by the user interface and the server program [12] and enable the integration of the two parts in the Web based architecture. Wrapping is also needed to enable the exchange of information between the non-interactive parts of a legacy system and the workflow database.

4 AN EXPERIENCE

We are experimenting the approach described in the previous section in an on-going technology-transfer research project, named LINK, aiming at transferring enabling business process reengineering (BPR) methodologies and workflow technologies to local small and medium enterprises (SMEs). The goal is to make software analysts of local SMEs able to move from traditional software development and maintenance to technology analysis, business process reengineering, and organization and software system integration. A peripheral organization of the Italian Public Administration is being used as the customer organization. Therefore, the administrative processes of this organization and the involved documents and legacy systems have been analyzed using the guidelines presented in Section 2.

4.1 Workflow prototype implementation

In our project we have analyzed and evaluated different web-based WfMSs according to the methodology mentioned in Section 3 and presented in [2]. The WfMS selected for our project is Ultimus Workflow\(^3\), a Web-based Client/Server workflow suite running on Microsoft Windows NT. The Ultimus Workflow Server controls the execution of the workflow processes. It is tightly integrated with Microsoft Transaction Server, BackOffice Server, Microsoft Internet Information Server (IIS), and enterprise databases, such as SQL Server and Oracle. It communicates with the Ultimus client using COM/DCOM and TCP/IP.

Ultimus Workflow provides an Integrated Development Environment (Ultimus Designer) to graphically design new processes. The Designer includes tools to graphically design new processes and decompose them into sub-processes (see Figure 1). It also uses DHTML, ActiveX and Java to realize the client interface and integrate it in a web browser and allows user interface customization at run-time (see Figure 2). Information for routing and interfacing with server-side databases are stored in the internal database in the form of spreadsheets. Exception handling is accomplished through event/condition/action tables. Whenever an event occurs for a process step (activation, completion, delay, return, resubmission), the condition is evaluated and the corresponding action is taken.

![Figure 1: Ultimus process model](image1)

![Figure 2: Building the user interface](image2)

---

\(^1\) In a non-decomposable system, interface components, application logic components, and database components are not separated [6].

\(^2\) In a semi-decomposable system only interfaces are separate modules, while application logic components and database services are not separated [6].

\(^3\) http://www.ultimus1.com
servers, and file servers. In addition, it is possible develop custom flobots.
The Ultimus workflow platform has been used to implement the prototypes for the administrative processes of the customer organization. The prototypes provides functionalities at two different levels:

- process level;
- document level.

The process level includes all functionalities needed to complete the different activities, through the user interfaces activated by the client under the control of the Workflow Engine. The document level includes functionalities to automatically generate the documents starting from the data in the form-based user interface. In addition to standard office automation formats, such as MS-Word format, documents are produces in XML to facilitate their retrieval. We have developed and integrated in the workflow platforms an XML-based document management system that allows document views to be retrieved according context-sensitive queries to satisfy the needs of different types of users.

4.2 Legacy system integration
Currently we are integrating the legacy systems of the organization within the workflow prototypes. We are building over previous experiences in migrating semi-decomposable legacy systems written in COBOL to the web [3]. In this experience the legacy system was decomposed into its user-interface and server (application logic and database) components.

The user interface was migrated into a Web browser shell using Microsoft Active Server Pages (ASP), and the VBScript scripting language. ASP is a flexible solution to quickly develop interactive Web pages using executable scripts in HTML pages. The main difference with respect to other solutions is that the code is executed on the Web server before the page is sent to the browser; in this way, the Web application can work with any browser.

The server component was wrapped using dynamic load libraries written in Microfocus Object COBOL and loaded into the IIS Web server. The wrapper interfaces provide a method for each legacy program to be accessed. The wrapper receives messages from the user interface through the VBScript functions embedded into the ASP pages. The messages received by the wrapper are converted into calls to the programs that perform the required services.

The technologies used in our previous experience are compatible with the workflow platform selected for our project and therefore can be used for migration and integration of the legacy systems into the workflow prototypes. In particular, the Ultimus client is able to load web pages, and in particular ASP pages, thus facilitating the access to the web-based applications. However, with this approach the workflow engine is not able to capture relevant information exchanged between the user interface and server part of the legacy application. We aim to develop the user interface of the legacy application using the Ultimus workflow client and the scripting facilities offered (it is worth noting that ASP is one of the offered facilities). In this way we can use the internal database of Ultimus to store the information exchanged between the user and the server part of the legacy system. This approach would also simplify the communication between the client and the server part of the migrated application. Indeed, in the previous experience, the data exchanged between the user interface and the wrapper are coded into strings. The sequence of data concatenated in the input and output strings are mapped onto the structure of records of the invoked wrapper method.

5 CONCLUDING REMARKS
We have presented an approach to web-centric business process reengineering. Our approach exploits web-based workflow technologies for the automation of the activities and the coordination and communication of the actors involved in the process and web-based legacy system migration strategies for the integration of the information systems involved in the business process into the workflow platform.

XML is currently used to produce documents and allows a retrieval according to the needs of different users. Future work will be devoted to use XML as information interchange format between the legacy applications integrated within the workflow prototypes. We are investigating the possibility of implementing the communications between the workflow engine and the external applications using XML-RPC, a Remote Procedure Calling protocol working over Internet. An XML-RPC message is an HTTP-POST request enclosing the request body in XML. The procedure is executed on the server and the value returned is also formatted in XML.

REFERENCES


Mechanically Transforming a Legacy Application
Into a Network Application

Ira D. Baxter
Semantic Designs, Inc.
12636 Research Blvd #C214
Austin, Texas, USA
+1 512 250-1018
idbaxter@semdesigns.com

ABSTRACT
The problem of converting an arbitrary legacy application into a distributed application is considered. The basic steps are to parallelize, measure information flows, extract computation and distribute with message passing. We sketch these steps.

These steps can all be implemented with a program transformation system. One such system, DMS, can process legacy applications of large scale.

Keywords
Internet, legacy, parallelism, dataflow, transformations

1 DISTRIBUTING AN APPLICATION
Assume a monolithic legacy application program (Figure 1) executing as a serial process (e.g., a COBOL program) on a powerful host CPU, processing some disk-based data which has structure (e.g., a database whose structure is defined by the ER schema), and interacting with a dumb client terminal by alternating input waits with bursts of computation over the data, and transmission of computed responses for screen display.

Our goal is to convert such an application to a distributed one, in which the dumb terminal is replaced by a smart terminal capable of running a program (Figure 2). Our intention is to decrease the amount of time the host CPU spends computing, and/or to decrease the delay user sees waiting at the screen for his answers.

We assume the following:
• constant host CPU speed and fixed communication rates.
• CPU time for input parsing, data communications, and data fetching from disk, are all negligible, being relegated to hardware.
• communication and data fetch delays are not; as disk and network bandwidth is finite.

Simple models of the original monolithic host computation and user delays are respectively:

\[
\begin{align*}
\text{cpu}_{\text{mono}} &= t_{\text{compute}} + t_{\text{display result}} \\
\text{delay}_{\text{mono}} &= t_{\text{input}} + t_{\text{data fetch}} + t_{\text{compute}} + t_{\text{receive full result}} + t_{\text{display result}}
\end{align*}
\]

A possible way to reduce costs is to choose part of the host computation and data and send that to the client for processing. Some of the input can be handled by the client, leaving some input still be processed by the host. Taking advantage of the time a user spends entering input, we divide the transmitted computation into two possibly empty parts: an always-transmitted part regardless of input, and a part that depends on the input itself.

More sophisticated schemes, not considered here, can be considered if computations or data sent by the host can be cached on the client over a series of interactions.

For this to be effective on the host side, the cpu cost of sending the computation (code+data) and receiving the answer must be lower than the cost of the host simply computing the result. For this to be effective from the client’s viewpoint, the delay must be reduced.

UpdateRecords:
Begin
  Open("MyDB",DataBase)
  Input Name
  CustomerRecord=Get(DataBase,Name)
  Print CustomerRecord.Address
  Repeat
    Input NewAddress
    while Bad(NewAddress)
    CustomerRecord.Address=NewAddress
  Put(DataBase,Name,CustomerRecord)
End

Figure 1: A Monolithic Legacy Host Program
The distributed application has a simple model:

\[
\begin{align*}
\text{cpu}_{\text{dist}} & = t_{\text{compute}1} \\
\text{delay}_{\text{dist}} & = t_{\text{compute}2} + t_{\text{receive}1} + t_{\text{datafetch}1} + t_{\text{displayresult}}
\end{align*}
\]

We observe that we can only optimize on the totals, as the individual cost terms for the monolithic application are almost all incommensurate with the cost terms for the distributed applications.

We sketch the task results for each stage.

Step 1. Computing the data flow can be done by generating an SSA representation \[5\]. The result for Figure 1 is a data flow graph such as Figure 3. Nodes are computational fragments, and accept data and control signals, and produce data and/or control signals. We have labeled most arcs with the data names used in program. Implicit data such as the display screen appears explicitly in this kind of diagram. Note that a single “variable” such as the screen actually appears several times (SCREEN0, … SCREEN N+1), each appearance corresponding to a different update of that value. The SSA information also can enable a parallelization analysis of the host application, providing a opportunities for the host to overlap computing with client; note that the database OPEN step can occur in parallel with the customer name input.
Figure 3: Data flow version of legacy program with estimated data flow volume

Figure 4: After adding communication primitives
Step 2. Estimate information volume flow along each data flow arc (also Figure 3, shown in italics). This is specific to the application. The database is estimated to be huge (100Mb), customer records and fields to be tens of bytes, the screen to be 1Mb (of pixels), and the Boolean from the while loop to be a single byte.

Step 3: Partition the graph according to the information flows. Clearly, the 100Mb database should not be sent to the client, and the screen should not be transmitted to the host. Additional information that might be used is physical location of resource. Based on the data flows, computations are assigned to the host (stippled) or the client (gray) in a way that minimizes the total volume of data flowing between the two.

Step 4: Insert communication primitives between any computations that are in different machines. Pairs of communication operations are inserted, shown as triangle to represent SEND and RECEIVE primitives explicitly, assigned to the appropriate partition. (The double pair leaving the BAD? primitive will be optimized into a single pair, because both values are transmitted at the same time).

Step 5. Split the program into two halves at the communication boundary. Connections between differently colored communication primitives are removed (dashed arrows in Figure 4). The splitting step is then trivial based on the node coloring.

Step 6. A communication method is selected (e.g., RPC, CORBA, etc.) and the communication primitives are installed in the original host program after removing code for the client. This is shown in Figure 5.

Step 7. The client application may or may not be written in the same language as the host. The target language is chosen, and the computation primitives are refined into that language. For this example, the same PDL used for the host is used, as shown in Figure 6.

Conversion to a net-centric program, in which the client receives code and data rather than just data, is straightforward. The Send Start statement in the host is changed to transmit the client program itself; the Receive Start statement in the client is simply removed, as execution of the client implies it has been received.

3 TECHNOLOGY TO ENABLE DISTRIBUTION
The essential theory for partitioning legacy programs for distribution seems well understood. What is needed for this to be practical are tools for the various steps.

Semantic Designs is working on the DMS Software Reengineering Toolkit, which is well positioned to implement many of these steps. The Reengineering Toolkit is a step along the way to a larger vision of automated support for Design Maintenance [1,3].

DMS is an industrial strength program transformation system, and provides the following facilities:

• Robust parsing technology for context free grammars [6,7], that automatically builds ASTs from context free grammar definitions
• Parser definitions for popular languages (C, C++, Java, COBOL, JavaScript, XML, SQL)
• Ability to define and execute attributed grammar over ASTs, which provides a key foundation for computing arbitrary analyses over source codes
• Generic Symbol Table facilities, so that name and type resolution procedures can easily be constructed using attribute evaluation
• Source-to-source program transformation abilities, enabling the modification of programs (in their AST form), or generation of programs (as ASTs)
• Pretty-print capabilities, allowing (modified) ASTs to be converted back to compilable source text.
• Scalable foundations, able to handle tens of thousands of files in single session.

DMS is presently used for legacy system analysis and modification. A typical example is detection and removal of duplicated source code across million-line systems [4], or reverse engineering [2].

By providing robust legacy parsing engines, DMS makes it possible to process legacy codes. By providing scale

```plaintext
UpdateRecords:
Begin
  Send Start // wakes client
  Open("MyDB",DataBase)
  Receive Name
  CustomerRecord=Get(DataBase,Name)
  Send CustomerRecord
  Receive NewAddress
  CustomerRecord.Address=NewAddress
  Put(DataBase,Name,CustomerRecord)
End

Figure 5: Modified Legacy Host Program

ClientProgram:
Begin
  Receive Start // wait to start
  Input Name
  Receive CustomerRecord
  Print CustomerRecord.Address
  Repeat
    Input NewAddress
    While Bad(NewAddress)
    Send NewAddress
End

Figure 6: Extracted Code for Client
```
management, DMS enables processing commercial scale systems. By providing program transformations, DMS enables the refinement of communication primitives and the client-side abstract computation into client code that may be in a different language than the host legacy program.

We are working on control flow analysis infrastructure, to be followed by SSA construction in the style shown in this paper. We expect to be able to implement scale examples of the paradigm in this paper in the next two years.

REFERENCES
Net Centric Computing: Now and in the Future – Some Fictional Insights

Cornelia Boldyreff
R.I.S.E.
Department of Computer Science
University of Durham
Durham, DH1 3LE, U.K.
+44 191 374 2638
cornelia.boldyreff@durham.ac.uk

ABSTRACT
It has often been remarked that fictional accounts of reality can tell us more about likely future events and give us a better understanding of the present than contemporary non-fictional accounts and analyses found in textbooks. Here two fictional accounts of systems are examined for their insights into the phenomenon known as “Net Centric Computing”. One discusses the implications of an infinitely large library; and the other gives a view of software engineers as knowledge workers in the future. Three themes are discussed in this context: evolution of net centric systems, novel networking of human and software elements, and security issues relating to such systems.

Keywords
Evolution, networking, security

1 THE GROWTH OF VERY LARGE INFORMATION SYSTEMS
With the widespread up-take of the Internet throughout the world, over the past decade, there has been unprecedented growth in its most pervasive application, the World Wide Web (WWW or simply the web). The contents of the WWW constitute the largest digital library developed by human society to date. Various practical digital library projects are currently engaged in attempting a more systematic organisation of some of the contents of the WWW, e.g. see NSF Digital Libraries Initiative\(^1\), DARPA’s D-Lib Forum\(^2\), and there is the original WWW Virtual Library\(^3\); however, the WWW at present is growing at an exponential rate and very little of its contents are subjected to any formal classification, structuring, or simple maintenance procedures. With the addition of dynamic elements, such as executable programs, to large distributed information bases such as the web, their power is extended and their maintenance becomes even more problematic.

Insights into this situation can be gained from studying Borges’ short story of the library of Babel – an infinitely large library found in a possible world of timeless nature [1]. This library contains all possible books that could ever be produced. As such it describes all that could possibly be the case in all possible worlds, including the world in which this infinitely large library exists! People wander the library going mad attempting to discover the true account of their actual world. This is more or less a spoof on Spinoza’s conjecture that if something is possible, then it must be actual, found in his treatise on Ethics. So if in this world, X is true, and if it is possible that X could be false (i.e. X is not a tautology), then there is a world in where X is false. Thus, accounts of both worlds would be found in Borges’ library. The story is a little more absurd, as the books contain every possible combination of elements of the printed language and are infinite in number. So some books will be partially true and otherwise flawed by being nonsensical or false through juxtapositions of letters, etc. Borges didn’t consider the introduction of images, voice, or video clips into the library he described, but it should be clear that these would complicate the story even further.

In many respects the web itself and its associated virtual libraries found on the web resembles Borges’ Library of Babel. Others have remarked on this similarity [2][3] in more sanguine terms.

Many web users of search engines find themselves in the position of Borges’ wanderers in the infinite library as they search for reliable sources of information of relevance to their information needs. Often the information needed is there, but there is far too much irrelevant information retrieved that the user has difficult in identifying the relevant sources. Through lack of maintenance, over time, once relevant sources of information can become unreliable and out-of-date. This is problem is not necessarily solved by dynamically generating web content from an associated database unless the database itself is systematically maintained.

\(^1\) http://www.dli2.nsf.gov/
\(^2\) http://www.dlib.org/
\(^3\) http://www.vlib.org/
In researching the concept of the library of Babel, I found a Finnish web-based poetry library of Babel that is a link collection and guide to poetry resources on the Internet [4]. On inspection, the collection does not appear to be built on the same principles as Borges’ library. It is simply a systematically organised site, with links to poetry web sites, whose purpose is “to bring some help to one of the biggest problems associated with Internet: the net contains all (or much of) the information you need, but it isn't organized in any way at all and is therefore very difficult to find.” Despite the fine intentions of its developers, through lack of maintenance, some links in the collection no longer work and who knows if the contents of working links have not been corrupted or changed as well. In some ways, reaching a non-existent web page is very similar to the experience of Borges’ disillusioned wanderers in the library coming across corrupted texts.

### 2. THE FUTURE FOR KNOWLEDGE WORKERS

With the Information Revolution replacing the Industrial Revolution and earlier Agricultural Revolution, there has been a growth in service and knowledge workers and a decline in industrial and agricultural workers, e.g. farmers now form less than 10% of the population in the U.K. Piercy’s novel, Body of Glass, envisages a world in the mid-2000s where the ratio of knowledge workers to service workers is 1 to 9. As the story itself is just under 600 pages long, only an overview will be given here.

By the mid-2000s, the world as we know it consists of large multinational corporations (the multis), some free cities, and the Megalopolis (the glop). The masses live in the glop, large densely populated areas, and service the multi’s various research and production facilities. The multis’ power is based on the knowledge stored in their Bases.

Knowledge and service workers are contracted to specific multis and are subject to their regulations. The multis bid for graduates from major international universities. This is not very different from our times. One of the principal characters in the story studied at Edinburgh; and another character says to her, when he wants help repairing a robot: Edinburgh is famous for insisting every computer student learn hardware and software. You had the equivalent of what would have been called an engineering degree. It’ll come back to you. [5, p. 299]

A typical knowledge worker, a software engineer, in this period is employed in developing interfaces. For example, the character from Edinburgh University works in the field of interfacing humans to the large artificial intelligences that form the Base of every corporation and to every other information producing and consuming entity in the world via the Network, or the Net, that potentially connects everyone. In the corporations and free towns, people have access to the Net as a matter of course. These people have sockets fitted in their heads. Once connected, they can gain access directly, projecting themselves into the worldwide network, or into their local Base through their own personal bases.

In this world, the Net is public utility that provides the communication infrastructure amongst the Bases. All Bases need to use the Net to communicate and access publicly available information, but all the corporations also need to ensure that their Bases are defended from intruders, the so-called information pirates. Pirates are people who lift information from one base and peddle it to another, or they may liberate information and make it available to the information poor in the glop. Here the author projects and combines corporate espionage, hacking, and Robin Hood. The free townspeople are interesting as they live by their wits selling their software and hardware developments to the multis. They appear to be the last remnants of the civilised world, as we know it. People living and working in the free towns recognise their privileged state as the following quotation illustrates: In a world parcelled out by the multis, it (i.e. the play of minds in the image world of the Base or the Net) is one of the only empowered and sublimely personal activities remaining. [5, p. 217]

One product of the free towns described in the book is chimeras. These are security systems that conceal the real base within false bases. Currently we have no known examples of these in our web sites although the practice of web site spoofing bears a faint resemblance to an unintentional instance of a chimera without necessarily concealing an authentic web site; and certainly chimera construction is not a currently recommended way of securing a web site’s contents. For an interesting proposal regarding how to achieve covert distributed computing via web spoofing, see [6].

In this era, mass literacy has been lost; few can read in the glop. The masses have also lost any awareness of the history of ideas, sciences, and human history. The Virtual Reality entertainments available allow people to obtain stimulated experience but without any link to reality, so that even historical characters seem contemporary with the latest fictional characters. Viroms, i.e. simulated environments, are a feature of these entertainments; and viron design provides employment for creative software engineers.

The net and bases resemble the Internet and the web sites connected through the Internet; it is interesting to note that this story was first published in 1991 before the widespread use of the WWW. The number of software engineers and other knowledge workers may decline in future if our society chooses a path similar to the one portrayed by Piercy’s novel; nevertheless, her portrayal of the work of an experienced designer building chimeras may point to a lucrative area of employment for software engineers in the future. In the text, the chimera developer describes herself
as building on a grand scale more dazzling webs [5, p.217].

Percy’s story of a future world and Borges’ story of the timeless world of the infinite library both raise many interesting prospects for the future of net centric computing. In what follows, these will be considered further under three themes: evolution of net centric systems, novel networking possibilities, and security issues.

3 EVOLUTION OF NET CENTRIC SYSTEMS

Applying methods and techniques for classical Software Engineering has been suggested as one way forward to improve the maintenance of the web. This has been a direction followed at Durham since the mid-1990s with our pioneering research on Web evolution and web maintenance.

It has been suggested that the metaphor of gardening is more appropriate that of engineering in the context of developing and evolving large information systems. By focussing on the technical aspects of web engineering and giving consideration to the application of models, methods, and techniques from Software Engineering to the engineering of web-based applications, it is possible that valuable approaches and lessons from researchers in Information Systems may be ignored. Information Systems approaches widen the focus and address the bigger picture of the system in use and its impacts on society at large. The constructivist viewpoint of Software Engineering is complemented by the environmental and gardening metaphors of Information Systems; see, for example, remarks of Ray Paul at a recent Software Engineering and Information Systems Network⁴ (SEISN) workshop in the U.K.[7]. The founding members of this network have recognised the value of bringing together both research communities and practitioners in these fields.

Also see, for example, remarks of David Lowe found in his “Engineering the Web” column entitled: “Web Engineering or Web Gardening?” where he suggests that gardening metaphor with its emphasis on growth and the cycle of growth provides insight into the “fine-grained ongoing incremental evolution” of web sites, both in their contents and structures [8]. Lowe does not dismiss web engineering; he argues for its adaptation to the domain of web development and the recognition of this finer-grained evolution.

Of course, it could be argued that the gardening metaphor is not new to classical software engineering as it is advocated by Brooks [9] and is a theme developed within the work of Nardi [10].

4 NOVEL NETWORKING POSSIBILITIES

In the creation of very large scale distributed systems and information bases, we run the risk of creating chaos. One way forward could be to actively maintain such large systems by linking human agents and software agents in collaborative networks – a form of advanced Computer Supported Cooperative Working. Such software engineering teams could actively engage in both the short and long term maintenance of the systems by becoming active parts of such systems. Just as software engineers now are currently available on-line via the Internet and collaborate with other software engineering colleagues also engaged in maintaining or evolving the very systems which facilitates their collaboration.

Open source projects are examples of such collaborative enterprises; for a more thoroughly developed account of the workings of the open source community, see the essays of Raymond [11]. Many developers contribute to and communicate through the WWW. At present, parts of the web are actively maintained by programs, e.g. an active element that counts the number of visitors to a web page; in the future, more intelligent software agents could be actively engaged in gathering data from the environment, analysing it, and updating the passive and active information providers and consumers over the web, perhaps alerting human maintainers of potential problems as part of a networked maintenance team. The recognition of need for periodic maintenance of the web has been highlighted by earlier work on web management undertaken in collaboration with Richard West and his colleagues at the CCTA reported in [12] and the associated workbench developed by Dalton provides assistance with this task.

It seem likely that in future, understanding of such large systems will have to be computer assisted. Already work is in hand through initiatives such as SOAP and other proposals to facilitate the construction of software components that can be queried to explain their behaviour or simply provide a record of their construction and maintenance. Such component understanding in context is necessary for achieving the dynamic re-composition of large distributed systems built from heterogeneous components.

It seems that incorporating humans into systems will provide an extension of the above self-describing components; and experiments with videotaped system documentation, live web broadcasts, and presence through ICQ already demonstrate the feasibility of this approach, e.g. see the homepage for CVS⁵.

5 SECURITY ISSUES IN NET CENTRIC SYSTEMS

Stable useful elements of large systems, particular the core elements of worldwide networks, such as the Internet, and our very large associated distributed information bases, such as the World Wide Web, need to be protected against

⁴ http://www.seisn.rdg.ac.uk/

⁵ http://www.loria.fr/~molli/cvs-index.html
both accidental and malicious damage.

As in Piercy’s world, the security of our networking infrastructure is necessary to ensure that our net centric applications have a secure foundation. There is certainly an argument to made for more controlled regulation of our worldwide networks, just as air spaces are controlled throughout the world today. Our web-based applications are only as secure as we design them to be and the use of third party software in strategic net centric systems carries an associated risk that should not be ignored.

The development of chimeras to protect our most sensitive information bases in future seems to be an interesting avenue of exploration. It is already known that criminal use of the web employs innocuous sites to conceal more sinister content. Detection and exposure of these may also become a routine task for future security experts.

6 SPECULATIVE CONCLUSIONS

In conclusion, it is interesting to speculate further on the concept of chimeras. In Greek mythology, a chimera was a monster with a lion’s head, a goat’s body, and a serpent’s tail. This is not unlike some large legacy net centric systems currently under maintenance today. It is reminiscent also of recently developed systems that have been composed and extended in an ad hoc fashion. And this is not in any way meant to deny the value of or utility of such systems. In his famous paper, No Silver Bullet, Brooks compares describing software visually to describing an elephant [13]. Software as the mythological chimera is perhaps more apposite to describe many net centric applications. Although whether in future, chimera will be employed intentionally within our net centric applications as described by Piercy is another matter. Brooks also prophetically in his follow-up version, Silver Bullet Revisited, advocates moving from the building of systems to the growing of systems, but this metaphor can be double-edged.

In Biology, a chimera is defined as an organism built by grafting parts or tissues from different genetic origins. This is reminiscent of phrases used by Manny Lehman, “Frankenstein systems”, achieved by “software evolution with blood on your bloods” through carelessly composing software from parts without any formal basis for the system engineering process or product. Certainly many large net centric developments suffer from a lack of any underlying formalisation or basic design at the conceptual level.

The first definition is the more fanciful definition and the second more sinister. Both images stand before us as possible futures for net centric systems, and examples of both are present in our current developments. In many of our large-scale net centric applications, we run the risk of creating monstrous systems, incapable of human comprehension. It may be necessary for us to learn to live with only partial understanding of some of our most complex systems although this will not exclude the possibility that some sub-systems will be well understood or that the principles of the system’s growth cannot be explained. This is why is important to study the evolution of net centric applications in order to gain a better understanding of their growth principles.

Without such explicit principles being employed in the development of net centric applications, we run the risk of creating monsters that we cannot understand and safely evolve, and consequently society may lose confidence in such systems and perhaps legislate against their development. It may be possible to incorporate human and software agents into future systems to actively maintain and regulate the growth of applications. Whether or not our information bases in the future will need to be concealed behind chimeras in the sense used in Piercy’s story is an open question.

At present, the uncontrolled growth of the web has resulted in something very similar to Borges’ library. However, classical Software Engineering and the emerging practices of web engineering and gardening hold promise that in the future the principles of systematically developing net centric applications will be established.

ACKNOWLEDGEMENTS

I completed this paper while visiting Jacob Cybulski in the Department of Information Systems at the University of Melbourne. I would like to note my thanks to him for his kind hospitality and for the many fruitful discussions we had during my visit.

REFERENCES


First published in the USA by Knopf in 1991 as He, She and It.


VARLET/BABEL: Toolkit for Net-Centric Legacy Data Integration
(position paper)

Yury Bychkov
Department of Computer Science
University of Victoria
PO Box 3055, STN CSC
Victoria, BC, Canada V8W 3P6
http://varlet.uvic.ca/~ybychkov
ybychkov@direct.ca

Jens H. Jahnke
Department of Computer Science
University of Victoria
PO Box 3055, STN CSC
Victoria, BC, Canada V8W 3P6
http://csr.uvic.ca/~jens
jens@acm.org

ABSTRACT
Due to the rapid growth of e-commerce (and other Internet-related technologies) many companies want to migrate their information systems to the net-centric environment. In this paper we are presenting a toolkit (Varlet/Babel) that helps the user to achieve this goal. This is accomplished in two stages. In the first stage database schema is obtained and refined, then an XML description for the database schema is generated and used it to wrap access to the information system. In the second stage a mapping between the generated schema and a standard (interchange) schema is created and used to make a second level wrapper around the information system.

Keywords
- Net-centric computing, data reengineering, migration, XML, Electronic Data Interchange

1. INTRODUCTION
With exponentially increasing popularity of Internet-related technologies (especially e-commerce) many companies today want to make their data available on the Internet or Intranet and use it in Electronic Data Interchange (EDI) to easily interoperate with other companies. In order to accomplish that the companies have to translate the data into some kind of easily understood format and ensure that the resulting document conforms to the interchange standard. Lately XML (eXtensible Markup Language) [1] has been gaining widespread popularity in this area, because of its easy-to-understand structure and ability to define the schema for a class of documents (to create an interchange standard).

While it is relatively simple for new companies to do so (they just have to buy DBMS that conforms to these requirements), older companies often have to deal with pre-existing information systems. These legacy information systems (IS) usually have evolved over long period of time and lack sufficient documentation, but they contain mission-critical data and cannot be thrown away and replaced with a new system. In order to solve this problem we are proposing a toolkit (VARLET/Babel) that helps to re-engineer the information system and migrate it into the net-centric world.

2. VARLET/BABEL PROJECT
We realize that it is almost impossible to completely automate re-engineering and migration process [2], so our goal with the VARLET/Babel project is to create a set of tools that will reduce the time required for migration by helping reengineer to accomplish the following tasks:

1. Extract a schema from the legacy information system.
2. Help the user to complete and refine obtained schema. This step is required, because often the extracted database schema is incomplete (especially in case of old file-based systems) and some of the components (e.g. foreign keys, relations) may be missing.
3. Convert completed schema to the XML.
4. Create a mapping between native XML schema and standard interchange XML schema.
5. Use these two schemas and a mapping to wrap access to the legacy IS and to create a standard interface.

Varlet/Babel consists of three types of components:
- Meta-Data Re-Engineering (RE) Workbench and Integration Workbench are fixed and do not depend on the legacy information system that is being re-engineered
- A new Extractor has to be custom-built for every type of legacy IS, unless it uses some standard interface (e.g. SQL Extractor can be used for any SQL database)
- XML Wrapper and Data Conversion Agent are generated by other components.
The toolkit can be logically divided into two parts: the first part is used to generate the initial automatic XML wrapping and allow the data to be used on the Internet/Intranet, second part is used to create a mapping between this auto-generated XML schema and some interchange XML schema in order to establish the standard interface and ensure interoperability.

**Initial XML Wrapping.**

First part (consisting of extractor, meta-data re-engineering workbench and XML wrapper) of VARLET/Babel project is used to obtain the schema from legacy information system, automatically generate XML DTD\(^1\) [1] for that schema and use it to wrap access to that information system. This initial wrapping will allow user to connect the IS to Internet/Intranet or other programs/components, but is not a good solution for the interoperability, because DTD is auto-generated and does not correspond to any external standard.

**Extractor**

Goal of the extractor is to obtain the schema from legacy information system and deliver it to the Meta-Data RE Workbench. Currently extractors parse database files to do this, but eventually other methods will be added (e.g. interface scrapers). Extractor is a domain-dependent component and usually has to be specifically created for each individual IS. A set of parsers that deal with standard interfaces (e.g. SQL, COBOL) will be included in the toolkit and can be used if the information system has one of those interfaces. In order to make interface between extractor and re-engineering workbench domain-independent we have designed a fixed API that allows re-engineering workbench to query the object returned by any customized extractor and build the representation of database schema.

**Meta-Data Re-Engineering Workbench**

Often the database schema obtained by extractor is incomplete (especially in case of old file-based systems) and some of the components (e.g. foreign keys, relations) may be missing. The purpose of re-engineering workbench is to help user to complete and refine the schema. In its current implementation this workbench provides only GUI that makes schema editing easier and does not include any tools that can offer/test hypothesis on how to complete the schema. In the future, our goal is to add to this support similarly to the functionality described in [5].

After the user is reasonably sure that the database schema is complete, it is saved as an XML DTD for use by other components of VARLET/Babel project. This DTD (native XML schema) is completely auto-generated: it depends only on the structure of the schema and doesn’t necessarily conform to any standard.

**XML Wrapper**

XML Wrapper wraps access to the information system and transforms queries/data between native XML schema and legacy IS schema. To implement the wrapper we are using a tool called XLE (XML Lightweight Extractor) [3] from IBM Alphaworks [4]. XLE uses modified DTD (DTD’s elements have to be linked to fields

\(^1\) DTD (Document Type Definition): A set of markup declarations that define the grammar for a class of XML documents
and tables in the database), which our re-engineering workbench will generate, in order to create such wrapper.

XML Schema Mapping

After initial XML wrapping is completed, data from the information system can already be used on the Internet/Intranet, but because this data conforms to auto-generated XML DTD it is not very suitable for interoperability with any external tools/information systems. Second part of VARLET/Babel project is designed to address this problem. For this part we are assuming that there is a standard DTD (interchange XML schema) that is available in some online repository and user’s goal is to conform to it.

![Database Schema Editor (Babel)](image)

**Figure 2. Meta-Data Re-Engineering Workbench GUI**

**Integration Workbench**

The goal of integration workbench is to help user to create a mapping between native and interchange XML schemas and generate data conversion agent based on this mapping. In most cases it is impossible to do this mapping automatically, so integration workbench will assist user by proposing likely modifications and checking if mapping has been completed. User will apply small changes (such as moving element’s position on the tree, changing element to attribute, etc.) to the native XML schema until this modified schema matches the interchange schema.

**Data Conversion Agent**

Data conversion agent is generated by the integration workbench and serves as a second-level wrapper around the information system. It uses the XSLT script (created from recorded user interactions by the integration workbench) to convert XML documents between native XML schema and interchange XML schema. On this level access to the information system corresponds to external standard DTD and any tool/database that conforms to the same DTD can easily
interoperate with it. This can also be used for database migration: if two databases have been wrapped using the same interchange XML schema, then all data can be transferred from one to the other (because they have the same interface on data conversion agent level) and the wrapper thrown away when it is no longer needed.

**Update Agents**

Though this component is not even designed at the moment, we hope that we will be able to implement it eventually.

XML schema mapping can be a relatively time-consuming process (especially when the DTDs are very large), so we would like to avoid going through it again if the interchange DTD changes. In order to do it the update agent can monitor the repository and if a small change in the interchange schema is detected (which probably requires only a small change in the mapping), it can communicate with all data conversion agents that use this DTD and change their mappings accordingly [6].

**3. RELATED WORK**

A comprehensive CARE environment for schema analysis and migration (DB-Main) has been developed since 1993 at the University of Namur, Belgium [7]. It provides the reengineer with a powerful scripting language called Voyager 2. DB-Main's primary purpose it to reverse engineer conceptual designs from legacy databases. Still, Thiran et al. have demonstrated that it can be used to generate conceptual wrappers for legacy database applications [8]. In comparison to our approach, these conceptual wrappers are much more powerful but also more heavyweight and proprietary. XML wrappers are lightweight and provide a standardized way of accessing and exchanging mostly hierarchical data over the internet.

Several commercial tools exist for mapping XML structures to (mostly relational) database schemas, e.g., the DB2 XML Extender [9]. However, they mostly follow a top down mapping process, i.e., the database schema is generated from the XML DTD. Obviously, this kind of process is not applicable for reengineering existing databases to XML. The XML light-weight extractor (XLE) that we use in our project allows for flexible mappings between pre-existing legacy schemas and XML schemas [3]. However, it does not solve the problem of how to attain those schemas. Our toolkit tackles this problem by offering a human-centered, graphical schema reverse engineering workbench that feeds into the XLE middleware. In addition, we approach the problem of mapping proprietary XML schemas to business standard schemas.

**4. CONCLUSIONS AND FUTURE WORK**

In this paper, we presented a toolkit that can be used for the complete migration of legacy information system onto the net-centric platform, starting from extracting the legacy IS schema, through re-engineering and the initial XML wrapping and ending with full compliance with a standard interface.

At the moment we have implemented the SQL and COBOL extractors (parsers) and first alpha version of the meta-data re-engineering workbench and hope to complete the first part of the project within a month.

After that our next steps would include performing a case study on the database provided by BC Assessment (this database uses a discontinued DBMS KnowledgeMan 2.0 and has been heavily evolved over 12 years of use), starting the work on second part of VARLET/Babel project (XML Schema Mapping) and integrating functionality from original Varlet imperfect knowledge-based analyzer into meta-data RE workbench.

**ACKNOWLEDGEMENTS**

We would like to thank Karin Breuer, Derek Church, Melis Dagpinar, Dennis Li and Sebastien Martel for their work on the VARLET/Babel project.

**REFERENCES**

1 INTRODUCTION
The specialized nature of embedded systems due to their low cost, size and convenience has led to the prevalence we see today. Unfortunately, the same properties that have made them popular have also made them inflexible and hard to update. Embedded systems cannot evolve or collaborate. Attempting collaboration between multiple embedded systems, which were not designed to collaborate, is practically inconceivable. We propose that much of this could be changed. Embedded systems could easily be networked and be upgradeable in a dynamic way. They could evolve and collaborate in a hot plug-able, seamless and scalable fashion.

To allow flexible evolution, components must be able to be integrated after they are designed and deployed, i.e. a-posteriori. This necessitates always having an awareness of what resources are on the network and how they interrelated. Through our partnership with Intec Automation (a local embedded system company) an infrastructure has been developed. The infrastructure is component-based and supports an introspection mechanism. The introspection mechanism allows our tool (called microSynergy) to interrogate the interface of a software component on an embedded system, or network of embedded systems. This allows microSynergy to connect components to one another.

2 USING SDL FOR SPECIFYING NET-CENTRIC EMBEDDED COMPONENTS
SDL (Specification and Description Language) is a high level object oriented language designed by the ITU (International Telecommunications Union) for use in communication systems [1]. Some of the properties of SDL, which make it attractive for microSynergy, are:

?? SDL has a graphical representation making it easy to use and learn
?? Definitions and relations are unambiguous and clear
?? The ability to check the validity and consistency of statements (before runtime)
?? SDL is platform independent.

Figure 1. Example embedded home network
Our approach is generative, i.e., we use SDL for the specification of components and connectors between components. The SDL is used to generate Java code to allow simple integration and connectivity.

3 RAPID PROTOTYPING WITH DIAGEN
In an effort to minimize the time and effort of development we use DiaGen (Diagram Editor Generator) [2]. It allows developers to quickly design and build graphical editors. Some of the facilities it provides are provides layout managers to automatically reposition of graphical elements. Layout managers allow the automatic reorganization based upon constraints.

Figure 2. A DiaGen Generated SDL Prototype

1. Generic functionality for building and analyzing diagrams
2. Generation of Java source code

The core components of a graphical editor can rapidly be generated by:
1. Creating a specification file
2. Running DiaGen on the specification file
3. Editing the set of largely complete Java source files created by DiaGen
4. Compiling the Java sources into Java classes

The DiaGen tool allows the rapid development of graphical components and relationships among any of the graphical diagram components. The relations can be based upon type, location, context, etc. Figure 3 is an example of a specification file used by DiaGen to create components, relationships, and reduction and grammar rules. The reduction and grammar rules provide semantics to the

//Specification file
//The graphical components and their attachment points
component
state[1] { State[CircleArea] },
block[1] { Block[BoxArea] };

relation
attached[2] { Attached[ArrowEnd, BoxArea] };

terminal
tNode[1] { String name; Variable xpos, ypos;},
tChild[2];

nonterminal
Tree[1] { Variable xroot, yroot, xleft, xright; },
Subtrees[2] { Variable xleft, xright, yleft, yright; },
constraintmanager diagen.editor.param.QocaLayoutConstraintMgr;

reducer {c:state(a) ==> t:tNode(a) { constraints: x.xpos == c.xAnchor; y.ypos == c.yAnchor; });
ar:arrow(b,c) inside(b,a) inside(c,d) c1:state(a) c2:state(d) => tChild(a,d) { constraints: ar.xStart == c1.xAnchor; ar.yStart == c1.yAnchor; ar.xEnd == c2.xAnchor; ar.yEnd == c2.yAnchor; });
}

grammar {start Tree; t:Tree(a) ::= n:tNode(a) | n:tNode(a) s:Subtrees(a,b) ;

s:Subtrees(a,b) ::= tChild(a,b) t:Tree(b)
}

Figure 3. Sample DiaGen Specification File

The Java sources, generated by DiaGen, speed development up dramatically as well as make consistent and reasonable code. The developer can edit the code to provide definitions for visual representations as well as mechanisms for testing the constraints applied to the components. The amount of work required designing and developing a graphical editor with layout control and semantic validity checking is thus minimized.

2
REALIZATION OF MICROSYNERGY

At startup, microSynergy inspects all micro-controllers existing in the LCN (Local Component Network) and queries them for their in- and out-gate components. The SDL generated from the query of a heater controller could result in an interface specification such that in Figure 4.

According to the graphical SDL syntax, microSynergy visualizes the applications deployed on each micro controller as rectangles shows an SDL model with our sample Heater Controller component and two additional components Alarm Control and Lighting Control. In order to combine these distributed micro controllers to a collaborative network, we need a mechanism to dispatch external signals between them. Note that in the general case, we cannot simply map in-gates to out-gates, because we want to provide mechanisms for a-posteriori integration of components. This means that we would like to be able to integrate micro controllers in distributed applications in a way that were not anticipated during their development. Consequently, there is the demand for more sophisticated connectors than simple channels or pipes for signals.

\[
\text{Interface} \quad \text{HeaterController;}
\]
\[
\text{In Signal} \quad \text{setLow()}, \text{setNorm()}, \text{turnOff()}, \text{setTemp}(\text{integer});
\]
\[
\text{Out Signal} \quad \text{turnedOn()}, \text{turnedOff();}
\]
\[
\text{Endinterface} \quad \text{HeaterController;}
\]

\[\text{Figure 4. SDL Interface for Heat Controller}\]

In our approach, we use SDL to specify and generate these sophisticated connectors. The example in Figure 5contains one such connector named LCN Control\(^1\). microSynergy renders connector differently from application components (dashed border) to make it easy for the developer to distinguish between both concepts. Note that the developer is free to specify more than one connector blocks serving different purposes. We have chosen SDL channels for connecting connector component to application components. Channels are represented as (directed) lines in. Rectangular brackets are used for specifying the signal names that travel in each direction. The tool assures that these names correspond to names of in- and out-gates of the connected application components.

\[\text{Figure 6. Specification of connector LCN Control}\]

For our example, let us assume that the developer of the Alarm Controller used microCommander to deploy an in-gate component in such a way that a signal \(\text{armed(stat)}\) is send whenever the Alarm Controller is switched on or off. Note that in this example \(\text{stat}\) is a parameter (with value 1 or 0) representing the status of the alarm controller, i.e., on or off, respectively. Furthermore, let us assume that the network developer now wants to integrate the alarm

\[\text{Figure 5. SDL model of LCN Property Management}\]

Note that, according to the formal SDL semantics, channels delay signals during transport by a non-deterministic duration. Therefore, no assumptions can be made about the arrival time of two signals that have been sent to two different channels at the same time. In addition to channels, SDL provides the notion of signal routes and non-delaying channels for connecting components blocks, i.e. components. We decided to use standard (delaying) channels, because we generate (and execute) Java code from SDL specifications of connector components. Java normally does not satisfy strict real-time requirements. We are aware of the fact that this decision restricts our current approach to applications that do not require real-time communication among distributed micro-controllers. Still, there are a large number of net-centric applications that meet this requirement.

For the actual specification of connector blocks, we use the concept of (extended) finite state machines (FSMs) provided by SDL. This is illustrated in Figure 6 for the example connector components LCN Control. An FSM in SDL consists of states (represented as rectangles with round corners) and transitions (represented by directed arcs). The initial starting state is clearly marked by an oval. In contrast to other state machine models (e.g., UML State Charts), transitions are always drawn from top to bottom and states can have multiple occurrences in the diagram.

\[\text{1 The abbreviation LCN stands for Local Component Network.}\]
controller with the heater controller of the property. The idea is to switch the heater to low temperature mode whenever the property’s residents are absent, that is, whenever the alarm control is armed. This scenario requires a mapping between the armed(stat) signal of the alarm controller and the setLow (resp. setNorm) signal of the heater controller.

Figure 6 shows that such a mapping can easily be created using FSMs. The SDL symbol for an in-going signal is a rectangle with a cut out (“in-going”) triangle. Inverse to this, the SDL uses a rectangle with an “out-going” triangle to specify an out-going signal. The left-hand side of Figure 6 specifies that if an armed(stat) signal occurs while the LCN is in state unnamed, the LCN will change states to armed if the condition stat=1 holds. In this case the desired signal setTempLow is generated. The middle part of Figure 6 shows an analogous specification for setting the heater control mode back to normal whenever the alarm controller is unarmed.

Note that signals of type armed(stat) are created whenever the alarm controller is switched on or off. Hence, initially, the LCN Controller does not have information of the status of the alarm controller, until the first status switch has occurred. Therefore, we have specified a signal statusAlert? which is initiated by the LCN Controller at startup. StatusAlert? triggers the alarm controller to publish its current status, i.e., to send an armed(stat) signal. Such a status enquiry can easily be implemented as a convention of all net-centric micro controllers by configuring a dedicated in-gate component with microCommander. Note that this can be done without prior knowledge about the specific networked applications the controllers will participate in.

The right-hand side of Figure 6 shows another example for networking the alarm controller with the lighting control system. Here, microSynergy allows for treating the light switches as additional sensors for the alarm system: an alarm(room) signal is raised, whenever a light is switched on and the system is armed. On the other hand, the light is automatically switched on in a room where the alarm controller has detected an alarm. Note signal symbols with a double triangle mark priority signals in SDL (e.g., alarm(room) in Figure 6).

There are many additional SDL modeling concepts that we currently do not cover within microSynergy, e.g., macros, procedures, exceptions, etc. Still, even the currently supported subset of SDL is powerful enough to create flexible mappings between embedded application components running on distributed micro controllers.

The generation of executable code from the described subset of SDL is straightforward. Similarly, code can be created for routing sent signals to the different application blocks, based on their interfaces and the specified SDL connection diagrams (cf. Figure 4 and). The generated Java code is deployed and executed as part of the microServer component depicted in our architecture in Figure 8. Note that generally only one microServer instance exists for each LCN. Any Java-enabled device with the required hardware ports can serve as the host for the microServer, e.g., an inexpensive Java micro controller.
// start up
Signal sig = new Signal("StatusAlert?");
LCN.send(sig);
status = "unarmed";
while (1)
  signal=signalQueue.next();
  switch (signal.Name())
  case "armed" :
    if (status=="unarmed") {
      if (signal.IntParam(1)==1) {
        sig = newSignal("SetTempLow");
        LCN.send(sig);
        status = "armed"
      }
    } else {
      status = "unarmed";
    }
    else if (status=="armed") {
      if (signal.IntParam(1)==0) {
        sig = newSignal("SetTempNorm");
        LCN.send(sig);
        status = "unarmed"
      }
    } else {
      status = "armed";
    }
  case "alarm" :
    ....

Figure 8. Generated Java Code

5 GLOBAL NETWORKING OF EMBEDDED COMPONENTS

Network evolution is one of the major challenges that have to be resolved in a successful approach to integrating global component networks (GCNs). Today, there exist a number of distribution middleware products, e.g., CORBA, DCOM, DCE, Java/RMI, Jini, etc [Emmerich, 1999 #92].

In our opinion, the concepts provided by Jini are most suitable for networking highly dynamic networks in distributed embedded applications. Jini technology has been publicly released in January 1999 by Sun Microsystems [Edwards, 2001 #75]. It has been built on top of Java technology. Thus, it is actually not a middleware in the traditional sense, because it does not deal with heterogeneous programming languages like, e.g., CORBA. Rather, it is a connection technology that has specifically been developed for Java.

At first sight, choosing a language-specific platform for integrating distributed components might appear unwise. However, by building on top of Java, Jini technology is leveraged by all modern software engineering concepts that have been built into this net-centric programming language, e.g., portability, mobility, component-orientation, introspection, security etc. Given the omnipresence of Java execution environments on all kinds of platforms starting from micro controllers to workstations, trading language independence against the benefits attached to Java technology appears to be a small sacrifice. Java is clearly about to become the dominant integration technology for global component networks (GCNs) in the foreseeable future. Even for LCNs Java has become increasingly relevant. However, depending on the specific requirements and constraints of particular LCNs, application-specific protocols and platforms will remain important in this area. Still, Java/Jini nicely interfaces with these other emerging network technologies like Bluetooth (proximity-based wireless networking), JetSend (intelligent service negotiation), HAVi (Home Audio-Video interoperability) [3].

A central feature of Jini is that it provides mechanisms for dynamically building and evolving distributed communities by plugging components in and out the network. Each community has a Jini Lookup Service which acts as a broker between available Jini services and their clients. Generally, a GCN includes many such lookup servers. When a component that offers or requests Jini services is plugged into the network, it tries to find at least one lookup server. This is typically done using the Jini multicast request protocol [4].

Figure 9. Jini Gateway to the Network

In a future version of microSynergy, we intend to use this protocol to register the microServers of each LCN with at least one Jini lookup server. Note that the encircled Int Figure 9 in stands for a registered interface of an LCN. Analogously, local microServers can query the Jini lookup service for the interfaces of all LCNs in their “network
neighborhood”. With this information, the developer can use techniques similar to the approach presented in Section 5 in order to federate several LCNs to global networks.

6 DEALING WITH EVOLUTION

Jini’s discovery service has a counterpart so that resources that are automatically allocated can be automatically deallocated. In contrast to other distribution technologies, Jini service objects cannot be used for an indefinite period of time. Rather, Jini introduces the concept of resource leasing, i.e., resources can be reserved (leased) for a certain time frame only. When this time has expired and the service is still needed, the lease has to be renewed. This simple but effective concept provides Jini networks with a sort of self-healing quality, because proxy objects for services that become unavailable will expire over time. Moreover, Jini clients necessarily have to take into account that services can become unavailable. The leasing mechanism coincides with Java’s built-in garbage collector so that it eventually removes all traces of services that have become unavailable. This improves stability, security and efficiency of resource use.

7 CONCLUSION AND FUTURE DIRECTIONS

Our research partnership with Intec Automation is ongoing. Intec intends to exploit the generated intellectual property and turn microSynergy into mature and well-defined products. It is important to note that microSynergy is just the beginning of a whole paradigm in object-oriented design and implementation for the embedded systems market. microSynergy can provide a simple integration solution along with the infrastructure for dynamic networks of embedded systems. The networks and the components on the networks can collaborate with each other and evolve in a seamless fashion. SDL provides the graphical ability to integrate components quickly and easily while checking for validity and consistency of design. Jini’s discovery and leasing concepts allow new hardware and software to be added to and removed from the networks in a seamless and transparent fashion. When components depart from the network all resources are automatically cleaned up leading to better resource usage, security and stability.

Potential real world applications are being realized daily. microSynergy is still in its early stages of development. Industry has already shown significant interest. An example of which is the Herzberg Institute of Astrophysics, where a representative suggested a large real time network for the linking of many telescopes used to form a giant telescope. Their requirements include real time connectivity, ease of development and maintenance as well as low cost.

8 ACKNOWLEDGEMENTS

We thank Intec Automation and the Advanced Systems Institute of British Columbia (ASI) for funding this industrial research partnership.

9 REFERENCES

Call Control Component implementing converged Telephony-Internet networks

Paolo Falcarin
Dipartimento di Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 Torino, Italy
+39 011564.7091
Paolo.Falcarin@polito.it

Patricia Lago
Dipartimento di Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 Torino, Italy
+39 011564.7008
Patricia.Lago@polito.it

ABSTRACT
Global telecommunications are in the early stages of a fundamental transformation, in which telecommunication networks and information technologies are yielding converged networks named Next Generation Networks (NGN). In this context this article presents a call control component able to handle calls between a telephone and a PC, accordingly to actions decided by the application layer that sets up and redirects calls between terminals over heterogeneous networks like Internet and PSTN. This goal is achieved through software adapters offering APIs porting standard protocols like Media Gateway Control Protocol and Intelligent Network Application Protocol (INAP), to CORBA-based software components.

Keywords
Call Control, MeGaCo, SIP, INAP, Parlay, Media gateway.

1 INTRODUCTION
The greatest benefit of the new trend of telecommunication systems is the openness of interfaces and protocols that permits interaction among components implemented by different vendors or service providers. In fact, past telecommunication scenarios were based on e.g. traditional circuit switching that bundles proprietary solutions and is generally providing voice-centric services. On the contrary, Next Generation Networks (NGNs) offer a decoupled switching architecture, with multi-service support over a common media-transport layer, lower cost implementation, and reuse of 3rd party components.

Also, this impulse in telecommunications is producing a rapid change within telecommunication services. Customers’ requests for services simple-to-use, permitting to communicate through any kind of terminal, and at the same time customizable, is always increasing.

To permit communication anywhere in a transparent way, we developed an architecture (and in particular a set of components) that operate communication over hybrid converged networks (in particular PSTN and Internet), and that carry service logic to customize this communication routing according to customer needs or provider network resources. These components (supporting Call Control) are based on distributed object-oriented technologies, and encapsulate the logic to treat converged networks as legacy systems, and provide standard interfaces for multi-provider inter-operability.

2 OBJECTIVES AND CONTEXT
This work focused on the development of a call control component able to establish, manage and finalize calls among a telephone and a PC, according to the actions “decided” by the application layer. The call model has to manage in a homogeneous way the characteristics of the different communication protocols used by the involved physical network resources. In this respect, the implementation has been preceded by the analysis of different protocols for standard communication like INAP [3], SIP [9] and MeGaCo [5]. The chosen programming language was Java, and the implemented software components are: the Call Control server (CC) and the Virtual Presence 1 Call Control server (VPCC). The project notation is OMG/UML  [6], extended with the definition of OMG/IDL interfaces. In tight collaboration with Telecom Italia Lab  [10] researchers, we partially reused some interfaces defined in the Generic Call Control Service (GCCS) Parlay 8 specification and we extended related service diagrams and state diagrams to fulfill our architecture requirements. The internal architecture of the two servers has been developed by considering also issues

1 Virtual Presence [4] is a service implemented in the context of the European Eurescom Project P909 “Enabling Technologies for IN evolution and IN-Internet integration”  [2]. It permits users to personalize the way they can be traced on hybrid converging networks, by use of User Profile.
like concurrency management and real-time constraints, efficiency in the management of multiple concurrent calls, and deadlock-safety. At last, these components communicate through Orbacus, a CORBA [1] implementation made by OOC [7].

The components of the complete architecture are depicted in Figure 1. They can be classified in four categories: network resources, adapters acting as bridges between specific network resources and the architecture core, services made of Call Control components, and applications.

![Figure 1: Architectural Overview](image)

In particular, the components directly involved in the communication with the CC are: SCP and the Media Gateway (MG) with the respective adapters.

*Service Control Point* (SCP) manages real-time signaling in the Intelligent Network. Its behavior depends on its programmable service logic, which has been used in the Project to expand PSTN domain to IT, e.g. to ask an Internet application to resolve phone numbers in the PSTN-IN domain, according to customized preferences stored in the application layer. Exporting SCP features in our architecture is possible by using an adapter (see IWU-SCP) that bridges between SCP and the architecture, and that exports the commands and events of INAP protocol to IDL interfaces, implemented partly by this adapter and partly by CC.

*Media Gateway* (MG) converts media provided in one type of network to the format required in another type of network. E.g. a MG can terminate bearer channels from a switched circuit network and media streams from a packet network (e.g., RTP streams in an IP network). This gateway is capable of processing audio, video (alone or in any combination) and full duplex media translations. A MG needs a Media Gateway Controller (i.e. the Call Control) to receive the correct commands, and send events notification. Commands and events adhere to the MEGACO/H.248 standard protocol that models a call as a context containing terminations. These represent sources and/or sinks of one or more media streams. Interaction between CC and MG is obtained through a MEGACO Adapter that exports a set of MEGACO-like IDL interfaces in order to interact with CC. The choice to specify an API founded on a protocol standard based on the possibility offered by this solution to make Call Control independent from the interfaces with the different protocols used by gateway providers.

On the VP-CC side, communication involves the *Invitation Handling component* (IH). The IH manages communication requests (invitations) from/to the IP domain, by realizing a conversion among the signaling protocol messages to invite users to a service session, and the method invocations on the component interfaces inside the service layer. IH connects IP-terminals by means SIP, an application-layer protocol used for establishing and tearing down multimedia sessions, both unicast and multicast. It has been standardized within the Internet Engineering Task Force (IETF) for the invitation to multicast conferences and Internet telephone calls. A user is addressed by using an email-like address “user@host”, where “user” is a user name or phone number and “host” is a domain name or numerical address. Each new SIP transaction has a unique call identifier, which identifies the session. If the session needs to be modified, e.g. for adding another media, the same call identifier is used as in the initial request, in order to indicate that this is a modification of an existing session. IH has two basic functions: listening for in-coming SIP messages forwarded by a SIP proxy server, and sending SIP messages accordingly to user actions defined by the Application Layer. The SIP application on client-side also starts appropriate audio-conferencing applications according to the session that has been established.

### 3. CALL CONTROL

Call Control (CC) offers two kinds of functionality: a Parlay-like API that offers service interfaces for applications; multiple specific API towards network layer, in order to receive events notifications from different network resources, according to standard protocols used. CC also provides an abstract call model that can cover the essential features of all the call models used by the underlying network resources.

Virtual Presence Call Control (VPCC) has to be intended like an extension of the Virtual Presence application used to implement application-side Parlay interfaces and IH interfaces used to receive invitation messages from IP network through IH component. Inside the CC other two types of objects are used to model the concept of synchronous call between two terminals:

- Class Call represents the information related to the
execution of a call.

- Class Call-Leg represents all the information related to a terminal involved in the call.

Both classes are described by a state-diagram, which reuses Parlay Call and Call-Leg states. Extension of these state diagrams have been produced after insertion of a new kind of transitions, represented by method invocations and parameters values described in MEGACO, INAP and IH interfaces implemented by CC and VPCC. Therefore CC and VPCC manage hybrid scenarios, because they set up calls between heterogeneous domains like PSTN and IP networks, e.g. both “Phone to PC” calls and “PC to Phone” calls. However CC and VPCC can handle classic homogeneous calls between two phones or between two PC connected by means of audio-conference applications. Another important feature is the uniform propagation of events notifications coming from different networks (like no answer, busy, disconnection, error) toward the application layer that can decide how to do depending by call-handling user-defined policies: e.g. application can decide to redirect call to another terminal (phone or PC) or it can decide to close the call session. VPCC interprets actions decided by application and executes these commands interacting with CC. In figure 2 is represented the “Phone to PC” sequence diagram.

In Figure 2, user A composes the telephone number of a subscriber to the Virtual Presence service that will have the structure XXXX-ZZZZ, where XXXX represents the prefix that identify this service on national staircase (as the number 800 for free-phone) while ZZZZ is the Personal Number that it identifies the service subscriber. Through the SCP this information, together with the caller number, is passed to IWU-SCP and then forwarded to CC through the method eventNotify() of its INAP-like. Now CC creates the object Call, assigning to it a unique identifier, and the object Call-Leg related to the caller and it transmits these data to the VP-CC invoking the method callEventNotify. VPCC invokes the method communication_evt() on the Virtual Presence that, after querying its private user database, decides how the subscriber wants to handle this kind of calls.
The actions decided, and, in this case the destination IP address, are returned to VPCC that has to set up the hybrid call between a phone and a PC. First of all invokes the method Invite() on the Invitation Handling interface, asking to it to send a SIP INVITE command (INVITE) to the IP address returned by the application layer. In the case that the user B accepts the invitation, the audio-conference application is launched on his PC and the message OK 200 is sent to the component of Invitation Handling. Now it will invoke the InvitationResponse() method on VPCC, to communicate to it that the called user has accepted the invitation to establish a synchronous communication. Now VPCC will ask CC to set up the hybrid call, invoking the method routeCallToDestination_Req() that triggers the creation of the second Call-Leg related to the called IP address and requires to the media-gateway to establish a Phone-to-PC call.

For first thing the CC invokes the method Modify() on the MEGACO interface of the MeGaCo Adapter in order to reserve a gateway phone-port for the arrival of a call from the net PSTN; this port number returned as a notify() parameter will be forwarded to the caller via the commandConnect() invocation on the INAP interface of IWU-SCP, in order to redirect the phone call to the gateway phone port. CC. With first Add(), CC commands to media gateway to listen to events of connection or disconnections from PSTN network. With second Add(), CC says to Media gateway to listen to events coming from IP network. Meanwhile the audio-conference application has already been performed, after invocation of invite() from Invitation Handling component, telling to called user’s PC to connect with media gateway IP address. Therefore, after that both network connections have been set up, CC receives notify(IP-CONNECT) and notify(PSTN-CONNECT) invocations from MeGaCo Adapter. Since the gateway used by the implementation has a unique IP address, the IP termination allocated is always the same one and it is immediately passed to the called PC as an output parameter of invitationResponse() method. When two Call-Legs are both connected, CC invokes routeCallTo-Destiantion_Res() on VPCC interface to notify that call is active. VPCC has now to listen to the events of disconnection from the two connected terminals, and therefore it will release resources.

4. FUTURE WORK
This architecture can be enhanced in different ways: Call Control could be dynamically replied on several servers in order to support a major traffic load. Every CC could handle more than one Media Gateway, in order to obtain a more scalable architecture, able to reserve resources depending on traffic and QoS requirements.

Second, we are planning to include new network resources (e.g. supporting unified messaging) in order to handle asynchronous communications too and to implement bi-directional translation of voice and text-based messages (like e-mail or SMS).

ACKNOWLEDGES
The results presented in this paper have been partially supported by the EURESCOM project P909-GI “Enabling Technologies for IN Evolution and IN-Internet Integration”. The authors would like to thank all EURESCOM P909 participants.

REFERENCES
Reengineering for Evolution of Distributed Information Systems

Holger Giese, Jörg P. Wadsack
Software Engineering Group, Department of Mathematics and Computer Science
University of Paderborn
Warburger Str. 100, D-33098 Paderborn
Germany
[hgmaroc]@upb.de

Keywords
System evolution, reengineering process, distributed information systems, middleware, schema integration, application integration, internet access

1 INTRODUCTION
The growing e-commerce market engender the need to open the existing, in local area networks utilized, information systems (IS) to the web. This requires the well understanding of the existing IS and makes reengineering a central aspect. However, most of today's reengineering activities and results address monolithic systems such as the centralized systems developed in the seventies. The required methods to reach the current state of reengineering for such systems were developed over several years. For the more recently developed distributed IS (DIS), the integration into the internet requires new adjusted reengineering techniques. The high complexity and the rapid evolution of DIS nowadays requires continuous DIS reengineering (DISRE).

In this paper, we conceive evolution as any change in or of the DIS and emphasize the following three actions:
• union of multiple running (D)IS to a single DIS,
• extension of a DIS by a new part or
• reduction of a DIS by discard a part
Union and extension induces the idea of integration, whereas reduction entails “reverse integration” which can be seen as a variant of decomposition. In this context, we interprete integration and reverse integration in evolution of DIS with focus on a suitable middleware concepts.

In this paper, a systematic DISRE process based on an analysis of available integration concepts and technologies is developed. The application of existing and required new reengineering techniques in such a process is sketched.

The rest of the paper is structured as follows, in Section 2 we discuss the advantages and restrictions of the use of middleware of DIS. In the next section (Section 3), we propose a DISRE process, considering the trade of between existing and new (adjusted) reengineering techniques. Finally, in Section 4 we draw some conclusions.

2 MIDDLEWARE FOR DISTRIBUTED INFORMATION SYSTEMS
To clarify the architectural scenario we assume a three-tier model. On the left side of Figure 1, we have the schema layer of the model. We choose the term schema to describe the information (data) structure of our DIS. Furthermore, we select distributed databases (DDB) as more concrete case study, there the term schema fits better. The application layer contains all the functionality to manipulate the informations stored in the schemas. Finally, we have the WWW layer, which represents the interface to the real world, i.e. users.

Integration and support for evolution of distributed information systems by means of an additional middleware can be done in various ways. To structure and analyze the advantages and restrictions of different approaches we consider the two following extreme views: (1) schema integration (SI) which provides a consistent virtual schema located between the schema and application layer containing the integrated schemas; (2) application integration (AI) which results in an addition virtual application layer in-between the application and WWW layer.

In Figure 2 the SI approach for integration and evolution is presented. The different application specific schemas are integrated into an overall virtual schema (grey) which is served by a single homogenous distributed database management system. System evolution in form of an additional application therefore would result in an updated virtual schema and a new application (dashed box). Batini et al [BLN86] present several methodologies for (database) SI. An approach to preserve semantic updates in SI is sketched in [VW94].
The AI scenario in contrast would integrate an additional application without modifying the different schemas, cf. Figure 3. Instead, the virtual application layer (grey) is used to coordinate the applications as required. This additional application layer does further permit to use different heterogeneous and physically distributed database management systems when support for distributed transaction processing standards [XA94] is present. For example extending the DIS by a new application with own new schema, cf. dashed parts in Figure 3.

The required coordination however has to be realized by code in the virtual application layer. The identified AI strategy is strongly related to enterprise application integration (EAI) [Lin99] approaches, but we assume that the applications composition does not involve event processing as realized either by polling on a shared database or using specific application APIs.

The proposed middleware layer for application and information system integration has to be build on top of available middleware technology. The common solution for database integration is distributed transaction processing (DTP) [XA94] as provided by transaction monitors [Hud94, Hal96] and middleware transaction services [JTS99, OTS98]. Another more scalable solution is reliable messaging [Lew99, Hou98] which results in a reliable asynchronous processing scenario.

<table>
<thead>
<tr>
<th>evolution schema</th>
<th>SI requires update for virtual schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>evolution application</td>
<td>AI requires update for virtual application</td>
</tr>
<tr>
<td>consistency</td>
<td>SI failures result in loss of data while inconsistencies are possible for AI failures</td>
</tr>
<tr>
<td>redundancy schema</td>
<td>SI has to exclude redundancy in the schema while AI control it at application level</td>
</tr>
<tr>
<td>redundancy application</td>
<td>SI does not address redundancy at the application level while AI can help to identify redundant functions which may be united in next evolution steps</td>
</tr>
</tbody>
</table>

Table 1: Evolution, consistency and redundancy

The AI approach requires the realization of coordination activities. A suitable solution is to employ available middleware technology. For the SI approach the same technology can be applied when the virtual schema layer is realized by code, however, also database technology like views can be used. Depending on whether a code or strict database solution has been chosen, the flexibility of the virtual layer permits variations to a great extent.

To compare both extreme concepts we further consider their impacts on evolution, consistency and redundancy in Table 1. The observations reveal that both strategies are rather contrary. While SI focuses on consistency and therefore try to exclude redundancy, AI avoids the integration of schemas and instead demand the proper coordination of the applications.

The specific aspects of distributed systems such as autonomy, heterogeneity and scalability are addressed in Table 2. The AI approach additionally permits integration of arbitrary legacy systems. When the virtual application layer provides the same interfaces as the applications the degree of transparency ensured by the SI strategy is also possible for a AI solution. In practice, the provided interface may include synthesized functionality and therefore a clear separation between the virtual application layer and added functionality is often difficult.

The differences of both approaches can be further clarified by considering whether state or transitions of the resulting system are composed. While the SI approach provides a unique state (virtual schema) and relates all application specific schemas accordingly (mapping), the AI approach permits partial states (the application specific schemas) but ensures, that all via the application code initiated state, transitions are coordinated in an appropriate manner.

The goal of information system integration is the linking and interconnection of beforehand independent systems. The resulting coupling, however, also result in a less reliable system. When the organizational coexistence is acceptable or even required, the AI approach and even temporary inconsistency may be seen in favor. If in contrast data consistency has highest priority, SI has to be employed. The AI solution may include redundancy but ensures losslessness while the SI approach can result in a possible loss of data but ensures no-redundancy. Therefore depending on whether the reengineering activity is not considered as a closed job, both approach will be rated quite differently.

In practice, which approach fits best, varies for each fragment of the system and therefore a compromise which merges both strategies is often most valuable. Such a hybrid solution should exploit the advantages of both strategies for a system specific solution.
3 A PROCESS TO REENGINEER DISTRIBUTED INFORMATION SYSTEMS

The drawn conclusion in the previous section obliges us to consider SI as well as AI in parallel. Investigating all such possible hybrid integration scenarios would however be an impracticable task. For this reasons our process (depicted in Figure 4) starts with facile investigations.

In a first step, a (light) reverse engineering activity is applied to the application tier, i.e. we analyse the interfaces from the user view (step 1.1). The result is a simple directed call-graph from the basic user operations to the main parts of the schemas. In parallel, in step 1.2, a data structure reverse engineering step is done on the different schemas. This step is limited in a premier time in extracting entities and obvious relationships.

Based on a comparison of the call-graphs and when the analysis of their offered functionality indicates a sharing between both applications, we check pairs of schemas on the degree of worth for their integration. This second process step provides us with a schema & application clustering. Here it might be necessary to have further information than the ones provided by the first steps. This leads on the one hand in step 2 to the need for iteration and on the other hand we have to provide techniques which allows more profound analyses. Those have to be program understanding as well as data structure recovering methods. The observed degree of schema compatibility permits to estimate the required integration effort. Thus, we can decide where SI is appropriate and where not.

To perform step 3.1, i.e. SI, we need the complete structure of the affected schemas. Two existing schemas reverse engineering approaches which adress the problem of an entire structural schema recovery are DB-Main [EH99] and Varlet [Jah99]. Moreover, we have to recover relationships between the schemas. This inter-schema relationships have to be considered and can be use directly for the integration, i.e. the construction of a virtual schema. But for a complete and consistent integration we need information about data (schema) behaviour, which induces again more profound application understanding.

Alternatively, AI is done in step 3.2 of our process. The central problem is to identify all applications which are concerned by concrete user interactions. Furthermore, we have to cover all possible user interaction activities. This implies the need of full knowledge about the interfaces, because missing only one application for one user interaction leads automatically to partial inconsistencies. To ensure the completeness of the reengineered information about the interfaces, we need, beside data structure (schema) and behaviour knowledge, information about the inter-schema relationships.

Figure 4 DISRE Process

An early overview of reverse engineering techniques for program understanding is presented in [BMG+94]. Possibilities and requirements for tool interoperability where recently discussed in Dagstuhl [EKM01].

According to the three layers of Figure 1, we propose an example to illustrate our approach. Assume, we have two insurance companies which want to fuse. The first is the InPrivate Ltd which only offers contracts for private insurances. Second, we have the BuInCo which only concludes agreements with business people and enterprises. Both companies operate DIS which should be integrated. Figure 5 depict a simplified excerpt of the two original DIS of the companies.

In the following, we apply our process to this insurance fusion example. This is of course a simplified view of the example due to lack of space. In Figure 6 we have two databases, CustomerDB and ContractDB, and an applications contract management for the InPrivate Ltd. BuInCo has also
two databases, PurchaserDB and IndentureDB / AgreementDB, and two applications the client administration and the agreement survey. The IndentureDB schema is for insurances for business people and the AgreementDB is to store information about company agreements. In addition, we have two kind of portals, one for the company employees (EmployeePortal) and the other for business clients (BusinessPortal).

After performing step 1.1 and step 1.2 of our process, the comparison of the applications contract management and client administration indicates overlapping between databases CustomerDB and PurchaserDB. A more detailed analysis of the two schemas reveals large similarities in the manner of storing client information. This entails an integration of those two schemas to a client schema. Consequently, following Figure 2, the new client schema has to be adapted to the access from applications contract management and client administration.

Overlapping for private insurance contracts and agreements for business people is indicated by comparing applications contract management and agreement survey. Further investigations in the corresponding schemas reveal that an integration of the ContractDB schema and the IndentureDB schema could result in risks for the fusionned companies, e.g. a private person may benefit from business people advantages.

In contrast, the ContractDB schema and the company agreement schema parts of the AgreementDB schema are easily integrable, since they have no overlapping at all. But in this case integration makes little sense, because they would coexist and generate a superfluous management overhead. For the given example no further suitable schema pairs for SI can be identified.

AI makes sense for contract management and agreement survey because this two services are planned to be operated in a net-wide environment. A web access for client administration in contrast will be a considerable risks. For these reasons net-based application only encapsulates contract management and agreement survey.

Finally, for the case of an extension, we consider adding a customer administration application. The overlapping of functionality with the client administration indicates that SI employing the derived client schema is an appropriate solution. Note that the EmployeePortal is connected to the client administration application to ensure that the employees have the same access to the application as the business clients.

4 CONCLUSIONS

The proposed process to reengineer DIS and the guideline of their evolution should, beside the identified complexity for SI or AI, consider whether a later consolidation is planned. For the SI approach the later merging is rather straightforward based on the given combined virtual schema. For the AI strategy, however, such a merging requires that schema and application code have to be combined while no reuse for the integration efforts in form of the virtual application layer is guaranteed.

While the phenomena of distribution is more naturally covered by the AI approach, the SI strategy links the modules more tightly and therefore help to avoid serious problems with redundancy. Besides the discussed technical aspects, organizational structures and requirements are also relevant for an appropriate solution with respect to the degree of coupling. Therefore, whether SI or AI is reasonable, is not only a matter of technical feasibility.

REFERENCES


An Integrated Approach for Software Documentation Based on MSR Standards

Jochen Hartmann
BMW Group
Munich, Germany
Jochen.Hartmann@bmw.de

Abstract

The engineering process for developing complex systems like engine management covers multiple phases handled by many different employees, departments and companies. The problem is, each sub-process requires a set of documents and data which are not necessarily known by the other participants. This paper outlines an integrated approach for software documentation based on MSR MEDOC (MSR Engineering DOCumentation) standards. MSR is a consortium of several German companies with the goal to support cooperative developments between car manufacturers and their electronic system suppliers [1].

1. Introduction

Nowadays engineering tools produce document views (flow charts, sequence charts, etc.) using report generators or interfaces to common word processors often proudly marketed as “documentation interface.” In a broader sense, the information in the engineering base (function names, relations between functions, descriptions of functionality, etc.) itself and not only the exported presentations must be treated as documents. This means that documents are mostly incomplete and therefore the developer has to read several documents to get the information he needs for his daily work. If views are required which reflect information from more than one engineering tool, then either some connection of the tools is required, or the documents must be assembled manually.

There is a gap between the presentation view and the engineering base that can be closed by introducing a middle layer, the document base. This layer must be built upon a standard data format which can be used to generate all the required outputs (document views) in the presentation layer while still having the power to keep the semantics of the engineering base at least to some extent.

This layer is the domain of SGML/XML, which is easy to create, able to define powerful data structures and not too hard to read by humans as well as by machines [8]. To improve the understanding of a system, new views can be generated by combining information produced by different tools. Furthermore, a standardized structure makes it possible to hand over information to a following system like the product data management (PDM). A common structure is basis for data exchange between the manufactures and their suppliers. This is particularly necessary if different companies are developing the same system commonly.

This paper shows how the MSR standard can used for documentation of software systems. Section 2 describes the underlying software development process. The MSR DTDs used in the documentation process are presented in section 3. A description of the documentation approach is given in section 4. Section 5 reports on some of the early experience of applying and using the MST standard for the documentation of real time systems.

2. Software Development Process

Many of today’s electronic and mechanical devices use embedded software, which in many cases has taken over what mechanical and dedicated hardware systems used to do. Indeed, embedded software appears in everything from telephones and pagers to systems for medical diagnosis, climate control, and manufacturing. Its main task is to engage the physical world, interacting directly with sensors and actuators [6].

To develop such kind of systems, varied documents and information have to be handled. For the specification of the whole system, for example, data of the sensors and actuators are necessary. Furthermore, for the documentation of the system functions requirements have to been taken into consideration.

Complex systems like an engine management units are developed nowadays by several developers from several companies. That’s why several kinds of documents have to be exchanged between the manufactures and the suppliers. Therefore, a documentation process has to support several manufacture-supplier-scenarios.
The underlying principle of the presented approach is a distributed development of engine management software where several developers of different companies are working on the same project. The software development based on from literature well known V-model [4][5]. This model is the nowadays-accepted standard for software development for embedded systems. The V-model laid down the totality of all activities and products, as well as all states of products and logical relationships between product and process during development, maintenance and changes of software systems. The V-model describes the development process from a functional point of view and does not go into special organizations forms, which exist, in the involved companies.

Before carrying out a project, the tasks, developers and organization units defined in the V-model have to be fixed. Each of these tasks or activities has to be documented correspondingly. The V-model is divided into the field of system construction, quality check (including testing), configurations management and project management. For each of these development steps interfaces for information exchange between the manufacture and the supplier have to exist. Therefore it is necessary to demand of documentation systems to provide these interfaces between the several development steps and the organization units.

The first step in each development process is the definition of the function requirements. This includes registration of new functionalities as well as changes of existing functions. For this tools like DOORS [3] are developed, which permit to register function requirements and to follow their realization. These function requirements are describing the physical background of a new functionality or a change in the software and therefore they are part of documentation. At this passage the “why” of a software solution is described. The reasons are stretches from requirements of the market to problem fixing.

In the next phase the consequences of these changes and extensions for the whole system are analyzed and the necessary conditions for the realization defined. A first model of the system was realized in this phase also. Within this step the functionality was described from a system point of view. In the following phase the requirements were specified in more detail and in the implementation step the requirements were realized in code directly. This could be done by coding manually as well as by use of graphical programming tools like ASCET SD from ETAS [2]. This tool is able to generate C-code for the target system automatically. Furthermore this tool offers the advantage to use the graphical description of the functionality for documentation also. In this part of the documentation the “how” of the presented software solution was described.

In the last phase of the software development the system was calibrated with tools like INCA [2] and tested extensively in the lab as well as in the car. The test and the corresponding results have to be documented also. A new software change has to be produced if there are any errors detected. Therefore a link between the test approach and the requirement management system has to be implemented.

3. Used MSR DTDs

What is MSR? The role of MSR is not to conduct any standardization of the systems or their features that are described with MEDOC (MSR Engineering DOCumentation). MEDOC supports the use of (inter)national standards and in-house norms, as well as non-standardized norms, for the description of systems of data relevant to the documentation of development process [9]. For a multi-language documentation the MSTR DTDs can be configured for multilingual operation. To use the multilingual DTD configuration the DTD switch “multilinguality: YES or NO” have to be set. Additionally to the multilingualism the MSR standard supports a distributed development and the exchange of documents and code files between the developers.

For exchanging documents a special CATALOG-DTD was developed, which allows to exchange any kind of configurations like code or documentation files. To be independent from configuration management systems meta information were defined for exchanging program and documentation configurations. As shown in Figure 1, a sub-class of this catalog file was used to bundle up several documentation files for a common formation process.

The engineering process normally produces several kinds of documents in each phase of development. Starting with the specification of new functionality, software development including function development and system configuration and last but not least the system calibration and testing. Therefore a set of MSR DTDs is used in the presented documentation approach. Results of the early phase, the system analysis, are established in parts of the MSRSW_DTD. Therefore a element "sw-function-def" was defined which can be loaded as a rule by the output of a CASE tool or a system design tool. The MSRREP-DTD together with the element <changes> can be used for function requirements and review protocols (e.g., design reviews).

For the system design, the physical model is derived from the functional logic model. In the MSRSW-DTD the
physical functional model includes the description of the context, the implemented functions and the information flow. In the system design part the compilation from functions into tasks and assigned to computing levels can be described. Information originated in the coding like variable names can be established with <sw-data-dictionary> of the MSRSW-DTD. The test report describes the environment, the application as well as the results of the module test. MSR-REP-DTD can be used for this. Hardware architectures can be described in the MSRSYS-DTD.

4. The Documentation Approach

The basic idea of the documentation approach is to keep the documentation close together to the corresponding software. Both, the software and the documentation are representing a system-specific configuration. This results in use of the Configuration Management (CM) like CONTINUUS system to save the files. For keeping the files up to date a mechanism for version control was developed. Additionally to the text files all corresponding figures are established in the Configuration Management system.

For file exchanging between CONTINUUS and the process for formatting the document in a printable format, e.g. PFD the CATALOG format is used. At this step a consistency check between the C-file and the corresponding document file is carried out. If the version of the document is not equal to the version of the C-file an empty page is issued. The construction of the documentation is included in the MAKE process. This offers the possibility to produce the documentation simultaneously to the corresponding program release.

Afterwards the CATALOG file is fed to a CATALOG interpreter which produces the configuration files for the merger [7] This merger generates a MSRSW instance for the formatting process. This instance includes all documents in the order given by the CATALOG file. Furthermore a mechanism was included to generated different views to generate several kinds of documents for several target groups, e.g. internal versus external developers. In the following formatting process documents in a publishable format like PDF or POSTSCRIPT are produced. Figure 2 gives an overview over the described process.

5. Conclusions

Documentation based on SGML/XML offers a great potential to establish a glue layer in complex engineering processes. Combined with redocumentation and reengineering mechanisms the documentation system could be effective facility to provide the developer with all relevant information. Through the use of a standardized immediate format information generated by different tools can be combined to produce user specific documentation. This offers the possibility to give the developer the information he needs. The closed link to the software development process makes it possible to keep the documentation up to date, which is of great importance for the quality of the documentation.

Acknowledgments

I thank Christian Haug for the discussions on the topic covered by this paper and for all the programming work he has done to carry out this project. I am grateful to my colleagues for the stimulating discussions about the documentations process.

References

Performance Optimization Appraisal on Web-enabling Legacy Applications

Rajagopal. R
Communications And Product Services (CAPS) Group,
Infosys Technologies Limited,
(NASDAQ:INFY)
35, Sardar Patel Road,
Alexander Square,
Guindy, Chennai – 600 032.
Tamil Nadu, India.
+91 44 230 0031
rajagopal_raman@infy.com

Eugene Xavier. P
Education and Research Department,
Infosys Technologies Limited,
(NASDAQ:INFY)
35,Sardar Patel Road,
Alexander Square,
Guindy, Chennai – 600 032.
Tamil Nadu, India.
+91 44 230 0031
eugene@infy.com

ABSTRACT
World Wide Web may not only be the biggest technical revolution, it may also inadvertently spawn a monumental change in the structure of enterprise computing. This change sees Client/Server based applications being usurped in favour of browser-based and thin-client application, so that businesses can take advantage of the considerable capabilities of Internet Technology. Most of the companies face the challenge of adjusting their business to the realities of e-commerce in the most economical way possible. With their hard-earned legacy applications already firmly in place, many companies are opting for solutions that rely heavily upon XML and Java technology because of portability, flexibility and extensibility. By Transactions Reuse, transactions produce data streams, messages and changes, which are reused. Process Understanding and Recovery, are key factors that specialize in the understanding of multiple computing languages through language parsers. This paper discusses the necessity of optimizing the performance while web-enabling Legacy Applications, the key issues involved and attaining solution through XML and Java Technologies.

Keywords
Legacy Applications, Enterprise Java Beans, XML, Distributed application.

1. INTRODUCTION
The enterprise computing undergoes a shift to a three-tier web based computing paradigm. The main strengths and benefits of the three-tier model include platform independence, and consequently, cost-effectiveness. The application server is the centerpiece of the new model. Although there are many definitions circulating today, an application server can basically be described as middleware located between clients and data sources and legacy systems that manages and processes applications in a three-tier fashion [10].

Computer networks are a very effective way to deliver information to large numbers of users virtually anywhere in the world. Although large businesses, institutions and government agencies have been using internal computer networks for decades, the rapid growth of the Internet has propelled many organizations to expand the scope of their networking activities. Instead of only providing information to their own employees over a private, internal network, organizations are now working to exploit the omnipresence of the global Internet to provide information and applications directly to customers, suppliers, distributors and other business partners. These extensions of the enterprise network outside the organization's boundaries have become known as Extranets. Analyzing and understanding the performance characteristics of Extranet applications is a complex topic, involving network design, server operating system performance, application design and application-specific usage patterns.

2. PERFORMANCE REQUIREMENTS
There are very many requirements to be met in terms of the performance of WWW legacy applications of which capacity, throughput, response time, consistency, user expectations, availability, scalability and manageability are the key factors under consideration. Capacity refers to the ability to support large numbers of concurrent end-users. Capacity is primarily a function of the hardware configuration of the server machine. It often varies directly

44
with the amount of memory available to maintain the session information for each client. Many performance analysts try to avoid measuring capacity in terms of some number of inputs such as the number of concurrent users. Instead, they focus on how much work is getting accomplished, for which they use the term throughput. It is much tougher to define the units for work than it is to simply count the number of end-users.

The better way to maximize the amount of work being done is by performing many high-value back-end interactions on behalf of each end-user, rather than incurring lots of overhead doing many low-value interactions with the end-user. Rapid real-time response to an end-user request is a common requirement for transaction-oriented systems. Response time is often inversely related to capacity - the more end-users on the server, the slower the response time, as they all compete for processor and memory resources.

In some instances it is more important that the response time is consistent from one interaction to the next. Consumer-oriented Extranet applications, such as e-commerce systems, must ensure that response times are consistent within a narrow band [3]. User Expectations is an important criterion for determining acceptable response times for a web application. Generally it takes 1 to 10 second response time, and it varies for each and every request.

A typical good response should have only 2 to 4 second response time for every request. Availability is a measure of the robustness of the system. If a component of the web environment is constantly failing or hitting a capacity limit, then the application will have poor availability. Many mainframe-based systems regularly achieve over 99.99% availability. In contrast, many popular web applications have less than 95% availability.

Scalability refers to the ability of a system to grow from a small set of users to a large set of users without encountering bottlenecks that make performance unacceptable [2]. The resource requirements and other performance measures increase linearly with the number of users, as opposed to growing exponentially or hitting a boundary condition.

Some applications may perform well in the development lab with a few users, or even under testing with a few dozen users, but when they are deployed in the real world with several hundreds of users, the system simply breaks down. As part of the performance tuning process, applications often require some redesign before they will scale up to a large user community.

As web applications scale up with multi-server deployments, it becomes crucial that system personnel have the tools to monitor and manage the level of performance being provided. Performance tuning is not a one-time event. Extranet environments are not static. As usage patterns evolve and network or server configurations are upgraded, the performance profile of the Extranet application will be affected. Management tools are essential for tracking performance and giving systems personnel adequate time to respond to trends before they become acute problems.

3. OPTIMIZING WEB PERFORMANCE FOR LEGACY APPLICATIONS - THE JAVA STRANDS

Application Servers offer server-side support for developing and deploying business logic that may be located on the server or more often partitioned across client and server. Enterprises rely daily on server-side business processing, ranging from mainframe transaction systems to Client/Server DBMS stored procedures [1].

Running business processes on the server provides Re-use. A variety of client applications can share the same business logic. Sensitive business logic often includes or manages trade secrets that could potentially be reversed engineered, which refers to "Intellectual property protection". It provides Security of business logic, which means, by leaving the logic on the server, user access can be controlled dynamically revoked at any time.

Security of network communications is being offered by Application servers which allow user of internet standard secure protocols like SSL or https in place of less secure proprietary DBMS protocols. Server-side applications are easier to monitor, control and update those accounts for Manageability. Database intensive business logic often performs much better when located near the database, saving network traffic and access latency that takes care of Performance. Internet clients most often require access to many different business processes that could require substantial network bandwidth and client memory to download all logic to the client.

Multithreaded Java Servlets are an effective replacement for CGI scripts. Performance is significantly better, servlets executes within the address space of a web server. Java servlets create a single process and share across the users by creating threads referred to as multithreading. They provide a way to generate dynamic documents that is both easier to write and faster to run.

Servlets also address the problem of doing server-side programming with platform-specific APIs: they are developed with the Java Servlet API, a standard Java extension [14]. So use servlets to handle HTTP client requests. For example, have servlets process data POSTed over HTTPS using an HTML form, including purchase order or credit card data. A servlet could be part of an order-entry and processing system, working with product and inventory databases, and perhaps an on-line payment system.

With servlets, the logic for generation of the dynamic content is an intrinsic part of the servlet itself, and is closely tied to the static presentation templates responsible
for the user interface. Thus, even minor changes made to the UI typically result in the recompilation of the servlet. This tight coupling of presentation and content results in brittle, inflexible applications. However, with Java Server Pages (JSP), the logic to generate the dynamic content is kept separate from the static presentation templates by encapsulating it within external JavaBeans components [12]. These are then created and used by the JSP page using special tags and scriptlets.

When a page designer makes any changes to the presentation template, the JSP page is automatically recompiled and reloaded into the web server by the JSP engine [11]. While there are numerous technologies for building web applications that serve dynamic content, the one that has really caught the attention of the development community is JavaServer Pages (JSP). And not without ample reason either. JSP not only enjoys cross-platform and cross-Web-server support, but effectively melds the power of server-side Java technology with the WYSIWYG features of static HTML pages.

The Enterprise JavaBeans (EJB) defines an application-programming interface that simplifies the development, deployment and management of multi-tier, cross-platform, distributed object applications [13]. Using the EJB API, developers can focus on writing business logic for middle-tier servers while spending less time coding and testing on the infrastructure aspects of a distributed application. Because each Enterprise JavaBeans component encapsulates an essential business function, developers do not need to know how to write specialized system-level programs that control features such as security and the ability to handle multiple transactions - often tedious and complex tasks.

4. DEPLOYING XML ARCHITECTURE IN WEB-ENABLING LEGACY SYSTEMS

Extensible Markup Language brings so much power and flexibility to Web-based applications that it provides a number of compelling benefits to developers and users: More meaningful searches, Development of flexible Web applications, Data integration from disparate sources, Data from multiple applications, Local computation and manipulation of data, Multiple views on the data, Granular updates and Open standards.

Once data has been found, XML can be delivered to other applications, objects, or middle-tier servers for further processing. Or it can be delivered to the desktop for viewing in a browser. XML, together with HTML for display, scripting for logic, and a common object model for interacting with the data and display, provides the technologies needed for flexible three-tier Web application development [4].

The ability to search multiple, incompatible databases is virtually impossible today. XML enables structured data from different sources to be easily combined. Software agents can be used to integrate data on a middle-tier server from back-end databases and other applications [5]. This data can then be delivered to clients or other servers for further aggregation, processing and distribution.

The extensibility and flexibility of XML allows it to describe data contained in a wide variety of heterogeneous applications, from describing collections of Web pages to data records. Again, since XML-based data is self-describing, data can be exchanged and processed without having a built-in description of the incoming data.

Once data has been delivered to the desktop, it can be viewed in different ways. By describing structured data in a simple, open, robust, and extensible manner, XML complements HTML, which is widely used to describe user interfaces [6]. Again, while HTML describes the appearance of data, XML describes data itself. In the case of a weather map, an average user might want to see just the high and low temperatures and precipitation data. But pilots and others might also want to know about air pressure, dew points and other information.

Since display is now separate from data, having this data defined in XML allows different views to be specified, resulting in data being presented appropriately. Local data may be presented dynamically in a manner determined by client configuration, user preference or other criteria. As XML is an open text-based format, it can be delivered using HTTP in the same way that HTML can today without any changes to existing networks [7].

As XML completely separates the notion of markup from its intended display, authors can embed in structured data procedural descriptions of how to produce different data views. This is an incredibly powerful mechanism for offloading as much user interaction as possible to the client computer, while reducing server traffic and browser response times. In addition, XML lets individual pieces of data be altered with only an update notice, greatly enhancing server scalability as a result of a far lower workload.

XML compresses extremely well due to the repetitive nature of the tags used to describe data structure [8]. The need to compress XML data will be application-dependent and largely a function of the amount of data being moved between server and client. Most scripting language do not involve any compilation, which means that scripts can be edited without any additional steps. This makes the development process much simpler as well as faster to execute.

5. CONCLUSION

The major benefits of extending conventional host-based applications from the prison of proprietary desktops and networks include big financial savings by cutting network and terminal costs, bonus return on the original investment in legacy systems, opportunities for electronic commerce using legacy databases. Through Weblications, businesses can create a presence on the Web and outsource functions
like order taking and servicing to end clients, more satisfied users who get an attractive graphical interface and more integrated and efficient functionality. The performance parameter appraisal done with web-enabling legacy systems illustrates the necessity of implementing various features of XML and Java Technologies in such a transition.

REFERENCES
Experience in Migrating Legacy Systems to the Web

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425 USA
http://www.ics.uci.edu/~wscacchi
wscacchi@ics.uci.edu

Overview

Three issues have been identified as the central focus of this Workshop on Net-Centric Computing (NCC). These are:

- decomposing legacy systems to identify logical components representing essential functionality,
- developing a new Web-enabled system using these components, and
- deploying the new system in an NCC environment.

NCC relies on portable applications running on multiple platforms, mobile data accessed via high-speed network connections and small footprint information appliances for local processing. Of particular interest are holistic techniques for Web-enabling existing applications that integrate various reengineering aspects into a "whole system" modernization process. Furthermore, workshop participants have been asked to contribute and exchange on information related to research and empirical results in areas including:

- Business issues in migrating to the Web
- Integration strategies
- Using distributed component technology in developing Web-enabled systems

Drawing from experience in reengineering operational business processes and systems in a complex enterprise serves as a starting point for the positions taken in this paper [6,7,9,10]. Accordingly, the remaining sections focus on identifying research and empirical results in the areas identified here. However, it should be noted that these positions are based on a view that it is more effective to start from understanding the organizational processes, the people who work within these processes, and the computing systems they use and are willing to adopt. In contrast, it is less effective to start from what is technically possible with NCC systems and then trying to transform an organization and people's work to exploit these new systems.

Business issues in migrating to the Web

In related research that empirically examines case studies of migrating legacy systems and redesigning legacy business processes associated with these systems to Web-based process execution environments, we have found two activities which are necessary to help migrate and transform an enterprise's processes to exploit Web-based solutions.

Facilitate collaborative understanding, redesign and transformation of business processes.

It is becoming very clear that people at work in complex enterprises rarely if ever maintain explicit representations of the business processes. Without an external representations of the process that can be shared and evaluated, process redesign and transformation to any kind of advanced information technology appears doomed to fail [6,7]. Instead, we have found it more effective to provide Web-based process modeling, simulation, and enactment.
environments that enable end-users to participate in and control of Web-based process redesign [7,9,10]. Examples can be provided [5].

**Iteratively prototype, demonstrate and refine a process-driven environment for deploying, evaluating, and refining NCC business processes.**

Much like the preceding claim, Web-based process technologies can also be used to support the iterative prototyping, demonstration, execution, and monitoring of business processes for people working asynchronously in geographically dispersed enterprise settings [5,8,10]. Central to the technical components that address this business need are (a) the development and use of a business process scripting language, and a distributed process execution operating system that interprets and executes the process scripts which invoke legacy application system components and their associated repositories [3,5].

**Integration strategies**

Experience with the decomposition and re-integration of legacy systems can occur in a number of ways. Two approaches that have been investigated include the decomposition and recomposition of the functionality of legacy systems centered about business processes, rather than application functionality [4,5,10]. Said differently, understanding and redesigning business processes is an effective strategy for business transformation and migration to Web-based systems. Subsequently, addressing integration at the business process level is more effective than addressing integration of application systems as the application level [4,10]. Process-level integration based on a process meta-models also enables the integration of networked business processes that may employ local process modeling or execution notations that are globally heterogeneous. Thus process-level integration can support process-level interoperation of networked business systems components and enterprise processes [4,5,10].

**Using distributed component technology in developing Web-enabled systems**

In addition to the techniques identified above, we have also been investigating new methods for structuring and configuring complex business processes that must be coordinated and executed across multiple business sites. Here we have proposed software process architectures that provide the means for business processes to be composed from networks of component business processes [2]. In the domain of software acquisition, which is concerned with the funding, engineering, and deployment of large software systems across their life cycle, multiple autonomous enterprises participate in interlinked acquisition processes [8]. To date, we have successfully prototyped and demonstrated Web-based systems that support the modeling and simulation of software process architectures for software acquisition that operate over a network of process simulation systems (implemented in Java) and process simulators (person-in-the-loop systems) [2]. Together, these systems are used to support the evaluation and refinement distributed Web-based processes, as well as Web-based walkthrough and training on-demand for these processes [2,6].

**Discussion**

It should be clear that none of the concepts, techniques, or tools that have been identified or cited in this position paper are specific to NCC, wireless system applications or the use of information appliances. By focusing attention to the business process level, NCC application systems and appliances emerge as lower level implementation choices that support business processes. These choices arise at different levels of analysis and strategy. In addition, these choices do not drive the redesign or transformation of enterprise processes. Subsequently, a choice to focus first on the choice of NCC technologies, instead of on business process transformation, may result in technically interesting systems that are more likely to fail to realize significant benefits from a business standpoint when implemented or deployed. Conversely, to maximize the likelihood of successful transformation and migration to NCC systems, it is more effective in a whole system modernization effort to start with a focus on understanding an enterprise's business processes, as well as how to enable end users in the enterprise to redesign and experiment with alternative NCC business processes. As a result of such iterative participatory design, prototyping and experimentation, it will then be possible to identify what NCC application system components and appliances are needed to support which NCC specific processes.
Acknowledgements
This research was supported by ONR contract N00014-94-1-0889, the Experimental Acquisition Research Program (EARP) of the Defense Acquisition University by contract N00244-00-C-0097, and a grant from the National Science Foundation #IIS-0083075. None of the material in this report should be construed as an endorsement. Prof. John Noll at Santa Clara University led the effort to develop and evolve the PDI technology used in this research. In addition, Dr. Peiwei Mi, now at Vigilance Inc., and Prof. Mark Nissen, now at Naval Postgraduate School contributed to the concepts and techniques employed in this investigation.

References


Towards A Reference Architecture for Service Integration on the WEB

Eleni Stroulia
Computing Science Department
University of Alberta
Edmonton, AB, T6G 2E8, Canada
stroulia@cs.ualberta.ca

Abstract

Today, due to the proliferation of E-commerce related activities, the interface migration problem becomes increasingly important. In this paper, we discuss a method we have developed for the migration of existing interfaces to an XML-based representation, executable by multiple devices used to access the World Wide.

1. Motivation and Background

Recently, we have been witnessing the proliferation of services available on the World Wide Web (WEB). The scope of E-commerce activities increases continuously and practically every organization has, or is in the process of developing, a WEB site for offering its products or services.

In this effort, organizations face several challenges. One major challenge is the proliferation of target devices and platforms from which a web site should be accessible. From HTML browsers on desktops to WAP browsers on handheld devices, each target platform, with its distinct capabilities, imposes its own constraints on the types and amount of information that the web site should provide, as well as navigational constraints.

Another problem is the maintenance demands of such sites. The WEB, as a global marketplace, has enabled an unprecedented number of service offerings. To distinguish its own services from those of its competitors an organization has to regularly increase the number of services it offers through its web site, or provide new variations on them. At the same time, mergers and acquisitions, within and across business sectors, give rise to the need for site aggregation and service integration.

Finally, a great challenge is, in many cases, the actual resource on which the offered services are originally implemented. Many organizations own legacy systems on which their business processes depend. To transform services from being accessible to employees of the organization to being electronically available to customers and partners, organizations have to enable and control the access of these internal systems through their web sites.

In the past few years, we have been investigating several different aspects and instances of the general problem of “migrating services from internal corporate LANs to the WEB” [9], most notably in our CeLEST [1] and TaMeX [2] projects. In general, there are three important aspects to the migration problem that influence the methods applicable to it and the complexity of the resulting, satisfactory solutions:

1. Is the purpose of the migration the evolution of the currently available functionality or, simply, platform migration?
2. How many systems are involved in the migration problem?
3. Will the target system be internal or external to the corporation owning the current system?

In many cases, organizations migrate their applications because they upgrade their hardware infrastructure for purposes of performance or availability. For example, to make some of their services available to customers and partners, an organization may “wrap” parts of its application in a component, that can be accessed from the web through a thin client, i.e., a browser. This activity may involve no changes in the application functionality, or, in some cases, extensions, such as authorization and security, may have to be implemented.

In addition to hardware upgrades, another cause for migration is the integration of some or all of the corporate systems in the context of an enterprise application. To increase operation efficiency, many organizations seek to streamline their operations by
automating the cooperation of their different systems, such as human resources, accounting and customer care. In these cases, data integration and exchange become two of the most important objectives of the migration process.

Finally, an important aspect of the problem is ownership of the migrating applications. In many cases, the organization owning the application manages the migration activity. In some cases, however, another entity, seeking to provide value-added services, such as bulk ordering and price negotiation among product sellers and buyers, is in charge of the migration activity, and although the owners of the original applications are cooperating with it, the application itself is usually not available for modification.

In our work, we are specifically interested in aggregating legacy applications to provide complex services on the World Wide Web. In terms of the problem-space dimensions we discussed above, we have focused on migrating multiple applications, in order to make them accessible to the Web without adapting their functionalities, and without having access to their code. The major objective of our work has been to make the overall process lightweight, so that the aggregate application can be easily maintained to respond to changes of the underlying applications that may independently evolve. Another objective has been to support the accessibility of the aggregate services from a variety of target devices, including desktop browsers, WAP-enabled devices, and devices for enabling access to users with disabilities.

As a result, we have developed a reference architecture that we have applied to several different instances of the problem. Our architecture relies on the reverse engineering of the interaction between the users and the legacy applications in a declarative XML representation. This representation captures the API of the application, viewed as a component in the context of the aggregate application. This API can then be used by a mediator component to provide information to the application and invoke its functionalities. At the same time, it provides the basis for constructing a new multi-platform user interface for the application.

Our reference architecture can be used to support two types of application integration. For services accomplished by a well-defined sequence of functionalities of the different applications, the mediator can provide a centralized point for the user to interact with. The mediator can also manage the information and control flow, i.e., forward the user input to the appropriate applications, collect and forward intermediate results, and return results to the user. When the applications are quite independent and their integration involves simply the maintenance of a consistent global state across them, the mediator can be used to monitor the users’ interactions with the individual applications and to propagate the relevant information to other applications that may have to act upon it to maintain the consistency of the overall system state.

The rest of this paper is organized as follows. Section 2 describes our reference architecture, its components, and our method for instantiating it in the context of a new integration problem. Section 3 discusses the two different types of modes of application integration. Section 4 provides a high-level overview of the process by which we instantiate the reference architecture in the context of different integration problems. Section 5 discusses the different tools we have developed to construct such aggregations. Finally, section 6 discusses our experiments to date and derives some conclusions from this experience.

2. A Reference Architecture for Service Integration

Our reference architecture, shown in Figure 1, for providing aggregate services through legacy application integration consists of four layers: the user interface layer, the mediator layer, the wrapper layer, and the resource layer.

The resource layer, at the bottom tier, contains the original applications that are being integrated. These applications may be quite diverse: they may range from character-based legacy applications running on proprietary platforms, to thin-client applications with browser accessible, HTML-form interfaces.
Traditionally, very different types of migration techniques have been proposed and applied to these different types of applications. Character-based applications are usually decomposed in clusters of collaborating modules that are subsequently “packaged” in components accessible by external applications through frameworks such as CORBA [12]. On the other hand, for thin-client applications with browser-accessible interfaces, post-publication processing of their interface data with XML has recently been the technique of choice. In our reference architecture, we propose the same integration method for all types of underlying applications, based on the explicit modeling of the interaction between the user and the native user interface of the application. The interaction model, represented in XML, describes the information exchanged between the application and the user for each different user task. This model functions as a public API for the resource, so that the same task-specific interaction and information exchange can be carried out through a new user interface, or enacted by a wrapper given sufficient data.

In the next layer, on top of the original resources, exist the resources’ wrappers. The role of a wrapper is that of an adapter between the original API of the resource, as represented by the application’s interaction model, and a new API, based on a canonical domain model common across all wrappers.

On top of the wrappers, exists the mediator of the aggregate application, whose role is to coordinate the activities of the wrappers, by receiving (forwarding) information from (to) them. It does so based on a model, in yet another XML language, of the types of information exchanged among the wrappers, i.e., produced and consumed by them. Depending on the style of the integration (a detailed analysis of two alternative styles follows in Section 4), the mediator may drive the wrappers or it may respond to events it receives from them.

Finally, at the topmost layer of the architecture belong the user interfaces through which the individual applications and the mediator may be accessed. These new user interfaces are accessible through a variety of devices, such as web browsers, cell phones, and PDAs. To that end, their structure and behavior are specified in XML and this specification is interpreted at run time by generic, device-specific translator components.

3. Deliberative vs. Reactive Integration

In general, there are two styles of application integration, and they imply first, different roles for the mediator of the architecture described above, and second, different styles for the interaction between the user and the integrated application.

In some cases, when the information production and consumption relationships among the underlying applications imply a sequential flow of control, the service provided by the integrated application can be described in terms of a well-defined task structure. Consider for example a travel-assistant application, supporting users to plan their travels by finding air flights and hotel reservations for them, given their preferences. In this scenario, the flow of information from the user to the underlying applications, i.e., calendar, airline-reservation systems and hotels, follows a well-defined, if not completely deterministic, sequence. The user has to provide some (range of) travel dates, the origin and destination of travel, and possibly preferences and/or restrictions regarding airlines and hotels. This information can then be forwarded -possibly in parallel - to the available airline-reservation systems. When some travel plans have been found by the airline-reservation applications, the travel dates can become firm, and the hotel-reservation applications can be invoked to provide possible hotel room information.

As an alternative, consider the problem of integrating a company’s accounting and customer-care applications, so that interesting states of the accounting data result in activation of customer contact, i.e., a customer’s total invoices exceeding a given level causes a “thank you” letter to be sent to this customer. This style of integration is much less structured than the travel-assistant example. There is no “high-level task” that requires the integration of the individual application functionalities. Instead, the integration involves simply monitoring the usual interactions of the user with the original applications, recognizing events of interest, and invoking other applications with the monitored information as input.

We will call the former style of integration deliberative and the second reactive. In the deliberative integration scenario, the user interacts with the mediator of the aggregate application, which, in turn, coordinates the flow of control and information to the integrated applications and collects the results to present them to the user. In this scenario, the user does not interact with the individual applications. In the reactive integration scenario, the user is not explicitly aware of the aggregation, but interacts with the original applications through their migrated user interfaces attached to their wrappers. The wrappers’ interfaces receive the user-provided information and enact the system-user interaction expected by the original application interface.
Furthermore, they forward records of the interaction to the aggregate application’s mediator. The mediator, in turn, may forward this information to other wrappers, according to the integration “business rules”, in order to invoke related functionalities on the wrappers of affected applications.

Finally, it is important to mention that these two styles are, by no means, mutually exclusive. A hybrid style would also be possible.

4. Developing aggregate applications for service-integration

To integrate a set of existing applications in an aggregate application following the reference architecture, described in Section 2 above, the following activities are involved:

1. reverse engineering the information exchange between the user and the native user interface, for each original application use case;
2. developing the canonical domain model, shared by all the applications wrappers and the mediator;
3. developing the model for the mediator functionality of coordinating information exchange among the wrappers;
4. developing new user interfaces for the original resources and/or the mediator.

The whole integration process relies on understanding and modeling the current system-user interaction, as implemented in the original system interface. This reverse engineering activity consists of three phases. In the first phase, multiple traces of the original system-user interaction are recorded by emulating proxies through which, expert users access the original system interface. Based on the recorded traces the original system interface is understood in terms of the unique different states it goes through and the transitions between them [4]. Note that this phase is redundant in the case of HTML interfaces, because the URLs of the different interface forms uniquely define the interface states.

After a model of the original interface behavior has been constructed, in terms of the interface screens and the transitions between them, a richer model can be constructed for specific tasks of interest. An expert user demonstrates the execution of a task through multiple different task instances. These “parallel” traces, that is, traces navigating through the same state sequence of the original interface, are compared in order to analyze the exchange of information between the user and the underlying system [3,5]. The result of this analysis is an “information-exchange plan” for each application use case, specifying the

navigation of the user through the original application user interface, the user inputs to the application at different states of the process, the application outputs, and their inter-dependencies.

Having identified the information parameters of the interaction process, a user-driven process may be invoked to categorize the data exchanged between the system and the user in terms of a domain model. The domain model may be constructed bottom up, i.e., a user may annotate the information exchanged between the applications and the user with labels indicating domain-object names, or top down, i.e., the user may import an existing domain model and may categorize the information exchanged in terms of its object types.

The next phase of the process is the specification of the new interface in XML. The new interface specification represents the complete sequence of elementary interactions that have to occur on the original interface in order for the task to be performed. Furthermore, it explicitly distinguishes between the necessary user inputs and the ones that can be derived either from earlier user inputs or from system outputs. Finally, it specifies, for each necessary interaction, an appropriate abstract interface widget. This interface specification has two very important properties. First, it is an API for the original application: given a set of values for the necessary user inputs and an emulator of the original interface, it can completely execute the task on the original interface. Furthermore, it is executable by different platform-specific “translators”. To date, we have developed XSL [2,5], XHTML and WML [1] translators for enabling the execution of the new interface by browsers and WAP-enabled cell-phones and PDAs. We have also developed, in a similar manner, Java-based interfaces [3,4].

The final phase of the migration process involves the development of the integration mediator [5,6]. The mediator’s process is described in terms of a hierarchical task structure [11] in the case of deliberative integration, or in terms of a set of event-condition-action rules [14] in the case of reactive integration. Mediators of deliberative integrations also have user interfaces, designed in principle as the front-end interfaces of the underlying integrated applications.

5. Tools

To date, we have worked on two projects, CeLEST and TaMeX, investigating the different aspects of the application migration and integration problem. CeLEST consists of the LeNDI, URGent, Mathaino and Babel tools and has focused mainly on the problem of developing reactive integrations of legacy
systems with character-based interfaces. TaMeX, on the other hand, has explored the problem of constructing deliberative integrations of thin-client applications with HTML-form interfaces.

5.1 Interaction Reverse Engineering

LeNDI [4] is the cornerstone tool of CelLEST whose main objective is to unambiguously identify the distinct screens that the legacy application exposes to its users. It does that by classifying recorded screen snapshots in clusters of same-screen instances based on the snapshots’ visual features. We have engineered a set of distinctive screen features to capture information about the elements on the screen and their layout. We have also experimented with different clustering algorithms. Based on the characteristics of each of these clusters, LeNDI then produces a predicate for classifying new snapshots in the clusters at run time. These predicates are for legacy screens what URLs are for HTML forms. Thus, this step is redundant in the context of TaMeX that focuses on migrating HTML interfaces.

5.2 Interface Migration

The next step in the process is the specification of the application wrappers and their user interfaces, which are required for reactive integrations. URGent [3,4] was the first system, developed in the context of CelLEST, for that purpose. URGent constructs an intermediate representation of the interaction between the user and the application and then uses this representation to construct a Java applet containing an appropriate set of input and display widgets to receive (deliver) information from (to) the user. The interfaces designed and developed by URGent are limited in the class of devices that they can be accessed from; only devices that can run the JVM can execute them.

Mathaino [7] is the successor of URGent, and its objective is to further automate the process of legacy system migration and enable the simultaneous migration to multiple platforms. In the context of Mathaino, we have developed an XML language for specifying the interaction plan. Furthermore, we have developed a toolkit of platform-specific interpreters for this language, so that the interaction plan can be interpreted at run-time on different platforms such as XHTML-enabled browsers and WAP-enabled [14] devices, which pose special constraints in terms of processing power and display capabilities.

Finally, TaMeX [5,8] uses similar interaction analysis methods to extract the interaction plans for the different services provided by the underlying applications. These interaction plans are represented in XML, in a language similar to the one used by Mathaino. We are currently working on bridging the gap between the two languages.

5.3 Mediation

Reactive integration in CelLEST is accomplished by Babel [6], a blackboard that monitors the interaction of the users with the underlying applications through the intermediate wrappers. At design time, the set of rules governing the integration is specified through a visual drag-and-drop interface in terms of an XML language modeling event-condition-action rules. At run time, as Babel receives “interaction events” from the application wrappers, it evaluates the conditions of these rules and when they are met, it generates new “interaction events” that it forwards to the appropriate wrappers. These new events cause the recipient wrappers to enact other actions in the applications they encapsulate.

TaMeX implements deliberative integration. It provides an XML language for specifying the mediator’s task. At run time, this task structure controls the interaction of the user with the aggregation and the collaboration of the underlying resources for delivering complex services as a hierarchical task structure. In addition, TaMeX assumes a representation of the ontology of the common application domain, also in XML. To enable the user interaction with the mediator, TaMeX uses a set of XSL stylesheets that translate the task structure in a menu-driven interface that invokes XSL forms for requesting user input and displaying information to the user. The TaMeX stylesheets are reusable: the same stylesheet can be used to render on user browsers any task structure specified in the TaMeX task-structure representation and the stylesheets developed for the application-domain entities can be reused by any mediator in the same domain.

6. Related Research, Experience, and Conclusions

We have described a method for legacy migration and service integration and a reference architecture characterizing the aggregate applications produced by this method.

This work lies within several different areas of research. The problem of application aggregation and mediation is not new. Even before the World Wide Web became so pervasive, similar issues have been examined in the context of database federation [11]. In the same context, there has been a lot of work on reactive integration, or continuous monitoring [10]. Although several technologies exist for deliberative-style integrations, such as CORBA, DCOM and RMI,
there has been, to our knowledge, not much work on declarative representations of deliberative integration schemes, such as the XML-described task structures of TaMeX. However, there is a lot of work on post-publication XML representation of HTML content [13,14]. Finally, another related area of research, which has influenced our work on URGenT and Mathaino, is that of model-based interface design [15,16].

Our experience to date indicates that the overall method is effective. We have used TaMeX to build a travel-assistant application by integrating, in the deliberative style, several airline-reservation systems and URLs with related information [8]. Using the CellEST tools, we have developed an aggregation, in the reactive style, that enables users to email library search requests and receive the library responses as email messages that they can read through their cell phones [6].

The main advantage of this integration architecture is its applicability to a variety of original systems, ranging from legacy systems with character-based interfaces to client-server applications with browser accessible thin clients. Furthermore, it is quite extensible in terms of the target platforms that can access the aggregate services. Finally, the interfaces produced, although not very “creative”, are quite maintainable, because they are declaratively represented in XML.

References
1. CELLEST-related information available at www.cs.ualberta.ca/~stroulia/CELLEST
2. TAMEX-related information available at www.cs.ualberta.ca/~stroulia/TAMEX
Abstract
This paper comments on the state of net-centric computing (NCC) as of the first quarter 2001. From the first NCC workshop in 1997 until today, much has changed. For example, backbone bandwidth and computing power have increased tremendously. However, some things remain the same. For example, consumer-oriented network connections remain relatively slow. Two new developments that stand out are infrastructure support, such as Microsoft’s .NET initiative, and applications, such as peer-to-peer (P2P) networking.

Keywords: net-centric computing, .NET, P2P

1. What is NCC?

The underlying principle of Net-Centric Computing (NCC) is a distributed environment where applications and data are exchanged among peers across a network on an as-needed basis. NCC relies on portable applications running on multiple platforms, mobile code and data accessed via high-speed network connections, and low-cost appliances for local processing [18].

This is in stark contrast to the use of powerful personal computers that rely primarily on local resources. In some respects, NCC resembles an earlier computing era of mainframes and dumb terminals. However, there are significant differences. NCC relies on portable applications running on multiple platforms, mobile code and data accessed via high-speed network connections, and low-cost appliances for local processing.

The last few years have seen interest in net-centric computing wax, wane, and then wax again [17]. At the moment, there appears to be a resurgence of interest in the area, driven in part by the proliferation of non-traditional computing devices, by dramatic changes in networking capabilities, and by renewed interest in centralized system administration. Witness the popularity of personal digital assistants such as Palm Computing’s Palm VII, which provides wireless Internet access (in selected cities) for mobile professionals [13].

While the Palm may be considered representative of a hardware NCC device, a software NCC application is a program that relies on an NCC environment to operate. A net-centric application is a collection of technologies that, taken together, characterize this new paradigm. A more holistic way of looking at an NCC application is to look at it as an amalgam of its two main components: the “net-centric” portion and the “computing” portion. Previous NCC workshops have discussed both aspects of the equation.

2. Previous NCC Workshops

There have been two previous NCC workshops, the first in 1997 and the second in 1998.

2.1 NCC ‘97

The first workshop on NCC was held in July 1997 in conjunction with the 8th International Workshop on Software Technology and Engineering Practice (STEP’97) in London, UK [9]. Through focused presentations and open discussion, the NCC workshop explored net-centric computing and its potential impact on software users, application developers, and system administrators—in essence, to sort out the hype from the reality.

The workshop had approximately 22 participants from 8 countries. It was originally planned to have a commercial vendor participate in the workshop so that attendees could get a hands-on evaluation of a network computer, but that proved infeasible. Instead, the workshop focused on three themes, with invited presentations and interactive discussions structured around them.

The first theme was tutorial in nature: exchanging information about NCC and exploring issues such as total cost of ownership of network computers, the NetPC, and Java appliances. The second theme was security in a networked environment. The third theme concerned leveraging investments in existing applications when
moving to a net-centric platform. A report summarizing this first NCC workshop’s activities is available to the interested reader for more detailed information [19].

2.2 NCC ‘98

The second NCC workshop was held in April of 1998 as part of a series of meetings concerning a research project sponsored by the Institute for Robotics and Intelligent Systems (IRIS) meetings at the University of Toronto, Canada. This second workshop was more focused on legacy system reengineering and, in particular, on the new challenges inherent in migrating to a networked environment for currently deployed applications. Key points of discussion included: business cases for NCC, relevant distributed object technologies, and strategies for extracting business rules, processes, and objects from legacy systems.

Technology and business cases go hand-in-hand.; you cannot have one without the other. To adopt technology, you need a sound business case. To formulate a business case, you need a sound understanding of the technology. Perhaps technology should simply be considered the cost of doing business and staying competitive. Usually, however, more convincing arguments are needed. In early 1998, intranets were becoming popular, causing major, perhaps fearful, changes that blurred the traditional divisions inside companies among sales, marketing, customer service, engineering, and IT. One significant issue was measuring return on investment on such an endeavor. Simply building an intranet using NCC technologies doesn’t guarantee customers. Did NCC truly enhance the customer experience in online shopping? Did NCC help to integrate, streamline, and manage the diverse activities of a large multi-national corporation? Did NCC serve to strengthen the corporate brand? These were key issues of interest to businesses.

Relevant technical solutions discussed for NCC included distributed object technologies (DOT) such as CORBA, DCOM, and Java. At the time, interest in document-centric computing using an infrastructure like OpenDoc (proposed by OMG for CORBA and competition with OLE) was dying. Such an infrastructure might have been fine for small software component makers, but the inertia was with the large application makers like Microsoft and Adobe. Also, it wasn’t clear that most end users would want to shop for and assemble their own applications from components. The workshop focus, however, was more on using DOT to wrap legacy systems (for web access or legacy systems integration) instead of using more conventional white-box reverse engineering.

Since time to market is so critical, an understanding of wrapping technologies and the need to extract interfaces from heterogeneous legacy code was a topic of lively discussion in the workshop. That is, there was a need to identify and ease the construction of business objects from legacy systems, and also relate business rules and objects (assembled like components by engineers) to reengineering business processes. One overarching motivation was the concept of cooperative information systems for supporting business-to-business operations.

As NCC’98 was a project-related meeting, workshop participation was limited to active members, which included Carnegie Mellon University’s Software Engineering Institute, IBM Canada, Techné Knowledge Systems Inc., the University of Toronto, the University of Victoria, and the University of Waterloo. No public report of this workshop is available.

3. What Hasn’t Changed?

At the time of writing (the end of the first quarter of 2001), about three years having passed since the last NCC workshop. It seems reasonable to ask what has changed and what has remained the same in the NCC arena. What does the current NCC landscape look like? It is illustrative to look back and see what didn’t happen since the last NCC workshop. From the net-centric portion, network bandwidth for most consumers remains relatively slow. Most people still connect using dialup modems, severely limiting the widespread adoption of net-centric applications.

From the computing portion, the PC didn’t go away. In fact, current market slump aside, the PC continues to dominate most aspects of computing, net-centric and otherwise. There were a few early rivals to the PC in the thin-client category, such as the NetPC, but they never really caught on. The more general product category of “Internet appliance” still remains dormant. According to Michael Dell, CEO of Dell Computers, “The concept of the Internet appliance has been an economic disaster. Nobody has purchased them.” [14]. The tribulations of strong NCC proponents such as Oracle Corp. to bring a true thin-client to the mass market attests to this fact. Even newer offerings, such as 3Com’s “Kerbango” Internet radio and their “Audrey” home Internet appliance have been withdrawn [8].

The dominant thin-client is a Web browser, running on a traditional (although vastly more powerful) personal computer. Calling Internet Explorer 5.5 running under Windows 2000 on a 1.7GHz Pentium 4 a “thin client” is perhaps something of a misnomer, nevertheless, that is how many users view their commodity computer. Email,
Web browsing, and word processing are still the primary client-side applications for most computers. In a server role, the PC is becoming a central hub for digital information to and from portable devices such as digital cameras, video cameras, MP3 players, and handheld computers. It seems the PC’s reign is far from over.

4. What Has Changed?

Since the last NCC workshop, new developments in NCC can be categorized according to two main components: the net-centric portion and the computing portion. As with most things in computing, advances in hardware tend to outpace advances in software. For NCC to become more prominent, new developments in both areas will be needed to create the next revolutionary application.

4.1 The Net-Centric Portion

The net-centric portion of an NCC application has experienced tremendous change since the last NCC workshop. Two areas in particular that are of central importance in the first quarter of 2001 are broadband connections, and wireless, mobile, and always-on connections.

4.1.1 Broadband Connections

Broadband is something always seems to be “just a few months away” for consumers. While the main network backbones have increased capacity tremendously since 1998, most users are still stuck with slow modem dialup connections. The so-called “last mile” problem is still an issue, and may so for the foreseeable future.

The trade press is replete with stories of consumers struggling to gain access to high-speed Internet connections [16]. Cable models and DSL are the two main contenders to replace the dialup modem, but coverage is spotty and installation difficulties are common. Until broadband connections become commodities, NCC subscription-based applications such as software served by application service providers (ASPs) will remain niche offerings.

4.1.2 Wireless, Mobile, and Always-On Connections

One proposed solution to the last-mile problem has been near-broadband wireless connections. In fact, wireless networking is one area that has exploded since 1998. Even consumers can install simple to use wireless home networks to connect their multiple PCs to share devices, files, or access to a high speed Internet gateway.

The next step in wireless networks for mobile appliances are always-on connections that maintain constant connectivity between local client and the network that surrounds the client. Even when the client is mobile, the ability to remain connected adds greatly to the functionality of the appliance. The coming generation of cellular phone services offers always-on mobile connectivity; if early feedback from Japan’s very successful DoCoMo service are indicative, this always-on aspect will truly change the nature of network computing.

4.2 The Computing Portion

Broadly speaking, current developments in the computing portion of NCC can be described as infrastructure and applications. For infrastructure or enabling technologies, Microsoft’s .NET initiative is a good example of a huge commercial investment in NCC [11]; whether or not it will pay off is unknown. As of the first quarter of 2001, .NET remains mainly a promise. For the latter, applications, peer-to-peer (P2P) programs were all the rage in 2000. Witness the popularity of Napster [12], Gnutella [6], and other file-sharing systems. P2P is, however, much more than just swapping MP3 files; it is a true example of the power of NCC to change the computing experience.

4.2.1 Microsoft .NET

There are many examples of net-centric applications today. One example already mentioned above is Napster, which uses a client talking to a centralized directory service and uses all the participating computers on the network as servers. Another very popular example is instant messaging, where a rich client uses a local directory (called a “buddy list”) to communicate with other rich clients in the network. Microsoft says .NET is aimed at accelerating this next generation of net-centric computing.

.NET can be viewed as “software-as-a-service” that uses messaging as its communication structure and XML for data storage, connecting all your data, devices, and applications together [3]. This allows any application or device to share piece of information and act on them, with or without the users’ awareness or active participation. .NET is an emerging infrastructure heavily dependent on four constituent standards-based technologies: HTTP, XML, SOAP, and UDDI. HTTP (Hypertext Transfer Protocol) is used to transport data and provide access to applications over the network. XML (extensible Markup Language) is the common format for exchanging data stored in diverse formats and databases. SOAP (Simple Object Access Protocol) enables applications or services to make requests of other applications and services across the network. UDDI (Universal Description, Discovery, and Integration) is a DNS-like distributed Web directory
that enables services to discover each other and define how they can interact and share information.

There are three facets of .NET that must co-exist for it to work properly: .NET servers, the .NET Web-based service framework, and .NET application building block services [4]. At the bottom of the hierarchy are .NET Enterprise Servers. These consist of hardware and a family of products that embody Microsoft’s own infrastructure for data center hosting of .NET applications and data.

The next layer up is the .NET Web-based service framework. It is a development infrastructure for programming .NET applications and services. It uses the new.NET-oriented Visual Studio developer toolset. The .NET framework service consists of three components: the Common Language Runtime (CLR) runtime environment, several class library hierarchies, and an advanced version of Active Server Pages (ASP) called ASP.NET. CLR allows for a single unified environment for Microsoft programming languages (such as VB.NET, managed VC++, and COBOL.NET). Microsoft has also developed a new programming language inspired by ideas from C++, Java, and Delphi, called C#.

The final layer is Microsoft’s universal building block services, a set of pre-built services that developers integrate with their own applications and services. The services, such as identity, notification and schematized storage, can benefit both users and developers.

Besides the CLR, .NET security relies on Microsoft Passport, a password and user authorization tool. It allows your identity to be passed between different services — Web sites, instant messaging clients, and so forth — without you having to enter it each time. Logging into one Passport service, like Messenger, Hotmail, or MSN, automatically logs you into the others. To let .NET provide efficient notifications, the system needs to know everything about you, such as user profile, what device(s) you use, your personal calendar and contact list, and your current location. Putting all this information into one single server might create an irresistible target for hackers. Privacy and security in an NCC environment is an area that should concern both developers and users.

### 4.2.2 Peer-to-Peer Networking

Napster is a popular program that has been widely used by people to share MP3-encoded music files. Napster implements the peer-to-peer (P2P) network concept. In a P2P network, the nodes do not behave as clients or servers exclusively. Instead, each node can act as both a client and server simultaneously. Anyone using Napster can make music files available off of their computer to anyone on the Internet who is also running Napster (or one of its functional equivalents). Napster provides a search capability to explore all of the files that fellow users are making available off their local hard drives. Once selected, the MP3 files can be downloaded from the server to the client by just clicking on them.

Gnutella, the first P2P program under that moniker, was developed in March 2000 by Justin Frankel and Tom Pepper of Nullsoft (owned by AOL/Time Warner). They never released a public version of Gnutella because the parent corporation declared it to be an “unauthorized freelance project.” But the original version, 0.56, had already been released as an early beta to several hundred people, and its underground popularity continued to grow.

There are many working clones of the original Gnutella release, with their functionality derived from a reverse-engineering effort of the original program. Clones are available for all major computing platforms (Windows, Linux, Macintosh), as well as Web-based interfaces to Gnutella programs. The current “sanctioned” leaders of the Gnutella-compatible software effort include BearShare [1], LimeWire [10], and ToadNode [20]. These clones vary depending on the type of front-end they offer, the type of programming language they are written in, and the operating system they are written for. Some do not implement all of the functionality of the Gnutella protocol. Many of these early clones tend to be crash-prone, since they are based on a buggy beta.

Gnutella is similar to Napster in that it implements a P2P file sharing network. However, Gnutella is more generalized than Napster, allowing users to share any type of file, not just music files. To accomplish this, Gnutella uses a different protocol than Napster. The Gnutella protocol realizes distributed file sharing by providing the mechanisms for the creation of a P2P network. Every node in a P2P network may act as a client (when it downloads files) or as a server (when it makes available files for others to download.) These nodes are popularly called servants (server+client). Each servant in a Gnutella Network implements the Gnutella protocol. One issue may be the scalability of a network that requires servants to discover the location of content instead of consulting a central index, especially with the preponderance of slow network connections.

The reason P2P represents such an exciting new “pure” NCC application is that it’s power goes beyond simple file swapping. P2P can be used to foster collaboration among geographically distributed coworkers, as with the Intel-funded Groove Networks, Inc. [7]. P2P can also be used to share computing resources, such as the SETI Institute’s efforts to leverage dormant PC cycles in a distributed virtual supercomputer [5]. What the next “killer app” will be is unknown, but it seems there is a good chance it will make use of P2P technology.
5. What’s Next?

Perhaps the most pressing problem for NCC, something that affects both the net-centric portion and the computing portion of NCC, is the critical issue of security. This was a topic of discussion at the first NCC workshop, and its importance since then has only increased. In fact, several pundits have written that the success of NCC applications such as P2P will only flourish if the security issue can be solved (or at least mitigated) [2].

In the book *Secrets & Lies*, well-known security expert Bruce Schneier advocates a more holistic approach to computer security [14]. As he often states in the book, security is a process, not a product. The reliance on technology as the sole source of security for today’s information systems is misplaced. After all, people use computers, and people are inherently unsecure. Moreover, the problem is exacerbated by the increasing complexity of today’s computer systems. The sheer size of these programs means that no one really knows how all the parts go together. This is particularly true for NCC applications, where code and data are mobile and visibility into the entire program is difficult to obtain.

By the time the next NCC workshop takes place, no doubt there will be many exciting new developments that have taken place. Hopefully by then some of the current shortcomings will have been addressed as well, such as higher-speed consumer network connections. The area of net-centric computing is a rich source of new challenges, full of opportunities for both academic researchers and commercial tool developers to explore.

References

ABSTRACT
This paper describes the application of the net-centric computing paradigm towards the definition of services supporting cross organizational cooperation among people and workgroups within their work. The services will be designed as complex and user oriented solution supporting the work of the member of cooperations. After describing the concept an architecture will be presented, which is part of a solution for the public administration. Basing on this example the function of the services will be presented.

Keywords
services, central architecture, cooperation, SOAP, XML, service description

1 Cooperation in public administrations
The work of the public administration must become more efficiently. There are guiding principles, for example the »thin administration« or the »Lean Government« guidelines, which should be followed by each authority. On this way modern IT-systems and organizational concepts will be introduced in the different authorities.

The authorities on their own often do not have the capacitance and the know how to setup up and administrate such systems. Therefore the authorities often consider to buy such achievement from other organisation or companies.

For the fullfillment of the work of the public administration cooperative work becomes currently and in the future more important. The tasks within cooperations will get more and more complex and cannot be performed by a single person or a single specialised team.

To offer solutions for cooperations different requirements have to be heeded. On the other hand the independent members of a cooperation each will get the same rights on the data and the results. Therefore a independent provider can deliver a solution for all members and can support their work especially their temporary work.

However looking closer onto cooperations different types of it can be distinguished.

Different kinds of cooperation
In some of our projects we looked into workgroup processes in the public administration. Especially the working in organization-encroaching groups was the focus. From these experiences we recognized different types of cooperation.

There is a kind of cooperation with definite structure and process, like the committees of the german parliament. This cooperation has definite times, when the participants meet and they have definite processes, how they work. These processes are partly defined by law. The subscribers of the cooperation are certain from start on.

Another kind of cooperation consists of different participants and different kinds of structure and processes. This can be evolved up to cooperations without defined structure and processes. The guidelines for these cooperations are very different and the subscribers can change. So they have different specifications within their lifecycles. The lifecycles here will be considered as simple ones, consisting of convocation, working and liquidation.

The cooperations in the public administration follow a common standard that is specified over the method of work. So for example there are guidelines for making decisions. This common base makes the cooperations in the public administration very interesting for the application of the service concept to support their tasks.

Example for a cooperation - drafting a bill
For demonstration, a particular example is described in this paper. This example describes the cooperation for »drafting a bill«.

In a committee for »drafting a bill« there are different members out of different organisations. These organisations can be from the public administration like ministries or other public organisations. On the other hand the members can come from the economy or social organisations like unions. So there are great differences between the members concerning different work cultures, technical equipment, etc. They are also independent between each other. They are not in a dependence, so that the rest of the members can be governed by one. In this example some ministries work as "full-member" within the cooperation and the union and the organisation of employer work as "guest-member".

The kind of membership may be different. Some members have only the right to make comments and not the right to consider the decisions. Some subscribers participate as specialists for a part-question of the topic to be processed. Therefore they participate in the cooperation not for the
whole lifecycle. For "drafting a bill" this can be the members of the ministry of justice. They have to check the bill if it can be a law or if there are some faults. So they don't collaborate with the technical workflow. Also their rights correspond to their status in the cooperation. So the "guest-members" don't get all information and documents.

The tasks of the cooperation change within the lifecycle of the committee. First a cooperation has to be created, therefore relevant members have to be searched and consulted. In the working-phase there are tasks for communication and cooperation necessary, like creating a bill, giving information to other committees, and so on. So the necessity of functionality is very different.

2 Solutions for cooperations
Cooperations are often presented as an answer to the challenges from the market [3]. The most important challenges have been realized as the fast reaction of changes, the production of individualised products and services [6][16]. Therefore many aspects of cooperations and virtual enterprises (VEs) will be taken into consideration. Since there is no common definition of VE available [15] we are not going to define the term in this paper but rather proceed with a general understanding of VE. As well the kind of providing information gets more in the interest of research [1].

[8] analyse the aspects of self organisational management structures between teams of different departments. The focus there is the organisation of different teams within a company to work on complex problems with inter-disciplinary knowhow [12]. The work with processes in virtual organisations or cooperations is also subject to research [13].

Other research deals with some aspects of working together in cooperations. Therefore [9] analysed within an empirical study the use of synchronous media for the coordination of working groups. For the synchronous work in a workspace [5] looks for special mechanisms [also 4]. Another kind of presence, especially in virtual communities is the objective from [10]. The virtual communities are supported with a system basing on the agent-technologie [11]. [7] describe the "Medium" as new platform for cooperation and the community itself. The services may be a special kind of support between such a medium of the business media.

The focus of our research is the support for cooperations based on services, especially complex services. "In generally services realise IT-functionality, which will be offered, arranged and made available by a corresponding network infrastructure. ... Complex services will be composed by less complex services, which can be ordered on the service marketplace" [14]. Services may have many kinds of levels and a trader combines the services for the user. Therefore Messer/Sandkuhl design a dynamic CSCW-System. A similar dynamic system for the support of collaboration environments, especially the workplaces describes [2].

3 Complex services for the public administration
The use of services seems to be a solution for the support of cooperations. The services becomes the bearings for supporting the tasks. For defining a service, the tasks of the specified areas of business, here the public administration, have to be studied for their typical structure and processes. So a service is a solution for a complete task consisting of several steps and therefore is a complex service.

Services as central cooperation functionality
Service on it’s own means the offer of functionality in an electronic way, based on a common network. Services can be offered by specialised companies, this results in some kind of outsourcing (see Pic. 1). On the other hand services can be delivered by an IT-departure within a company. In both cases the services will be supported by a central institution. Therefore the customer doesn’t need to investigate special competences on it’s own.

Another benefit of central service management is the change management. If there is no need for a functionality any longer the customer doesn’t subscribe the service any longer. The other way round, if the customer needs an additional functionality, he is possible to subscribe this one from the provider. Services can be subscribed or unsubscribed if they are not used any more. So only the cost during the use of services is charged to the cooperation. This adopts the model of application service provision to the concept of central service management for workgroup/cooperations.

Pic. 1 central service offering

Services as complex solution
As described above it is the goal of services to satisfy the complex requirements of the users. A service is called a complex one, if it has a process part. Services that have no process part are called basic services. The user of a service uses the single functionality in a context of his work. So he uses "best of breed" solutions for single tasks. In another way the services have to be adaptable to the needs of the users. By that the users get an individualised solution.

The flexibility of the configuration is also an argument for using services.

Flexibility of service-configuration
The services deliver functionality, which is needed by the cooperation, i.e. the committee. In the lifecycle of the committees work there is different need in functionality. Some functionalities are only needed to instanciate a committee, so these functionalities are only subscribed for the first phase. Other functionalities are only used for the liquidation. Therefore it’s needed to react flexible in putting
together the right set of services.

4 Service-offer
Having described some requirements for the services so far we are now going to describe a possible realisation for the services. Therefore an architecture will be presented, which is used to realise services for the public administration. The example is a part of an architecture which was developed in different projects.

Service-architecture
Based on a module-structure new or more complex services have to be composed by other services. So a service for a user can be configured individually. The provider can react very flexible to the requirements of different users by building up different, individualized „service trees“ as a result of the modularity. The configuration of the service can be changed during the lifecycle of the cooperation.

Pic. 2 structure of services

This module-structure should be portrayed in an architecture (see Pic. 2). The architecture displays the dependencies between the services. It allows the developer to get an overview about the services which exist. So it’s easier to compose new complex services, with functionalities of other services.

The service »Protocol writing« needs some other services to offer the complex functionality, which is required by the customer. The dependencies between services can be different. So there are options which depends on the general functionality of the service, like here the »Register of members management«. These option can be choosen by the subscriber for all common users of the cooperation. A single requirement of one member of the cooperation can be the support for specific software, in this case the word processing tool. Therefore single requirements can also be considered by the service provider.

Service integration
The fulfillment of the service will be mainly done in the environment of the service provider. Nevertheless the user wants to use some local functionality or data. So an integration of the user interface of the service within the IT-environment of the customer has to be possible. The representation of the service can also be designed for supporting mobile customers. In this case the user work with the services in different ways, for example in the office with a more integrated user interface and in a mobile situation with a more restricted towards the capabilities of the mobile device.

This integration of the services can be realised in different ways. The simplest way is the presentation of the services within a browser. In this case the developer does not have to consider the environment of the users. A second way is the integration within an existing software product. This can for example be a groupware-tool like Microsoft Outlook or Lotus Notes. In this case the integration in the environment can be designed much closer than the browser offering. On the other hand there are more dependencies between the services and the environment that have to be considered for the developing of the services, for example the different versions, the different compounds of the users environment. A third case can be realised by a separate software which assumes the communication between the user-interface and the services. The loss of the last realisation is the additional tool for the user.

Service calls
Within an architecture a service can be a single one with it’s own functionality or it can be part of a more complex service. In the example (see Pic. 2) the »Document management« service delivers a part of the functionality of the »Protocol writing« service. Therefore the »Protocol writing« service has to be able to integrate the »Document management« service in it’s process. So the more complex service send calls to the minor complex service to use the specific functionality.

On the other hand, the »Document management« service can be provided as a single service on it’s own. In this case the user-interface (i.e. the service client) send calls to the service to use its functionality. To get a consistency between the different kinds of use of the services it’s necessary to design one interface for both use-cases.

For the architecture of the services the communication between the services has to be standardized. Also the user interfaces has to send calls to their services. This way contains some different obstacles, like firewalls. So we are looking for a simple realisation which handles these obstacles. Therefore we choose the SOAP-Protocol (Simple Object Access Protocol) for defining the specific calls.

The SOAP-protocol additionaly abstract from the realisation of the software which builds the services. Therefore the use of the services from different provider, within different networks and environments becomes possible.

The SOAP defined calls for the services represents the basis for the developers of the service. On the other hand the user although wants to know which functionality the service offers. However end-users usually are not capable to understand SAOP calls, resp. thinking for example about public administration as one application area: users coming from an application area should never be enforced to learn IT
languages. This would extremely reduce user acceptance. Therefore we use another benefit of SOAP, which is the XML-structure for the calls. So we are developing a language for supporting different views each one described in a language oriented towards the users need and integrating different views on the services within in one description. Therefore we use the XML-structure to define different parts of description, the view for a user, the view for the IT-specialist who integrates the services in the environment of the user and the developer of the services.

For the developing of user oriented services different tasks has been done. One of them is the definition and documentation of services. On base of this documentation the services can be chosen by the customer and developed by the provider. Therefore the structure of the interface has to be defined, so that there is a common description for the interfaces.

5 Prospects of service-development

The use of services will be increase in the next years. The start makes the different Application Services Provider (ASP) which provide single software products over a network. The ASPs althought pursue a different purchasing model. The next steps will be the integration of different functionality up to the user oriented services.

For this trend it’s necessary to give a conceptual support. So we are developing a languages for the description of services. With an additional step the processes in different branches have to examine to get the relevant services. For the public administration different services will be developed and realised within a prototype.

REFERENCES

16. Wolfgang Faisst; Mertens, Peter: "Virtuelle Unternehmen, eine Organisationsstruktur für die Zukunft?" In: technologie&management 44 (1995)2, p. 61-68.0
Strengthening XML Architecture Utilizing XML Schema Artifacts for B2B E-Commerce Applications

Eugene Xavier. P.
Education and Research Department,
Infosys Technologies Limited, (NASDAQ: INFY)
35, Sardar Patel Road, Alexander Square,
Guindy, Chennai – 600 032.
Tamil Nadu, India.
Tel: 91 44 230 0031, Fax : 91 44 230 0091
eugene@infy.com

ABSTRACT
Establishing common vocabularies makes it easy to build software that processes information according to a clearly defined set of rules. As computers are doing most of the XML processing, expressing those vocabularies in a form that computers can understand has become important. The formal description must be regular, unambiguous, and relatively easy to process. On the human side of the information interchange equation, formal descriptions of vocabularies can provide a core set of rules for all participants in a series of transactions. Schemas can make it clear which kinds of information are required or optional for which kinds of transactions. This paper discusses the design considerations for maximizing the utility of XML Schema language in creating documents for B2B E-Commerce Applications useful in Enterprise Application Integration (EAI). In discussing the usefulness of XML Schemas, it touches upon many related technologies, but restricts itself to the technological enabling of e-commerce, avoiding discussion of the many legal and business issues as having no direct bearing on schema design.

Keywords
XML Schema, DTD, EDI, interoperability.

1. INTRODUCTION
As XML use spread among a wider range of programmers, it has become clear that the document-oriented tools provided by DTDs are not up to the challenge of describing more complex kinds of data structures [10]. DTD syntax, unlike XML document syntax, is not easily extensible. DTD information has also proven more difficult to manage with tools like the Document Object Model as well. DTDs are structured differently from XML documents. The importance of XML schemas in developing robust, extensible business-to-business (B2B) e-commerce applications needs to be understood properly.

Use of XML brings the key advantage of providing DTD-based validation using generic tools. However, DTDs do not have the data-typing features, which are essential for Electronic Data Interchange (EDI) applications. This is an area in which XML schema languages excel, making them an enabler for XML/EDI application design.

Schema languages not only provide the rich data typing associated with ordinary programming languages, but also include the capacity for new types to be defined [3]. This means that data typing can be customized for particular application contexts, e.g. to enforce particular number formats, field lengths, etc. The second key advantage of schema languages is the provision for breaking a schema into separate components [1]. This encourages reuse of existing definitions, leading to a greater chance that interoperability can be achieved.

There is a lot to be gained from looking at existing EDI frameworks and standards. EDI applications have been in use for years, so the analysis to define data types has been well tested in the field. Existing type definitions can be used as a starting point while developing the schemas. The key issue is to get the data types clearly defined and agreed with trading partners. Structural differences are less important, as they can be removed by transformations. Data types should be defined with extension and refinement in mind.
2. XML DTD (DOCUMENT TYPE DEFINITIONS)
XML's use of SGML DTDs made it possible to use XML documents in the existing base of SGML software. At the same time, however, XML's allowing documents not to have a DTD at all makes it much easier to question the need for DTDs and to ponder very different replacements for DTDs. DTDs provide an excellent set of tools for describing document structures, as well as a powerful set of tools for reusing descriptions. It is not necessarily what a developer working with Object Oriented Programming (OOP) is used to, but they have got a lot of power. Between the lack of clear explanation in the specification and the fact that most users of XML come from outside of SGML, much of the power of DTDs goes unused. Schema for Object-oriented XML uses techniques from object-oriented programming, notably inheritance, to define XML document structures [9]. In addition to all the XML-based schema varieties, there are database structures, Java class structures, flat-file text documents, and lots of other things out there. Some kind of automated any-to-any model seems necessary.

3. SOFTWARE FRAMEWORKS WITH XML SCHEMA
3.1 Coagulation of Technologies - XML and EDI
The emerging trends in business-to-business e-commerce are demanding greater levels of interoperability for E-Commerce Applications and EAI. This is very much a continuation of the original vision of SGML: standardization enabling the use of complex information across platforms and applications. The radical redefinition of the domain across which information may be reused has raised the stakes, and places much higher demands on the technology. Of all current standards, XML Schema is the most important in living up to this challenge. The power of XML is also its greatest weakness: on the one hand, we can describe any data that our Internet applications care about, to whatever level of detail is needed. On the other hand, we all need to agree on what that data is. XML alone is not enough.

The e-commerce world is clearly divided into two camps: EDI and XML.EDI (X12, UN/EDIFACT) is an early technology that helped many large companies realize a tremendous vision, generally at great expense [11]. The vision of electronic purchasing is very seductive: it increases efficiency, saves money, and offers capabilities that were not available to people working with traditional systems. EDI - used in its restrictive sense, because in truth we are all involved in electronic document interchange regardless of our preferred syntax - is the only way to realize this vision for many years.

EDI is most notable in two ways: implementers from the EDI world have ten years and more of experience wrestling with the problems that have gone mainstream with the popularity of Internet-based B2B e-commerce. EDI messages today provide us with the best standard descriptions of practically useful semantics for e-commerce. The more negative aspects of EDI languages include tag bloat and syntax overloading [4]. As implementation guidelines are not formally explicit, there is a high degree of latitude in how any given segment is used. Sometimes, developers simply use things for non-standard purposes. While it is true that a finite set of tags can describe all useful data structures, this approach generally results in a very large tag set that is difficult to learn and utilize. EDI has embraced this approach, and it has caused many problems with non-standard implementations. EDI provides us with examples of messages that simultaneously provide multiple ways to encode a single bit of information, implemented such that some of the standard constructs are overloaded.

Ultimately, many of these problems stem from having a syntax that does not have a mechanism for formal validation. There is no easy way to determine conformance, resulting in very high integration costs, tag bloat, and overloading. XML represents a relief from some of the more negative aspects of EDI [5]. It has many well-known minor benefits: it is cross-platform, it understands Unicode natively, it is human-readable, etc. These are not the specific items that set it apart from EDI in the most important way, however. XML - as opposed to SGML - increasingly has good tools support.

Browsers, development tools, code libraries, distributed programming objects like COM and Javabeans, repositories, databases, editing tools - the list of XML-enabled software packages is long and growing. These are not all specialized tools for working with just XML - in many cases, traditional enterprise tools have expanded to support XML. Not only will this reduce the cost of software, but also the cost of integration and system maintenance - XML programming skills are becoming increasingly more common.

A negative aspect of XML technology is that it comes to us from an information-publishing tradition, with its own set of inherent assumptions about what aspects of functionality are important. This is most evident when we consider how XML DTDs are used to validate documents: they can validate the structural aspects of documents with great power, telling us where the rules have been violated, and how [6]. What they fail to do usefully is datatypeing: there is one basic string datatype used for leaf-node element content, and a set of strange and not terribly useful datatypes for attribute values loose numeric types, strings without whitespace, etc.

The key thing to realize here is that EDI messages are documents, in one sense, but they are also something more - they are messages which require very strong data validation to be useful in an e-commerce setting. One can look at e-commerce as a set of relational systems talking to each other over the wire with documents/messages. It becomes easy to see why having strong data validation - the kind of validation that evolved in the EDI world around codelists, for example - is so important. XML has not been built to do this kind of validation.
3.2 XML Schema - Backbone to E-Commerce Application Framework

XML schemas allow us to solve many of the technical challenges presented by e-commerce requirements, and to leverage the best of EDI technology. Schema languages contain native datatypes that align with the standard datatypes of some programming languages: strong numeric types like signed integers, floats, doubles, etc.; enumerated lists; date, time, and datatype; and, of course, strings.

XML Schema languages such as SOX, XDR, and the W3C Recommendation for XSDL further allow us to describe custom datatypes that are even more useful for particular applications. It becomes possible to capture code-lists as enumerations, to limit the length of string types to fit into typical database fields, and to describe numeric types very exactly.

Component reuse is very important while seeking to produce interoperability, which simply helps to guarantee leaf-level correspondence across business documents [7]. Another aspect of XML schema that helps to solve the tough problems of e-commerce is the ability to not only reuse another person's data models, but to refine them. In the best case, this mechanism gives access to the chain of inheritance of a given data structure, in the same way that class-based object-oriented programming languages do.

One person's data type or element structure can be taken, and it can be subclassed by adding pieces that are needed specifically. At runtime, processing applications can identify what the differences are between the parent class and it's subclasses, enabling polymorphic processing, that is, using a child in place of the parent, where the parent is required.

The application of this capability from a design perspective is that we can build minimized components in anticipation that they will be extended to fit particular requirements. For supporting Global interoperability, the ability to extend, reuse, rename, and refine other people's components is a major enabling technology [12]. While the capability alone is not sufficient - we must have well-documented and preferably automated methodologies for doing this - it does give us the raw power to solve many difficult problems in accommodating required variation within a class of business documents.

Schema, when properly employed, also makes it fairly easy to translate document structures into normalized relational structures. If we remember that e-commerce can be seen as a system in which relational systems trade information by transmitting documents, the graphing capabilities of XML schema languages can provide us with lots of power. DTDs could not do this effectively - especially since the strongest datatyping existed in attribute values, and the simplest way to handle graphing requirements has been to use element structures [8]. DTDs just are not designed to support this requirement. It is always a good idea to remember what your end goal is, and when you are creating XML Schemas to describe business documents, the basic requirement is simple: interoperability.

The basic steps for creating document structures such as schemas and DTDs are very similar to those of any kind of software application or systems development. Defining System-Level Requirements includes looking at what users need, what kinds of software systems are involved, and what kind of information the design is being made for. In the case of traditional DTD document analysis, the scope of the exercise is generally the enterprise, or at most a particular industry.

In the case of e-commerce schemas, the trading community is the scope. It is much larger, has more users, and generally presents with a larger problem space. The end result of this activity is a set of requirements that will point out what kinds of documents that need to be created, and what the design principles will need to be. In the case of e-commerce, it is important to focus on the business process that is being enabled, as single document types are not being looked at, but groups of documents that make up the exchanges in dialogs.

The next step us to create a design which is an activity that involves looking at the requirements and figuring out how they are going to be met. While doing XML schema development for e-commerce, it need not be started from scratch. The last thing that is needed is another one-off XML business library. The end result of this activity is a thorough written record of the design, including commentary on what sources are used and what has been determined about them. This document should be created in reference to the requirements and design principles. Developing the Schema involves choosing a schema language sufficient to meet the needs, and producing a first draft of the documents themselves.

Most schema languages have a facility for putting commenting inside the code. This is useful, but not enough. Thorough and professional documentation based on the design documents and the records of the development cycle need to be produced. Publishing the work is best done through the public repositories that are being established: XML.org, BizTalk.org, etc., but will depend on the particular scope of effort. The particular trading community may have its own points of distribution.

Data types are a core feature of XML Schemas, but the typing approach has been extended through the specification. Types can be built from other types, nested, modified. The distinction between declaring types and declaring elements also adds flexibility. While DTDs consist mostly of type declarations there are only a limited number of types and restricted mechanisms for constructing new types. Developers used to working with object-oriented development seem to find XML Schemas much
more appealing than DTDs, although more document-centric folk are often bewildered by XML Schemas.

Schemas effectively described linked collections of structures [2], along with rules for how those structures may be used. This model may look very little like the final document instances, but its formal structures contain those document instances as possibilities. Typing and inheritance promise to simplify the reuse of document structures across document types. Data types in particular are easy to reuse, but more sophisticated structures may be next. Eventually, this might lead to more horizontal or bottom-up vocabularies.

Like XML 1.0 DTDs, XML Schemas provide the ID type for identifiers that must be unique within a document, IDREF for values that must match one of those identifiers, and IDREFS for cases where more than one value might be referenced. XML schemas add the concept of keys to the ID/IDREF mix, using XPath information to add requirements to documents. Developers used to working in an object-oriented environment may find themselves at home in XML Schemas. While some purists may not like certain features of Schemas - equivalence classes, or the extra pieces for uniqueness, a solid grounding in the world of objects can make XML Schemas much more comfortable.

4. CONCLUSION
XML Schema includes extensive support for type-inheritance, enabling the reuse of previously defined structures. Using facets, new types can be derived that represent a smaller subset of values of some other types, for example, to define a subset by enumeration, range, or pattern matching. When HTML was first published, the hypertext experts of the SGML world claimed that it was insufficient to do serious hypertext. No doubt they were right, but the sheer demand for any amount of functionality in this area, and the simplicity of HTML, made it a phenomenon. Now, more serious applications of Web-based technology are driving the data formats used in more complicated directions.

Undoubtedly, we will make many mistakes in the use of XML-based technology in the e-commerce arena, and hopefully learn from them. It is also true, however, that the use of XML schemas is a critical part of developing usable systems in this area, and today is a focus for the development of useful data structures for Internet-based trading. The design considerations discussed in this paper for maximizing the utility of XML Schema language in creating documents for B2B E-Commerce Applications would be very well useful in EAI.

REFERENCES
1. XML Schema Part 0: Primer Web site On-line at http://www.w3.org/TR/xmlschema-0
8. Datatypes for DTDs (DT4DTD) 1.0 Web site On-line at http://www.w3.org/TR/dt4dtd
Towards a Web-centric Legacy System Migration Framework

Ying Zou and Kostas Kontogiannis

Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, ON, N2L 3G1, Canada
{yzou, kostas}@swen.uwaterloo.ca

ABSTRACT

With the rapid growth of B2B and e-Commerce activities, the “Service Web” has emerged as a promising framework to address the issues related to process integration. In this paper, we present a stack of interrelated protocols, such as SOAP, WSDL, and UDDI, to support service integration using the Web as the medium. Specifically, we aim for a framework that allows for the identification of reusable business logic in large legacy systems in the form of major legacy components, the identification of interfaces between the legacy components and the rest of the legacy system, the automatic generation of CORBA wrappers to enable for remote access, and finally, the seamless interoperation with Web services via HTTP based on the SOAP messaging mechanism.

1 INTRODUCTION

With the rapid global adoption of B2B and e-Commerce activities worldwide, the concept of the Service Web [8] aims at empowering the business process integration over the Web. To facilitate the use of the Web for business process integration and collaboration between trading partners, the “Service Web” builds on the top of components referred to as Web services [4]. Such Web services offer specific business related functionality, reside in the application servers, can be programmatically integrated with other systems, and perform interactive tasks involving multiple steps on a user’s behalf.

To allow for Web services to interoperate, a common industry standard, such as SOAP (Simple Object Access Protocol) [1, 5], WSDL (Web Service Description Language) [2], and UDDI (Universal Description, Discovery, Integration) [3] has been proposed. In a nutshell, SOAP, encoded in XML and HTTP, is a message passing mechanism and serves as a uniform invocation mechanism between Web Services. The WSDL describes the interface points of the Web services that are further indexed in searchable UDDI [6].

For many organizations, the task of renovating their legacy systems towards such a new architecture advocated by the Service Web is not an easy one, due to the significant risk and investment required to re-implement large portions of their legacy software systems to comply with network-centric technologies. An alternative technique to renovation is the reuse of the business processes in the form of wrapped components.

In this paper, we present a framework to migrate legacy systems in a new Web enabled environment where they are considered as distributed services. It consists of two major steps. In the first step, the monolithic legacy systems are decomposed into software components, and migrated from their standalone form into a CORBA distributed objects by automatically generating wrappers. In the second step, a layer between CORBA objects and the SOAP protocol wraps the CORBA object requests with SOAP messaging. In such a way, SOAP becomes a unified mechanism to tie the migrant legacy services with other systems and to coordinate the invocation between the different service entities.

This paper is organized as follows. Section 2 provides a brief overview of standards for the support of Service Web. Section 3 discusses the concrete approach for the migration of legacy system into Service Web. In the end, section 4 gives conclusion.

2 STANDARDS FOR SERVICE WEB

A large-scale legacy system can be divided into four layers, including standards and guidelines at lowest level, basic common services, value-added functional services, and mission-specific services [7]. The levels of basic common services and value-added functional services can be reused as common facilitators in an application domain. In this case, it is valuable to migrate such blocks into the Web enabled environments as Web Services to maximize the accessibility and reusability of software components.

To define the infrastructure of the Service Web, a stack of interrelated standards, such as SOAP, UDDI and, WSDL are presented. Such standards are denoted in XML, and therefore are extensible, flexible and, can be rich in semantic content. Moreover, they can be transported via the HTTP over the existing Web
infrastructure. An overview of these technologies is provided below.

**SOAP**

SOAP provides a simple and lightweight mechanism to exchange structured and typed information, such as commands and parameters, between HTTP clients and servers. The SOAP protocol consists of three parts [5] namely, the SOAP envelope, the SOAP encoding rules, and the SOAP RPC representation.

The SOAP envelope carried by HTTP conveys the message content for the application-to-application communication, for example, the request or response in a client/server model.

The SOAP encoding style specifies the rule for serialization of application-defined data types. The encoding schema specifies simple (scalar) types and compound types. It is a generalization of the common features in type systems, including programming languages, databases and semi-structure data.

The SOAP RPC representation defines a binding between SOAP and RPC to make remote procedure calls and responses.

SOAP is an application level protocol standard that utilizes HTTP, and therefore enables for pervasive access. Moreover, SOAP defines a simple, yet flexible messaging mechanism without the complex object model as CORBA and DCOM. In a nutshell, SOAP is responsible for the delivery of Web service requests/responses between clients and servers. Once the message conformance to SOAP is received by the server, the concrete implementation, invocation and execution of Web Services are based on the distributed object technologies, such as CORBA, Java RMI, Java Servlet, Java JSP, or RPC. The SOAP run-time engine runs in an application server, marshalling or unmarshalling the SOAP based requests and responses.

**WSDL**

WSDL is an evolving specification partially based on Microsoft’s SOAP (Simple Object Access Protocol) Contract Language and IBM’s NASSL (Network Accessible Service Specification Language) [4]. WSDL precisely describes programmatic interfaces of Web services in XML. Meanwhile, WSDL can be used to generate the proxy objects for the Web services in the client and server sides [6]. The language is limited to message formats or network protocols that conform to SOAP 1.1, HTTP get/post and MIME (Multipurpose Internet Mail Extensions).

**UDDI**

The UDDI is a Business Registry standard for indexing Web Services, and enables businesses to find preferred services and transact with one another. In this context, Web Services can be registered in a standard way in service repositories and be located at run time by client processes. The information required by UDDI to register a service is structured in three sections [3] namely, white pages containing company descriptions, contact information, and business ID numbers; yellow pages with companies organized into business categories, such as products, geographic regions, and industry sectors; and finally green pages, which will provide transaction requirements. The standard is based on extensible markup language (XML), and HTTP and Domain Name System (DNS) protocols. UDDI addresses cross-platform programming issues through its adoption of the Simple Object Access Protocol (SOAP) messaging specifications.

### 3 MIGRATION OF LEGACY SYSTEM INTO SERVICE WEB

Within the context of re-engineering legacy systems into a Service Web environment, it is important to analyze the legacy software to recover its major functional components that relate to an application domain and specific business logic. Reverse engineering techniques that allow for the identification of such components have been investigated in depth by the research community. Once specific legacy components have been identified through the use of program analysis, their behaviors can be specified in terms of well-defined object oriented interfaces. In the context of our research, a wrapping technique was adopted for the deployment of the identified components into a distributed environment based on CORBA infrastructure. Although CORBA IDL is a platform-independent standard, it requires the communication entities have the ORB brokers installed before the services can inter-operate. This requirement is difficult to meet in the Web environments, due to heterogeneous web-enabled devices, clients, and protocols that are supported in different sites. The SOAP messaging mechanism is the preferred choice for the migratory CORBA services to be accessed in such a wide diversity environment. In the following subsections, our approach for the utilization of these technologies is discussed in more detail.

#### 3.1 IDENTIFYING LEGACY COMPONENTS

To identify software components that maximize reusability and flexibility, our approach aims to transform selected procedural code from a legacy component to a new object oriented architecture where the interfaces of the component are clearly defined and the related operations that deliver specific functionality are fully encapsulated in classes and wrappers.

The first step towards the migration of a procedural system to an object-oriented architecture is the selection of possible object classes by analyzing global variable data types as well as, complex data types in formal parameter lists. Analysis of global variables and their corresponding data types is focusing on the identification
of variables that are globally visible within a module. A
module is defined as a consecutive set of program
statements that deliver particular functionality and can be
referred to by an aggregate name [9], for example a file
or a collection of files. Any variable that is shared or is
visible (that is, it can be fetched and stored) by all the
components of a module is considered as a global
variable. Clustering techniques and architectural recovery
techniques presented in [10, 11] can be used in order to
obtain an initial decomposition of a large system in terms
of module components. For each variable its
corresponding type can be extracted from the Abstract
Syntax Tree, and a candidate object class can be
generated.

Once an initial set of candidate classes is identified,
then the methods to be attached to the classes can be
identified by a set of evidence models, such as function
return type analysis, state modification analysis, use
analysis and metrics analysis.

This task can be automated to a large extend using a
number of different software analysis techniques. However, no matter how sophisticated the analysis
techniques are, user assistance and guidance is crucial on
obtaining a viable and efficient object model. Significant
domain information can be utilized by the user to guide
the discovery process, locate the common functionality in
an application domain and obtain a better and more
suitable object model.

3.2 MIGRATING COMPONENTS INTO
NETWORK CENTRIC ENVIRONMENTS

Once a software component has been extracted from a
legacy system, its interface can be extracted and
represented in XML. The CORBA IDL and object
wrappers can be automatically generated by the wrapper
generator from the XML-based specification.

Representation of Legacy Services

To wrap a legacy service, the first step is to determine its
interface. The interface description allows the
implementation details inside the component to be
hidden from its external clients. In general, the interface
of a software component may contain the information
related to data types, references to external specifications
that point to related components, descriptions of public
attributes and methods, and return types and parameters
that specify input and output.

The representation of a component interface can be
independent of a specific programming language. OMG
IDL provides such a language-independent interface
description for distributed CORBA components. However, in order to automatically generate wrappers, a
specialized IDL parser is required to access the interface
information.

In our approach, the XML interface representation of
a software component is adopted in order to encode
interface information. Figure 1 illustrates an XML based
component interface representation. It consists of several
aggregated elements, such as data type definitions,
interface operation definitions, and interface data
member definitions. The data type definition section
publishes the data types in the component other than
those in its defined interface. The interface data member
definition declares the accessor and mutator methods
associated with a data member. Such specification aims
on automatic generation of OMG IDLs and CORBA
wrappers. Furthermore, with the XML specification,
additional information such as, reflection information,
URL address, pre- and post-conditions, and performance
characteristics can be easily denoted for a given
component.

Automatic Generation of OMG IDL and
CORBA Wrapper

Once the interface is defined, the process of generating
CORBA wrappers is similar for every identified legacy
component that is being migrated. The wrappers
implement message passing between the calling and the
called objects, and redirect method invocations to the
actual component services. The concrete process to
accomplish wrapping is accomplished in terms of three
major steps.

The first step focuses on automatic generation of
CORBA IDL interface for the identified components. In the XML interface specification, the signatures are indicated in terms of C++ types. Specifically, the IDL generator reads the XML script and converts the interface descriptions to IDL types, conforming to the mapping rules of the OMG IDL specification, and writes IDL style interface in a new file.

The complete mapping of basic data types is given in [13]. For example, C++ “long” is mapped into IDL “long”, C++ “char *” to IDL “string”. Due to the platform independence of OMG IDL, some of basic C++ data types are not specified, such as “int”, which is 16 bits in MS-DOS and Win3.1, but is 32 bits in Win32. For the identified component, we assume it works under a 32 bit operating system.

The second step deals with the CORBA IDL compiler to translate the given IDL specification into a language specific (e.g. C++), client-side stub classes and server-side skeleton classes. Client stub classes and server skeleton classes are generated automatically from the corresponding IDL specification. The client stub classes are proxies that allow a request invocation to be made via a normal local method call. Server-side skeleton classes allow a request invocation received by the server to be dispatched to the appropriate server-side object. The operations registered in the interface become pure virtual functions in the skeleton class.

The third step focuses on automatic generation of wrapper objects as CORBA objects, by inheriting from the skeleton classes. The wrapper classes encapsulate the standalone C++ object by reference, and incarnate the virtual functions by redirecting them to the encapsulated C++ class methods. The wrapper generator maps the CORBA C++ classes used in CORBA objects into native C++ classes recognized by the identified components using the XML interface specification for the component being wrapped. Since the primary data types are passed by value, the parameters from the client request methods can be directly passed into native C++ operations, according to the same mapping rules from C++ to IDL.

For the complex data types, such as class, string, array, the parameters are passed by reference. The signatures of such types in wrapper classes are defined by pointers. In this case, a method is added into each of CORBA wrapper objects to accomplish the translation from a CORBA C++ wrapper pointer to a native C++ class pointer.

### 3.3 Wrapping CORBA Objects with SOAP

To enable the universal access to CORBA objects without installing ORB run-time capability on the client-side, especially the thin clients, a lightweight, Web-native approach is critical for the success. In this context, the SOAP protocol is adapted as a uniform messenger between CORBA objects and the rest of Web services.

The layered structure is illustrated in Figure 2. The Web client sends their request, conformance to SOAP, through HTTP to a Web server. The Web server redirects the request to the servlet engine, which is responsible to process the HTTP/POST and HTTP/GET requests. The SOAP run time engine is apt to handle the interpretation and dispatching of SOAP message. At the heart of the structure, a SOAP/CORBA IDL translator, is implemented as a collection of Java Beans to process inbound SOAP requests, which represent invocations to back-end CORBA objects. Internally it encapsulates all the services like SOAP serialization and deserialization, bridges the gap between the SOAP RPC and CORBA IDL and finally, generates the stub to dispatch the requests to the back-end services on the client’s behalf. Meanwhile, the WSDL of back-end CORBA Server can be automatically generated from the component interface specification, shown in Figure 1, and can be registered in the UDDI central repository.

### 4 Conclusion

With the occurrence of Service Web, a stack of interrelated standards based on XML has emerged to provide the language-independent data representation and platform-neutral support for the Web services integration.

This paper presents a framework to address the issues on migrating legacy systems into a Web enabled environment. The approach consists of three major steps. First, a legacy system component is leveraged by identifying parts of a legacy system that relates to a specific application functionality. Second, a CORBA wrapper generator facilitates the rapid movement into the distributed environments, and enables the migrant services remote accessible. Finally, a SOAP/CORBA IDL translator layer bridges the gaps among heterogeneous implementation technologies for Web services, and solves the universal integration issues by SOAP and HTTP. A prototype system that supports the framework presented in this paper is currently developed.
REFERENCES


[2] “Web Services Description Language (WSDL) 1.0”,


IBM Toronto Lab.


[6] IBM Web Services Toolkit,


http://www.extreme.indiana.edu/soap/design/.