Software System Generation from an Enterprise Service Model

D.D. Cowan,[1] P.S.C. Alencar,[1] D.B. Brown,[2] H.D. Covvey,[1] I. Gimenes,[3] C.J.P. Lucena,[4] W.J. Malyk,[1] D.W. Mulholland,[1] A. Robins,[1] K. Young[1]

Abstract

This paper introduces an approach to constructing information systems based on enterprise service architectures (ESAs). An ESA model is specified using declarative business process modeling or workflow commands in conjunction with service framework instances where the business process model and the instances are captured in a data structure. A service framework is an outline of different types such as reports, databases, agents, diagrams and maps and an instance is produced by completing a framework's adaptation points. Adaptation points are defined by an adaptation interface specified in a declarative language and presented through forms. The business process or workflow is specified similarly. An automatic transformation is then applied to convert the model into an operational information system.

This approach can be viewed as model-driven software development without the necessity of writing code to complete the implementation. Thus, we avoid the symptoms of architectural drift that occur when applications evolve independently from their model. The control structure is declarative and separate, clearly delineating the concerns, thus making it easier to address future anticipated, unanticipated and crosscutting concerns.

We have created tools to support this approach, and although many of the tools are general, we have concentrated our efforts on the Web because of its pervasive nature. Thus, we have produced the Web-based Informatics Development Environment (WIDE) technology, a set of tools and processes used to transform a model based on ESA principles into a web-based information system. We have designed and implemented over 30 operational web-based systems to validate our approach.

Introduction

Models of enterprise service architectures (ESAs) form the foundation for many of the information systems currently under development, particularly those based on the Web. This form of architecture uses loosely coupled independent software services accessed without knowledge of the underlying implementation. A wide variety of services and legacy technologies such as databases, content management systems, and geographic information systems (GIS) compose such systems. There are many approaches to the construction and composition of information systems based on ESAs; however, they still often embed low-level programming constructs [1, 2], rather than higher-level models.

This paper introduces a new approach to constructing information systems based on enterprise service architectures (ESAs). A model of the system is specified using declarative workflow or business process modeling concepts in conjunction with instances of service frameworks. The process is data-driven in that the workflow or business process model and the instances are captured in a data structure rather than a program. Once the details of the system are complete, an automatic transformation is then

[1] David R. Cheriton School of Computer Science and Computer Systems Group, University of Waterloo, Waterloo, Ontario Canada.
[2] Department of Physics and Computer Science, Wilfrid Laurier University, Waterloo, Ontario Canada.
[3] Departamento de Informática, Universidade Estadual de Maringá, Paraná, Brasil
[4] Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro Brasil

applied to convert the model into an operational information system; there is no need to provide further detailed code other than that generated directly from the outlines.

In this context, a service framework is an outline of different types such as reports, databases, agents, diagrams and maps and an instance is produced by completing a framework's adaptation points. Adaptation points are defined by an adaptation interface which is specified in a declarative language and presented through forms. The concept of frameworks in this paper is general and not restricted to object-oriented frameworks.

Derivation of many different underlying implementations from enterprise service architectures is possible and we have created tools to assist with this process. Although many of the tools are general, we have concentrated our efforts on the Web because of its pervasive nature. Thus, we have spent significant effort on the Web-based Informatics Development Environment (WIDE) technology, a set of tools and processes used to transform a model based on ESA principles into a web-based information system. WIDE consists of a workflow service framework plus a number of other service frameworks that can be configured and composed to generate a broad range of web-based information systems.

We have designed and implemented over 30 web-based systems using this approach, and extended the WIDE technology to encompass new application domains and associated services when required. Three current systems under development using the WIDE approach show its power and are briefly described in this paper. The first such system, the performance indicator monitoring system (PIMS), supports gathering of indicator and work planning data related to smoking cessation programs and is being developed in partnership with the Ontario Ministry of Health Promotion and the Ontario Tobacco Research Unit. The second system, the Stewardship Tracking System, supports inventories of ecosystem restoration projects and is being developed in partnership with the Ontario Ministry of Natural Resources, with the support of GeoConnections, the Oak Ridges Moraine Foundation and the Metcalfe Foundation, and in collaboration with conservation authorities and non-government-organizations in natural heritage conservation. The third system, Project NOW, is an interactive multi-lingual web-based system which provides information and support for immigrants to Waterloo Region and is created in partnership with the Waterloo Public Library.

The original motivation behind the work reported in this paper was to seek ways to lower technology barriers to the implementation, evolution and maintenance of enterprise systems based on services. Lowering technology barriers would make modern information and communications technology (ICT) more accessible to small-to-medium enterprises (SMEs) and non-governmental organizations (NGOs). These organizations often do not make effective use of ICT because of its apparent complexity and associated cost. With this approach, it becomes possible for an application developer with domain knowledge to specify and create an application without knowing all the arcane knowledge associated with ESA and web development.

The systematic approach described in this paper integrates enterprise service architectures with workflow technology. Both the workflow and the adaptation interfaces of service frameworks are declarative in nature and can be implemented using a forms-based approach. The model proposed by Jackson and Twaddle [3] provides a foundation for the declarative workflow where it is captured in a data structure based on entity relationship models rather than as a sequence of program steps.

The method described in this paper, which we call enterprise service systems generation (ESSG), is novel because it derives an implementation directly from a service model; changes are always made to the model and then the transformations are applied. Since the transformations are automatic, there is no need to modify or augment the operational code when a change is required. If there is an error in the workflow (control structure) then the underlying data structure is easily modified. If there is an error in an instance of a service framework then the service framework or its adaptation interface are modified and substituted back into the system without modifying the workflow.

Two other benefits accrue from the ESSG approach. The workflow in conjunction with the service-oriented nature of our approach leads to a methodology for creating pluggable and easily re-usable components for ESSs. As well, the use of the workflow approach brings a discipline to the design of ESSs helping to identify the types of services that need to be present for a complete system.

The control structure is declarative, clearly delineating the various existing concerns in the system. Because control and services are separate, it becomes much easier to identify how to modify the system to address new concerns as they arise. Thus both anticipated and unanticipated system evolution is more manageable.

The approach just described can be viewed as a model-driven software development approach as characterized in [1, 2] without the necessity of writing code to complete the implementation. Thus, we avoid the symptoms of architectural drift that occur when applications evolve independently from their model. ESSG supports isolation of concerns and can handle crosscutting concerns, a topic that has attracted interest in recent years usually through aspect-oriented programming technologies [4]. In our approach crosscutting concerns are incorporated into the model, rather than the detailed code, an approach that supports ease of evolution and maintenance as noted in [5].

In this paper, we first discuss how a software system based on an enterprise service architecture maps into a workflow model, and how the workflow further maps into an entity-relationship form. We then outline the steps in the process of describing the design and implementation of a service-oriented system followed by an overview of how the system can be transformed into operational code. We then focus on the instances of the service frameworks and provides examples of how their adaptation interfaces might be specified. We then describe a specific mapping of the service-oriented paradigm for the web and describe a simple example to illustrate many of the concepts. Subsequent sections contain a description of the WIDE toolkit and a brief description of many of the operational web-based systems that have been implemented using the ESSG approach. The paper concludes with related work and possible future directions for this research.

Background

Before providing details of the ESSG approach, we examine data-driven design, service frameworks and an entity relationship approach to workflow/business process modeling.

Data-driven design versus process-driven design

We follow Jackson and Twaddle's approach to data-driven design where all entities in the system, including the workflow processes and services that manipulate those entities, are first captured in an entity relationship model and then transformed to a set of data structures. An "engine" then interprets or compiles the data structure into an operational format such as that used on the web. In contrast, process-driven design usually relies on a procedural language [6] such as Java, BPEL [7] or UML State Machine Diagrams [8] to express the process or control flow with corresponding programming techniques.

Service frameworks versus object-oriented frameworks

In [9] the authors define a framework as a set of components working together to address a number of problems in one or more domains. In the book "Design Patterns," [10] the authors define an object-oriented framework as a set of cooperating classes that make up a reusable design for a specific class of software. The service frameworks described in this paper fit this definition in that they are a set of components that together compose a service and whose adaptation interface consists of a collection of adaptation points specified in a declarative format. The components can be, but are not necessarily, object-oriented.

Workflow/Business Process Modeling

Workflow describes the composition of services performed by individuals with some automation to support them. In contrast, business process modeling (BPM) describes the composition of transactional services, activities that are primarily automated and dynamically coordinated, and used to deliver a business process [11]. Except for the degree of automation, both workflow and business process modeling address the same issues and encompass the same concepts [12]. Thus, we use one notation to describe both workflow and business process modeling. Services are often interactive and involve both manual and automated operations. For example, selecting a method of payment is manual, but the action of performing the payment is automatic when a credit card is used.

We have chosen the workflow notation as presented by Jackson and Twaddle [3] to describe the models that support workflow or a set of business process activities as epitomized by service-based systems. This approach maps workflow onto entities and relations among services applied to those entities. Thus, the entire control structure related to the workflow maps onto an entity-relationship model. Because the abstractions represented by data are identified clearly, it becomes easier to understand the relationship among entities and services applied to those entities. In the next few paragraphs, we introduce this notion of workflow. A more complete description appears in [3].

Entities

The first step in producing a workflow representation is to develop a data model in terms of entities that are central to the description of the business processes. The data model consists of entities such as customer, product and order. These entities usually have corresponding entries in the information system database.

Lifecycles, Stages and Services

Each entity in an operational system has an associated lifecycle. Each lifecycle goes through a number of sequential stages and each stage contains a number of services, applied to the entity, some of which may be performed in parallel. For example, a customer buying a book online has a lifecycle as he/she proceeds from the stage of identifying and choosing the book, to the stage of payment to checkout. One can think of those stages as "birth," "life" and "death" where the "life" stage might be further divided into sequential stages such as "youth," "early adult," "middle age" and "old age."

Services provide the actual processing associated with entities. Services are interdependent in that the application of one service may have to wait for the completion of services in other stages or lifecycles. Figure 1 illustrates the relationship among an entity, lifecycle, stages and services. Although not shown in the Figure 1, services can be executed conditionally or repeatedly. The model also supports parallel execution through the splitting (fork) and joining (join) of services.

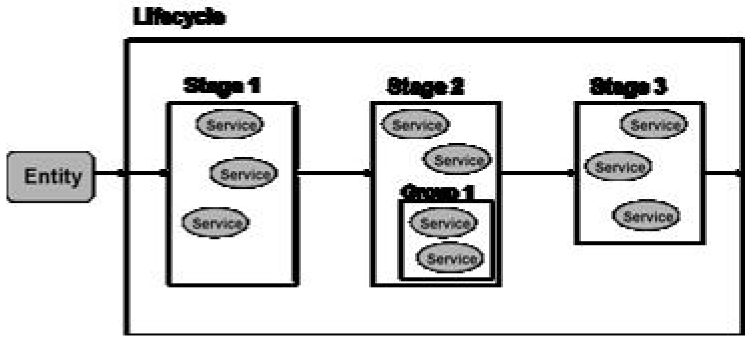


Figure 1

Workflow/Business processes as lifecycles, stages and services

Groups of Services

Services are often closely related. For example, identifying a book and then saving information about that book's identity. Such related services are in the same stage of the lifecycle. Closely related services are often enclosed in a group as a direction to the designer that these services belong together. A group appears in Figure 1 in Stage 2. Grouping not only serves as documentation but is also of value in the generation of applications for environments such as the web where several services may be assigned to the same page.

Workflow as an Entity Relationship Model

Workflow or business process models can be represented as a program structure connecting services, but the lifecycles, stages, groups, services and interdependencies can also be represented in the form of an entity-relationship (E-R) model [13] based on the diagram in Figure 1. Although not exactly identical, entity-relationship models can then be further mapped into a relational database where the concepts in the system (entity, lifecycle, stage and group) and the relations among them each correspond to a table.

Thus except for the operational code associated with services, workflow is represented as a large data structure. Such a data structure makes it possible to present easily the entire process in a graphical format, thus supporting a significant degree of visualization. In the model presented in Jackson and Twaddle [3] the name task is used for service. We have chosen to generalize the notion of task in our approach and part of this generalization is to use the word service.

The structural relationship between lifecycles, stages, groups and services in a workflow or business process model can be modified by changing the values in a data structure rather than changing a program structure. Modifying workflow is much easier, because the abstractions corresponding to the workflow or control structure are clearly identified. Further E-R models can be easily represented by XML tagged structures thus allowing the development of a declarative domain specific language (DDSL) for the description of workflow or business processes.

The declarative representation of the workflow can be transformed into other representations such as BPEL [7], UML [8] or XMI [14] through transformations defined in languages such as XSL [15]. This approach has been demonstrated in [16] where XML-based declarations for agents were transformed into code written in C.

Complex Services

To this point we have described workflow in terms of entities, lifecycles, stages, groups and services. Each service can itself be quite complex. For example, if the service is “clean the house,” then the entity house can acquire its own lifecycle, stages and services such as “wash the dishes” and “clean the floor.” There are several different ways to model this larger-scale service using variants of this model.

To generalize the concepts of lifecycle, stage, group and service, we designate a task as an atomic or elemental service. In the previous example the service “wash the dishes” could consist of several elemental services or tasks such as “load the dishwasher,” “start the dishwasher” and “unload the dishwasher.” In our use of the model, a task corresponds to an instance of a service framework such as a specific report or agent.

Constructing Service-oriented Information System Using Workflow Concepts

In this section, we use a top-down approach to outline the structure and steps involved in creating an enterprise service information system. Subsequent sections provide details on the representation and methods for defining service frameworks and workflow.

An Enterprise Service Information System as a Set of Applications

A service-based information system is composed from a number of interdependent applications, which are themselves composed from services. Each application is delimited by the collection of services that are applied to one class of entity in the system. Entity classes represent collections such as people, organizations, orders, contracts or inventory items. In other words, entities are collections of different objects that form the data model for the information system.

Services as Instances of Service Frameworks

Services are instances of service frameworks, where a service framework is a partially completed software structure intended to be instantiated by adding further structure or program code segments. A framework defines the architecture for a family of services and provides the basic building blocks to create them. The definition of a service is left to the reader, although our definition of service should become apparent from the context.

Categories of service frameworks include reports, agents, geomatic services, multimedia and workflow. For example, the report collection could consist of different types of report service frameworks such as invoices, purchase orders or report cards. Similarly, agents could be classified into copying agents, typing agents or matching agents. Each member of a collection of service frameworks is stored in and retrieved from a related global framework repository. Each instance of a service framework produced for an application is stored in a repository associated with the information system composed from the set of applications.

Certain parts of a service framework constitute adaptation points that are not completely specified when the framework is defined. We call the collection of adaptation points for a service framework an adaptation interface. For example, an agent that copies files does not have the names of specific files and related information such as whether it is a text or binary file. Similarly, a report service framework would not have the query statements defined and a geomatics framework would not specify the initial map to be displayed or what controls should be on the map display. Controls allow the user to manipulate the map display, such as positioning or zoom-in and zoom-out.

The adaptation interface for each service framework is specified using an XML-based declarative domain-specific language (DDSL), which can be transformed into a set of forms to be completed. A corresponding workflow connects those forms and thus prescribes the order of completion. Thus, the forms define the adaptation interface for the service framework and completing the forms in the order prescribed by the workflow produces an instance of a service framework.

Adding a new service framework to an existing collection of service frameworks is straightforward since a new framework is likely an extension of an existing one and only requires some modification and merging of one or more framework descriptions. Producing a new collection of service frameworks is a more creative act as both the framework and the DDSL for the adaptation interface for the collection need to be defined. However, we have produced a large number of frameworks and therefore we expect that the structure of most new frameworks may be inferred from existing ones.

An instance of a service framework is created by completing the forms to specify the adaptation points. For example, an instance of an invoice could have extra columns added or renamed. An application is composed from services, which are instances of service frameworks. The composition mechanism to create applications uses the workflow or business modeling approach already described. This workflow composition mechanism is itself a service framework and its adaptation interface is described in an XML-based DDSL. There is only one such service framework for workflow, which is described more completely later in the paper. Figure 2 provides an overview of how an application is built.

Based on the previous description we describe a detailed sequence of steps to build a collection of applications that can be combined into a service-based information system. Although we describe the steps sequentially, they can be implemented in many different orders and can be re-visited at any time to modify the information system.

[1] ***Determine the entities in the data model***. This step is a modeling activity and requires an in-depth understanding of the domain and the entities required to represent the domain adequately. Customers, inventory items and orders are typical entities.

[2] ***Construct the database corresponding to the data model***. Database tables corresponding to the data model are designed and implemented. Constructing a set of database tables is an instance of a service framework where the framework adaptation interface is described as a DDSL. In this case, the tables and their columns are the framework adaptation points. After the adaptation points are specified they are processed by an XSL transformation to produce the SQL commands to construct the database.

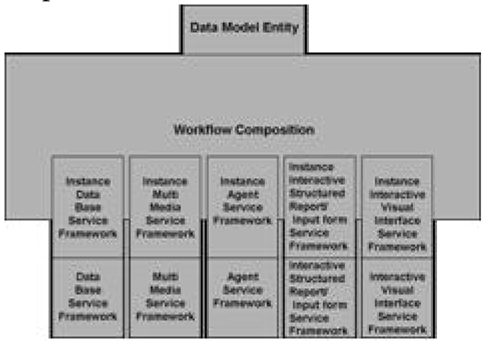

Data Model Entity
Workflow Composition
Instance Data Base Service Framework
Instance Multi Media Service Framework
Instance Agent Service Framework
Instance Interactive Structured Report/ Input form Service Framework
Instance Interactive Visual Interface Service Framework
Data Base Service Framework
Multi Media Service Framework
Agent Service Framework
Interactive Structured Report/ Input form Service Framework
Interactive Visual Interface Service Framework

Figure 2
An Overview of Application Building

[3] ***Construct any new service frameworks needed in the development of the information system.*** New service frameworks and declarative representations for their adaptation interfaces are defined in this step. Examples of service frameworks are

presented later in the paper. Several different types of service frameworks can be used to compose an information system. We have developed declarative representations for the adaptation interfaces for all that we have encountered so far.

[4] ***Activate the workflow service framework.*** Describing the model for an application is itself workflow. We call workflow applied to describing applications "meta-workflow." The meta-workflow is straightforward and is illustrated in Figure 12 with the details appearing in Figure 13.

[5] ***Start creating an application by producing an instance of a workflow by filling in the forms associated with the adaptation interface of the workflow service framework.*** In this step, the workflow service framework prompts us to define an entity, a lifecycle, stages within the lifecycle, and groups within stages and services within those groups. Note that a partial workflow can be saved and completed later.

[6] ***At specific stages of the creation of the workflow instance there is a requirement for an instance of a service.*** At this point in the development of an application, the need for a service is identified. The appropriate service framework is chosen and the details of the adaptation interface are specified to create an instance.

[7] ***Steps [4] through [6] are repeated for every application that is needed to compose the information system.*** This step just ensures that all applications for the information system have been produced.

[8] ***The services that compose an application can be interdependent and must have dependencies specified.*** For example, a book order cannot be completed while quantities of the same book are being added to inventory. Thus, creation of a service as part of the workflow associated with an entity requires specification of its dependencies. Not all dependencies are known when an application is being specified, but we ensure that each service framework is defined to capture dependencies between services. Once all the applications are complete then the entire information system is examined to complete any outstanding dependencies. In this final step, the service in each application is examined to determine if there are any dependencies between services (particularly those in other applications) that have not been taken into account.

Compiling the Workflow Model into an Operational System

Workflow has now been converted to a data structure where only one type of component, the atomic service or task encapsulates the work to be done. The remainder of the data structure captures the relationships and dependencies among the tasks. How can the services represented by those tasks be made operational?

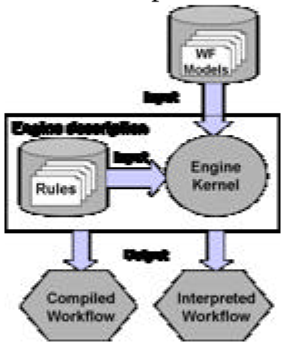


Figure 3
The workflow engine/machine

The approach, which we use throughout this paper, is to create a workflow machine or engine (compiler or interpreter) that is capable of traversing the workflow data structure in the proper sequence including dealing with dependencies. As the machine travels over the data structure it can either interpret the data in the structure or translate the data into operational code, which can then be launched. The services can be manual, meaning there is a requirement for human intervention as in completing a form, or automatic as in processing a credit card payment.

The workflow engine shown in Figure 3 is driven by a set of rules. The rules, which produce more complex behaviours are composed from three basic forms of command, which are the language that drives the workflow engine. The fact that there are three basic forms of command (sequence, conditional and iteration) corresponds to the result of Bohm and Jacopini [17] and Ashcroft and Manna [18] on structured programming. By simple changes to the semantics of these three rules, the engine can act either as an interpreter or compiler.

Constructing Service Frameworks

We describe a set of basic Web service frameworks that can be used to build a large number of applications to form the foundation for our discussion. For example, these service types can be used to build e-commerce or e-health systems using access control, catalogues and shopping carts. Some basic service types are shown in Figure 4.

Data Base Service
Multi Media Service
Agent Services
Interactive Structured Report/ Input form Services
Interactive Visual Interface Services
Workflow Services

Figure 4
Some basic services

Each service type shown in Figure 4 defines a service domain that includes service types for creation and operation of: databases, multimedia reports, agents, interactive structured reports and input forms, interactive visual-interfaces (including maps and diagrams) and workflow. A service framework is an outline for an instance of a service type and a service type may have several service frameworks. For example, there are many different forms of structured reports and each form can correspond to a service framework.

Constructing an instance from a service framework consists of completing the adaptation interface of the framework. This interface is declarative and represented by a set of forms which is completed in a prescribed order and is itself workflow. Thus, the same set of tools that are used to complete an application can be used to specify the steps in producing an instance from a service framework.

Creating instances of specific service frameworks

In this section we outline a general approach to defining adaptation interfaces for service frameworks and producing specific instances of those services. Since the service framework for workflow forms the foundation of our approach, we describe it separately.

Each service framework is assigned to a service type and is then defined through an appropriate development approach. The adaptation interface consisting of adaptation points is specified in a declarative domain specific language (DDSL) related to that service type. The DDSL is written as a set of XML declarations that can easily be converted to a forms driven interface. These forms are used when an instance of a framework is produced. Each service framework and its declarative adaptation interface are stored in the service framework repository shown in Figure 5. In this section, we only describe the adaptation interface to the service framework; we do not describe its underlying implementation. For example, a framework implementation could be in object-oriented or procedural languages, web pages with scripts and applets or combinations of these technologies.

An instance of a service framework is produced by:

[1] providing the identifier that selects the appropriate service framework adaptation interface declaration from the service framework repository;

[2] following the associated workflow steps to complete or modify the adaptation interface of the service framework by filling in the forms accompanying the service framework and produced by the workflow;

[3] translating the service framework and the values specified in the form representing the adaptation interface into an operational form;

[4] and storing a form of the service instance with the system being developed.

It should be noted here that the form of the instance being stored may be compiled into operational code or need further interpretation at run-time. For example, reports are stored in an intermediate form, which is further interpreted at runtime, and this allows dynamic modification to both the structure and appearance of reports.

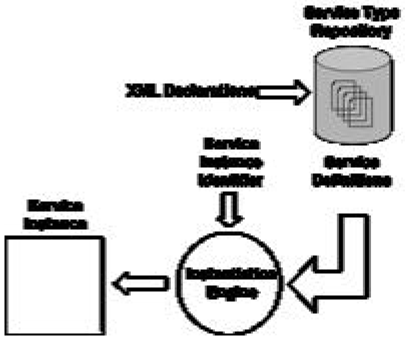


Figure 5

Defining and Instantiating Services

There are two different versions of the instantiation engine as shown in Figure 5 depending on whether the intention is to produce operational code or code that needs further interpretation at runtime. For operational code the instantiation engine is an XSL interpreter and an accompanying XSL script that describes a transformation to be applied to an instance of a specific service framework. When further interpretation is required then the parameters captured in the adaptation interface are stored in a data structure and processed at runtime by an interpreter.

The transformations for the adaptation interfaces for each class of service framework are different, but the frameworks and transformations have any similar characterisitics. Thus, producing a new service framework, its adaptation interface and corresponding transformation should not be a difficult task since a number of sample service frameworks exist and can be followed. Outlines of specific examples of declarative adaptation interfaces are presented in this section.

Databases

The service framework for relational databases contains entity, relation and column adaptation points with optional constraints on the values that can appear in any given column. An instance is created through the adaptation interface by naming specific entities, relations and columns within those entities and relations. Each column is also assigned a value type and constraints on those values. Instead of just using tables, we distinguish between entities and relations as this makes it easier for the designer to map an E-R representation into a relational database consisting only of tables. It also may make it possible to perform validation on the representation at a later stage.

The completed adaptation interface is then input to the instantiation engine which produces a script containing SQL create commands to convert entities and relations into tables. This script can be fed to an SQL engine to instantiate the system tables that represent the database.

Creating the database instance from the service framework can be defined as workflow and one possible workflow is illustrated in Figure 6. Note that there can be several different workflows, which depend on how the adaptation interface for the service framework is declared. Creating a service framework that provides a workflow amenable to a web-based representation is part of the art of creating frameworks.

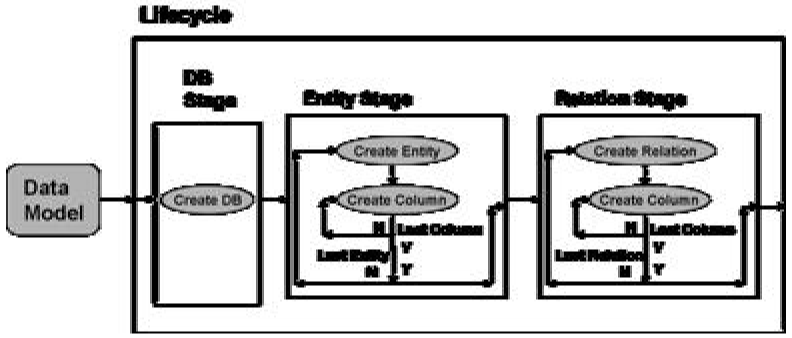
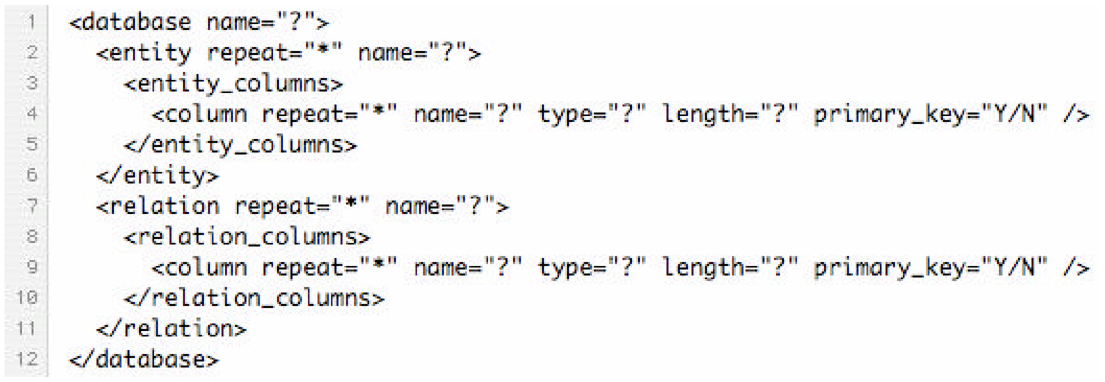


Figure 6
Workflow for a Database Instance



```
<database name="?">
  <entity repeat="*" name="?">
    <entity_columns>
      <column repeat="*" name="?" type="?" length="?" primary_key="Y/N" />
    </entity_columns>
  </entity>
  <relation repeat="*" name="?">
    <relation_columns>
      <column repeat="*" name="?" type="?" length="?" primary_key="Y/N" />
    </relation_columns>
  </relation>
</database>
```

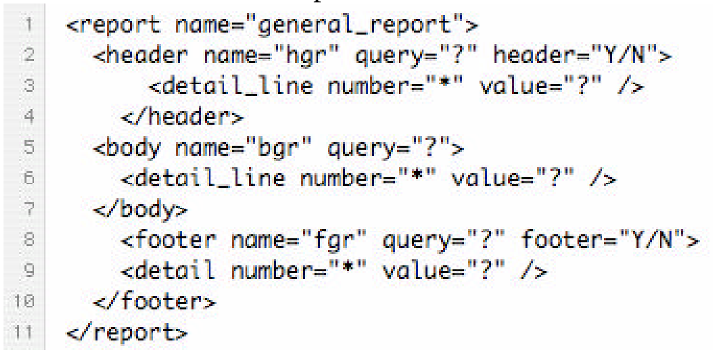
Figure 7
A Service Framework Adaptation Interface for Databases

The workflow provided in Figure 6 implies a data model and corresponding database description that consists of five separate tables: one for each of the database, the entities, the entity columns, the relations and the relation columns. An outline of these five tables is illustrated in Figure 7. The question mark character (?) designates a parameter that must be completed to produce a complete instance.

The XML description for the database service framework adaptation interface is used to generate a set of forms and a workflow that supports the definition of a database instance. The forms are used to capture the names of the database, the entities and relations (tables), and the name of the columns, their type and length and whether a column is part of a primary key. Other parameters are also collected although we have described the basic ones. The forms are generated using the structured reports service framework presented in the next section.

Structured Reports

The service framework for reports contains structures such as headers, bodies and footers each of which is composed of detail lines that contain one or more columns. A simple adaptation interface for a service framework for a general structured report is shown in Figure 8. In this case the parameters of the report are stored in a data structure, which is then interpreted at runtime. The question mark (?) designates parameters that must be completed to construct an instance and the designation “Y/N” indicates whether that specific section is to be included in the report.

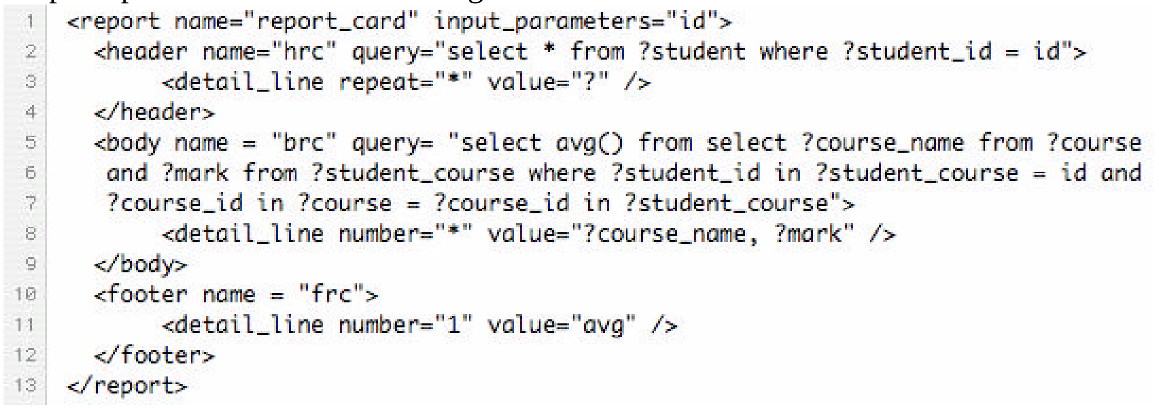


```
<report name="general_report">
  <header name="hgr" query="?" header="Y/N">
      <detail_line number="*" value="?" />
    </header>
  <body name="bgr" query="?">
    <detail_line number="*" value="?" />
  </body>
    <footer name="fgr" query="?" footer="Y/N">
    <detail number="*" value="?" />
  </footer>
</report>
```

Figure 8

A general service framework for structured reports

One could base all reports on this single simple service framework, since report cards, invoices, purchase orders, and even simple listings can all be derived from the same report format. However, since this format must satisfy all possible reports, the queries would be quite complex and would be a significant barrier to the creation of instances of frameworks by a domain expert who had only a moderate familiarity with the nuances of SQL. For example, the query in a simple listing may just access all elements in a single table with certain parameter values, whereas a query for a detail line in an invoice has to join the order information with the product information and then multiply the quantity of the product by the cost to produce a price for each product detail line. For these reasons, we have chosen to create several report service frameworks that match common report categories. An adaptation interface for a service framework for a simple report card is illustrated in Figure 9.



```
<report name="report_card" input_parameters="id">
  <header name="hrc" query="select * from ?student where ?student_id = id">
       <detail_line repeat="*" value="?" />
  </header>
  <body name = "brc" query= "select avg() from select ?course_name from ?course
   and ?mark from ?student_course where ?student_id in ?student_course = id and
   ?course_id in ?course = ?course_id in ?student_course">
       <detail_line number="*" value="?course_name, ?mark" />
  </body>
  <footer name = "frc">
       <detail_line number="1" value="avg" />
  </footer>
</report>
```

Figure 9

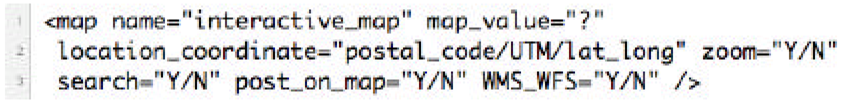
A Report Card Service Framework

The question mark (?) and question mark (?) immediately followed by a name designate parameters that must be completed to construct an instance. The use of the question mark name combination allows the person building the service framework to construct a complete SQL query and identify related elements, thus simplifying the task of later instantiating the framework.

A Geomatic Service Framework

Geomatic services can be quite complex but the corresponding adaptation interface for this service framework is relatively simple because the details of many of the complex controls needed for dealing with maps are hidden from the applications designer. Figure 10 contains an example of a definition of such a service framework called “interactive map” where the map to be displayed initially is a parameter designated by a question mark (?). The remaining parameters are controls that can be activated to add functionality to the basic map. In this case, the controls can be used to:

- locate places on the map and the type of coordinates to be used,
- move closer or farther away (zoom),
- search for locations inside an outline such as a circle, rectangle or polygon,
- post points or areas on a map with associated attribute information.
- support the Web Map Service (WMS) [19] and Web Feature Service (WFS) [20] which are wrappers that allow geographic information systems to be easily accessed over the web.



```
<map name="interactive_map" map_value="?"
 location_coordinate="postal_code/UTM/lat_long" zoom="Y/N"
 search="Y/N" post_on_map="Y/N" WMS_WFS="Y/N" />
```

Figure 10

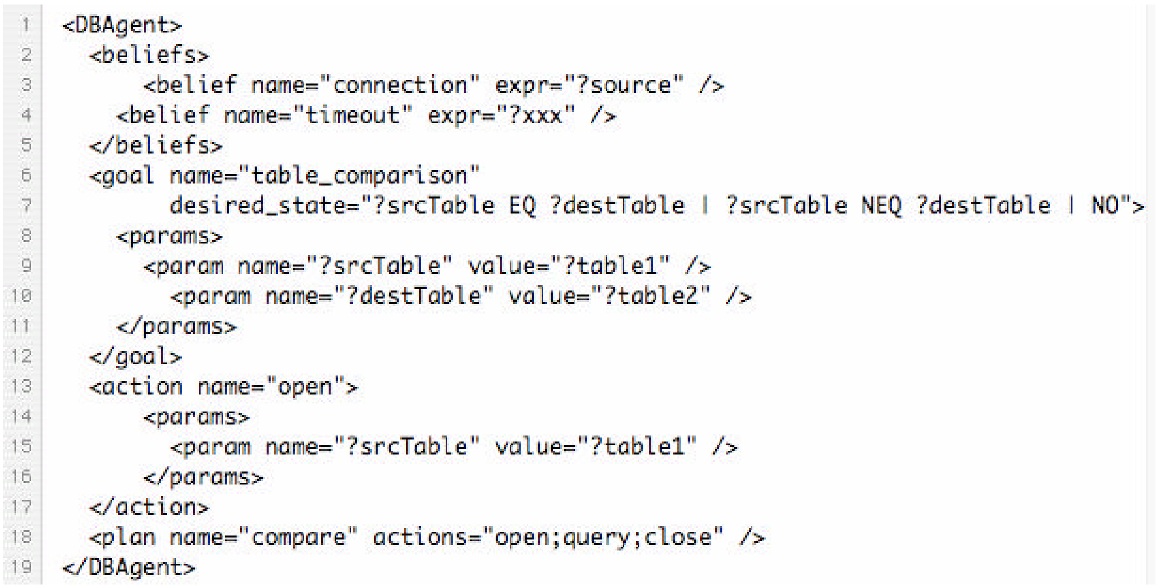
A geomatics service framework adaptation interface definition

Figure 10 is an illustration of a simple geomatics framework adaptation interface. It could also include input forms to collect attributes associated with search or post on map parameters.

Agents

In the context of ESSG, agents are small programs that work almost autonomously to complete housekeeping or background tasks. In general, agents are defined by four basic properties: beliefs, goals, actions and plans [16]. The DDSL for an agent service framework adaptation interface uses these four properties for its definition.

The XSL script first acquires an instance of a specific agent service framework and then produces a script that converts it into a program requiring parameters to operate. The final program is usually in a language such as C, C++ or Java.



```
<DBAgent>
  <beliefs>
      <belief name="connection" expr="?source" />
    <belief name="timeout" expr="?xxx" />
  </beliefs>
  <goal name="table_comparison"
        desired_state="?srcTable EQ ?destTable | ?srcTable NEQ ?destTable | NO">
    <params>
      <param name="?srcTable" value="?table1" />
        <param name="?destTable" value="?table2" />
    </params>
  </goal>
  <action name="open">
      <params>
        <param name="?srcTable" value="?table1" />
      </params>
  </action>
  <plan name="compare" actions="open;query;close" />
</DBAgent>
```

Figure 11

An agent service framework adaptation interface description

We illustrate our approach by defining a database agent that connects to two databases and performs consistency operations between them [16]. Operations consist of tasks such as comparing and copying database content. The service framework definition for the database agent is shown in Figure 11. We could use question marks (?) to represent the parameters but in this case have chosen to use question marks (?) followed by a name for purposes of documentation.

Service frameworks could be defined for many other agents such as an indexing agent framework that indexes words in a document. The only information that needs to be supplied is the set of names of the specific files or Web sites to be indexed.

Service Frameworks for Initiating each Application

At specific times during the operation of an application, there occurs a need to initialize an application or the entire system. Such initialization may require the setting of specific global variables. There are frameworks provided for such activities, which are usually incorporated in the "home" page of the application or the system; we do not provide details here. It should be noted that in a service-based environment, connections to databases are opened and closed as needed and so are incorporated into reports rather than as part of the initialization of an entire application.

Constructing a Service Framework to Support Exceptions

An information system based on an enterprise service architecture can discover, access and use external services. Such an information system must contain instances of service frameworks that can discover services, access them and can handle exceptions in an organized manner. We do not provide details of such service frameworks in this paper.

Constructing a Workflow Service Framework

Describing the model for an application is itself workflow. We call workflow applied to describing applications "meta-workflow." Since the workflow consists of creating a lifecycle that contains several sequential stages where each stage can contain groups and each group can contain several services, the meta-workflow description is quite straightforward. The meta-workflow is illustrated in Figure 12 and Figure 13 where we use the terms meta-entity, meta-lifecycle, meta-stage, meta-group and meta-service.

In Figure 12 a meta-entity can take many forms such as a user, an order, an inventory item, an application, or a form. Each meta-entity has only one meta-lifecycle, one meta-stage and one meta-group consisting of five meta-services. The five meta-services shown in Figure 13 generate an entity, its lifecycle, all the stages in the lifecycle, and all the groups in a stage each of which can contain multiple services.

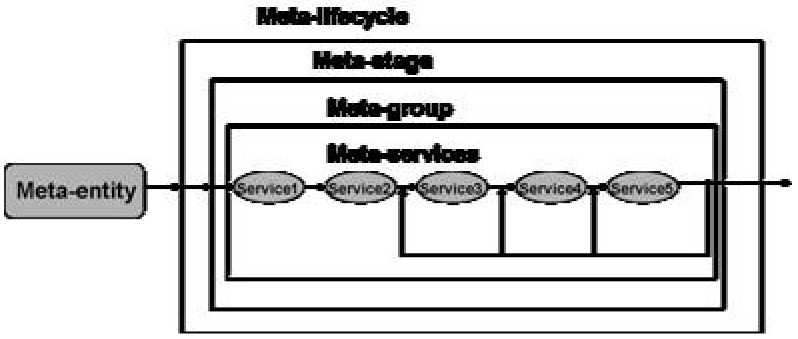


Figure 12
Meta-workflow

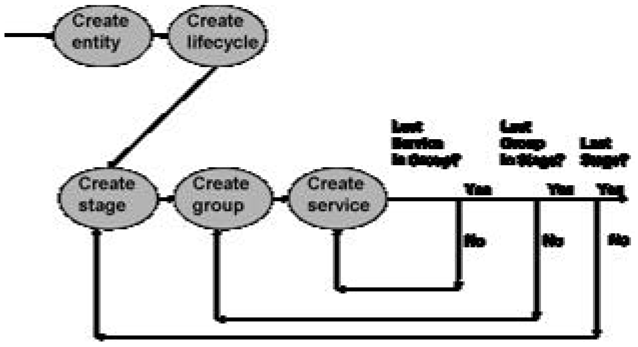


Figure 13
Meta-concern description

The meta-workflow is quite simple; it can be easily converted to a relational database format using the tools provided. The conditions needed to determine if there are more stages, groups or services to be added to a workflow do not need any complicated logic since they are determined by the application designer.

Dealing with Concerns

Separation of concerns is a basic activity in building robust and maintainable software systems [21]. Simply stated it is "divide and conquer;" you deal with complex problems by dividing them into sub-problems that can be solved separately and then combined into a complete solution.

Many concerns can be handled with modularization techniques using modules such as procedures or classes. However, there are certain concerns for which these modules are inadequate. These concerns are called crosscutting as they intersect or crosscut several different modules or building blocks. Typical crosscutting concerns include logging, synchronization and distribution.

Ideally, all concerns should be isolated and handled as early as possible in the specification to avoid costly reworks later when the system is deployed. Further, a well-written requirements specification is characterized by a set of requirements statements where each statement represents a single requirement or concern. Such an argument is presented in [22] where the authors recommend expressing each requirement in one sentence and avoiding complex sentences since they might inadvertently contain more than one requirement.

The approach presented earlier in this paper follows these guidelines for requirements. Concerns that can be described simply are separated into modules such as reports, databases and graphics. These modules are implemented separately as instances of service frameworks and then combined using workflow techniques. Web-based software systems also contain crosscutting concerns. Our approach described in the next subsection deals with this type of concern as well.

Software systems usually evolve over time to meet new requirements that may or may not have been anticipated in the original specification. Subsequent design and implementation need to be revised when these anticipated or unanticipated concerns appear. We address these concerns in the context of our model in the next section.

Anticipated and Unanticipated Concerns

Even though the functional and non-functional requirements for a system are specified, there may be specific ones that cannot be implemented. For example, it may be recognized that there is a future need for creating an audit trail for all accesses to the database but the business rules or legislation may not be in place to specify the detailed audit requirements. Similarly it may be noted that one or more reports may need to display results in both a tabular and graphical format, but users must be extensively consulted through demonstrations to determine the best approach.

These two examples represent concerns that have been anticipated and their eventual incorporation could be built into the initial design of the system. The audit trail concern must be related to every database access, is distributed over the entire design and is known as a crosscutting concern. The graphics concern could be embedded in one or several reports. If this concern is in one report then it is an isolated or local concern, if it is distributed across several reports then it is crosscutting.

Similarly, the two examples of audit trails and graphic reports could have been ignored in the initial requirements specification and thus could have been unanticipated in the initial design. If the designers did not anticipate the appearance of audit trails and graphics, the design might not be as amenable to change to include these concerns.
In the next section, we address isolated and crosscutting concerns and how they might be incorporated into our approach.

Isolated Concerns

Isolated concerns are restricted to one location in the system and can involve changing the structure, changing or introducing an instance of a service framework or a combination of both.

Changing the control structure is quite simple, since it is explicitly described in the workflow structure that has been captured in a database. Introducing an instance of a service framework is straightforward as long as a DDSL for the framework adaptation interface exists. Changing or augmenting an instance of a service framework is a function of the robustness of the implementation that defines the framework.

Crosscutting Concerns

Crosscutting concerns are aspects of a program which affect other concerns and cannot be cleanly decomposed from the rest of the system in the design and implementation. For example, in an application involving medical or financial records, core concerns are examining and changing the records, while logging accesses to those records, or authenticating appropriate access, would be crosscutting concerns since they interact with more parts of the program.

Working with crosscutting concerns when a software system is implemented in a programming language involves several steps. The first step requires identifying all occurrences of the concern with which the crosscutting concern interacts. This identification is usually performed by a search process based on a search pattern known to the program implementer and described in a regular expression. Obviously, the author of the search requires an in-depth knowledge of the program if they are going to identify all such concerns. Once these concerns have been isolated, they all can be modified to reflect the needs of the crosscutting concern.

We believe our approach is much simpler. Since all components are declarative and described with tags, it should be much easier to identify concerns that are being crosscut. Of course, this does require a disciplined approach to choosing tag sets, but is still easier than an equivalent disciplined and less obvious approach to program structure.
Crosscutting concerns are usually characterized as coming before, coming after, or replacing a concern with which they interact. Since the control structure is explicitly defined, it is easy to find all points of insertion. Once the concerns are identified, they can be separated into a new lifecycle as described earlier in the paper. Treating the concern as a new entity with accompanying lifecycle makes it quite easy to insert new stages and concerns, thus enabling crosscutting concerns in a very natural way

Replacing instances of a service framework may mean modifying some existing frameworks or augmenting the current ones. For example introducing graphics into several reports may mean modifying several different service frameworks. As stated in the previous paragraph, the ability to make these modifications easily, relates to the robustness of the framework.

Some crosscutting concerns are related to performance issues. For example, the response to every query must occur within ten seconds. Many crosscutting issues related to performance can be controlled through system deployment. In this example, performance can be improved by distributing data and web services over several servers. Deployment itself can be described through workflow in much the same way that the entire system is built, although we do not address that here.

Augmenting Existing Service Types

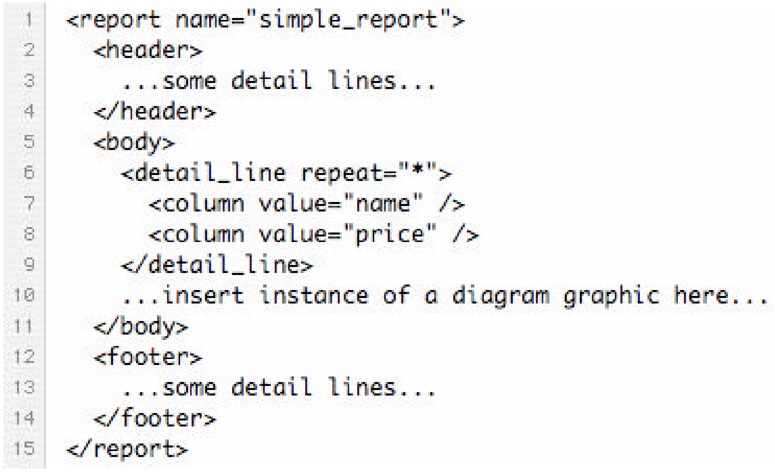
In the previous section we discussed both isolated and crosscutting concerns and we discussed how to modify the code to incorporate new concerns. Specifically we mentioned replacing a service framework that would incorporate a concern. In this section, we demonstrate how a service framework can be designed to allow for inclusion of new concerns. We use a report service framework to demonstrate the approach, but it is completely general.

Since all service frameworks adaptation interfaces are specified in a DDSL based on XML it is easy to insert new declarations at any point between the atomic elements that constitute the adaptation interface of a service framework. Further, a form can be generated from the service framework that can hide the DDSL statements and yet allow insertion between the elements of the framework.

An instance of a partial service framework for a report is shown in Figure 14. We have chosen to change the adaptation interface for the framework by inserting a diagram that illustrates the data reported in the body of the report. This is easily done by inserting a reference to a completed diagram in the report instance. When the instance is accessed during program execution, the graphic instance is substituted in a manner similar to executing a macro instruction. Such an approach works for both isolated and crosscutting concerns. In the case of crosscutting concerns, it is simply a case of determining where the graphics are to be placed and inserting the name of the instance. Locating the graphics is a matter of looking for the reports in which a graphic should appear.

<u>Understanding Workflow in the Context of a Web-based System</u>

Everything described to this point is directed to general enterprise service architectures. However, most service-oriented systems are currently web-based. In this section we describe how our approach can be applied to the production of web-based information systems. We do this in two stages by showing how the web maps into a workflow-based system and then demonstrating the techniques through a simple example.



```
<report name="simple_report">
  <header>
    ...some detail lines...
  </header>
  <body>
    <detail_line repeat="*">
      <column value="name" />
      <column value="price" />
    </detail_line>
    ...insert instance of a diagram graphic here...
  </body>
  <footer>
    ...some detail lines...
  </footer>
</report>
```

Figure 14

An instance of a partial adaptation interface declaration for a report service framework

Mapping the Web into a Workflow-Based System

How does workflow fit into the design of a web-based information system? We propose to answer that question in this section. There are four primary concerns that arise in a hypermedia or web-based information system namely content, style or appearance of that content, structure, and navigation [23]. A good design should keep these four concerns separate so that changes to any one occur with minimal or no effect on the others.

Content

A web-based information system is composed of pages where each page P_i contains <u>content</u> copied from local or remote files containing tagged text. The web browser and related web server interpret the tags to produce the content that is visible on the screen. Some pages may contain mini-applications or applets that exhibit complex behaviour such as querying a database and processing the results of the query. Some pages may not produce any visible content as they are dealing with background services that are not directly relevant to the user.

Style or Appearance

The visible content in each page has an associated <u>appearance or style</u> sheet S_i. The style sheet specifies the appearance of content of a page and can be modified separately from the content.

Structure

Each web-based information system has an initial starting page or "home" page that contains hyperlinks to a number of other pages, which themselves are hyperlinked to other pages and so on. Thus each page P_i has a set of k hyperlinks $<H_{i,j}>^{j=1,k}$ where k can be different for each page. Usually one member of the set of hyperlinks on a page is a hyperlink to the home or initial page. These sets of hyperlinks form the structure of the web site.

Navigation

From any given page it is usually possible to:

- jump to a preceding or succeeding page through web browser controls (such as forward or back buttons),
- go directly to a random page in a website using a hyperlink that has been saved and perhaps shared with others (a favorite).

We call these two methods of moving within a web site navigation.

The original design of the web did not support the passing of state information between pages. However, modifications to the web now allow state information transfer between pages through mechanisms such as hyperlink parameters, posting or cookies.

The forward and back buttons and favorites can refer to a web page, which may need state information from a previous page that is no longer available. Thus, a "favorites" page or the page succeeding the use of a forward or back button may have to contain an applet that determines if the required state information is available.

Workflow

Workflow can be divided into two words work and flow. In a web-based information system, the content of the pages corresponds to the work, while the flow maps directly onto the structure of the web-based system. In effect, the flow corresponds to the hyperlinks.

Verifying State Information

Although the web was originally stateless, it is now possible to pass state information from page to page through different sharing mechanisms. It is also possible to navigate to pages in a web-based system through the back and forward buttons on the browser or through a favorites page. Thus, a page could be entered without the appropriate state information. Each page that relies on having state information from previous pages in a web-based system should contain web code that ensures that the state information is available. Our system contains service frameworks that can be tailored to ensure that each page has appropriate state information before completing its associated service.

An Example - Constructing a Web-based Information System

In this section, we use a web-based system that evolves as two simple interdependent applications to illustrate the basic concepts. We use a simplified version of our approach so as not to overwhelm the reader with details that are not relevant to understanding the methods. The system provides a tour of historic buildings within a city and viewers operate the tour through a web browser on either a standard personal computer or a wireless device. The two applications provide:

1. a list of historic buildings with descriptions.
2. a list of tours which groups historic buildings together based on districts.

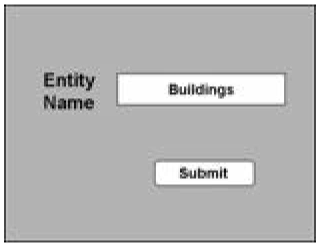
Application 1 – a list of historic buildings

This application consists of a set of historic buildings and so there is only one type of data entity, namely historic building, in this initial application. Each member of the set is one such building and has a unique identifier, a name, an address and a detailed description. The address includes geo-referencing information such as a postal code or geo-coordinates. The description of the building could include a history, audio-visual materials, by-laws, interior layout and outstanding architectural features. Each row in a single database contains the parameters of one historic building.

This first application called "buildings" supports access to a specific building in the list and a display of the relevant description, which we assume, can fit on one web page. There is also a map accompanying the description that shows the location of the building. Since there is only one application, the application and the system are synonymous.

Creating the application requires the completion of each step in Figure 13. This application has two stages that we call the "welcome stage" and the "view stage." As the application develops, it is apparent that the welcome stage has one group while the view stage has two groups.

The first step in producing an application consists of a form, shown in Figure 15 that requires the definition of an entity; in this case, the entity is named "Buildings." Only a name is required for the entity and that name is stored in the database table corresponding to entities.

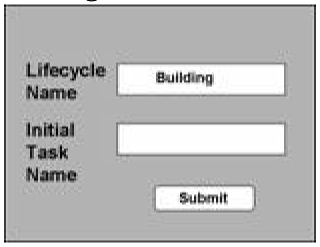


Entity Name	Buildings

Figure 15

A form for defining an entity

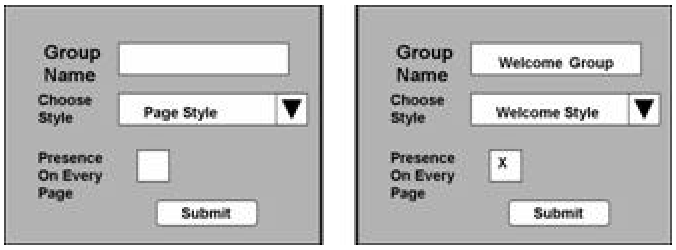
We use a similar form to define the lifecycle for the entity and each stage in the lifecycle and slightly different forms for defining groups. Where we choose multiple stages and groups, we must choose names that are unique within the corresponding lifecycle and stage. In the database, we must link the lifecycle to the entity, each stage to its corresponding lifecycle and each group to its related stage. Thus, the entity-to-lifecycle relationship is one-to-one, while the lifecycle-to-stage and stage-to-group relationships are each one-to-many. The form for defining a lifecycle is in Figure 16 and in this application, its name is "Building."



Lifecycle Name	Building
Initial Task Name	

Figure 16

A form for defining the lifecycle



Group Name	
Choose Style	Page Style
Presence On Every Page	

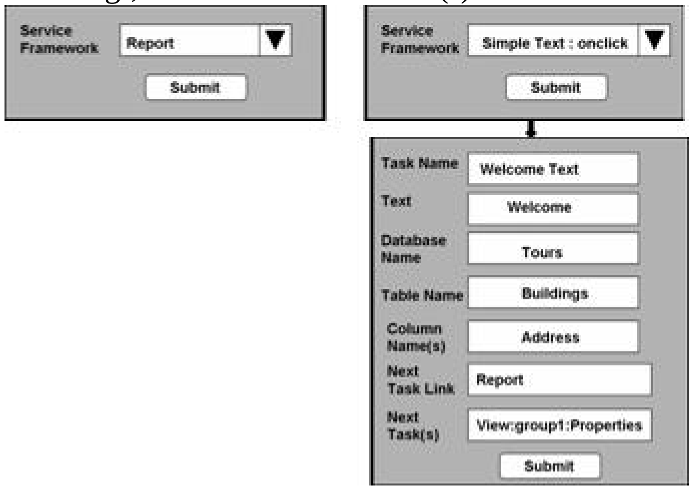
Group Name	Welcome Group
Choose Style	Welcome Style
Presence On Every Page	X

Figure 17
A form for defining a group

Since the form for stages is similar, we will skip it and move on to the form for creating a group which is in Figure 17. Note that as well as providing the group name there is an opportunity to specify a style for the text or other material that appears explicitly on the page, since a group in our context corresponds to page. In addition, there is an opportunity to specify whether the link to this group (web page) should appear in every other group (web page). One can think of this specific group as the "home page" and every other page should contain a link to the home page. Similarly, the developer can create a table of contents by checking the same box in other groups. The style sheet accompanying each group or web page controls the appearance of the table of contents. The form as it appears is on the left with the style chosen and the indication of a presence on every page shown on the right in Figure 17.

Once the initial basic structure is established then we define the tasks in a specific group. The first group, which is the "Welcome Group," corresponds to a welcome page, which contains one task shown in Figure 18 consisting of a block of text, which says "welcome" where the text also provides the hyperlink to the list of historic properties. This link is specified by the "onclick" annotation beside the name of the service framework.

Since the link specifies a database access, we must also specify the name of the database including its location, the name of the table, the names of the columns from which data is to be extracted, and the form of the link between this task and the succeeding one. In this case the link specified as "report" specifies that the next task receives the data from a database query. In this example, the database name is "Tours," the table name is "Buildings," and the column name(s) is "Address."



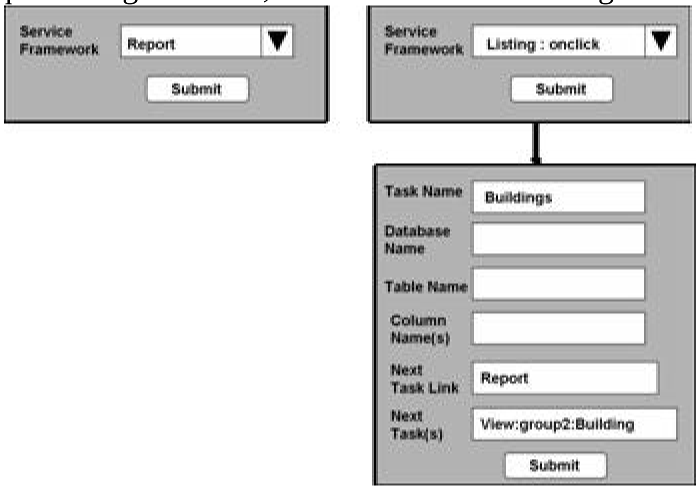
Service Framework	Report

Service Framework	Simple Text : onclick

Task Name	Welcome Text
Text	Welcome
Database Name	Tours
Table Name	Buildings
Column Name(s)	Address
Next Task Link	Report
Next Task(s)	View:group1:Properties

Figure 18
Form for creating a hyperlink text box task

The task to display a list of buildings is in Figure 19. Of course, this task is in a different stage, namely the viewing stage and in a different group, the description of which is similar to the one described earlier and therefore not repeated. The task for displaying a list of buildings is quite similar to the task in Figure 18 except that the service framework is of type “listing.” Note the modifier “onclick,” which indicates that the mouse can activate each entry in the listing. This type of framework requires a second step in defining the task, namely the description of the columns in the report. Thus pushing the submit button on the task produces a second set of forms for defining the name and types associated with each column in the report and providing a title for the column. The column form is repeated as many times as there are columns in the form. Since this step is quite straightforward, we do not show it in a Figure.



Service Framework	Report

Service Framework	Listing : onclick

Task Name	Buildings
Database Name	
Table Name	
Column Name(s)	
Next Task Link	Report
Next Task(s)	View:group2:Building

Figure 19

Steps in creating a report listing task

The final task in the application displays the description for a single building. It just contains pictures and text and is shown in Figure 20 and like other web pages will contain a table of contents. The style of the page is governed by the style accompanying the group enclosing this task.

Before completing this task, we must insert a map task into group2, the group that contains the description of the building. The form to produce a map task is in Figure 21. There are no links to a succeeding task as this is the last task in the lifecycle. Note that the Next Task Link in Figure 20 is “Submit,” which implies that the map is only activated when requested from the previous task. There are also check boxes to indicate which features are requested for the map.

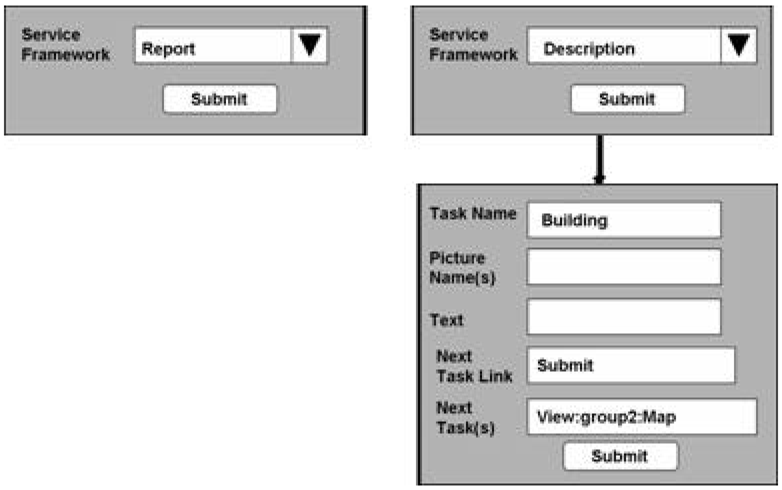


Figure 20

Steps in creating a description

Application 2 –creating tours of historic buildings

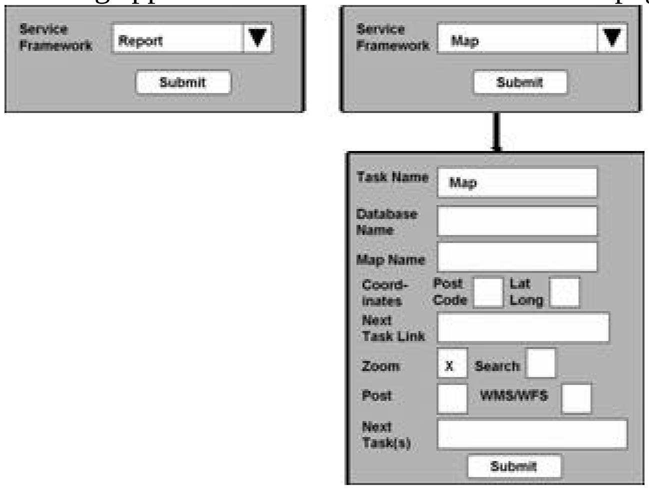
We now introduce a second concept called the tour. The tour groups buildings by districts so that the user can view all buildings in one area. The tour requires the addition of a second entity in the data model and two more tables in the corresponding database. The two tables are an entity table called "tours" which describes the tour and the related district and a relation table called building-tours that relates the tours or districts to groups of buildings. In the "tour" application, we introduce a new entity called the "tour," which has its own lifecycle, stages, groups and tasks. Since the Figures for these artifacts are very similar to the ones already presented, we only describe the differences in the text.

In the lifecycle, we introduce a "welcome" stage followed by a "view" stage similar to the "building" lifecycle. In the first or "welcome" stage, we introduce a welcome group where the "welcome" group contains a "welcome" task similar to the "welcome task" in the "building" group in Figure 18. The view stage contains a view group and a single task labeled "Tours." This task is connected to the preceding "welcome" task and then connects to the "Building" task shown in Figure 19. This connection illustrates a simple dependency between two tasks in different lifecycles.

If the system now uses the "welcome" task in the "tours" application as the starting point we can eliminate the welcome step from the "buildings" application since it is no longer reachable code. However, this does raise an interesting question: "What if we want both the "tour" and the building applications available?"

The answer to the question lies in introducing a new entity called "system;" this entity represents a collection of interrelated applications. The entity system has one value namely the URI for the entire system. As before, we introduce a lifecycle, a stage, a group and a task. In this case, the task is the welcome page already described. This approach allows us to capture a number of applications in a single "welcome" page. Thus, we create two dependencies in this application. We can also remove the "welcome" page from each application as this web code is now unreachable.

As we demonstrated, there are dependencies between services in this application and other applications within the information system. If other applications exist then these dependencies can be completed. The user can complete the information about the dependencies as the service is selected. If dependencies appear later, as the information system is developed, the applications can be revisited, since they are completely stored in the workflow database, and the dependency information can be completed. Such a possibility occurred in the application just completed, where a decision was made to make both tour and building applications accessible from the home page.


Service Framework: Report

Service Framework: Map

Task Name: Map

Zoom: X

Figure 21

Forms for a task for creating a map

The structure of the information system including lifecycles, stages and services, and their pattern of interconnection and dependency are stored in the workflow database. The workflow engine is now used to traverse the entire system structure and produce a set of interdependent operational applications. In the example just described the workflow engine compiles the application by filling in the links between the various forms and adds the style sheet to each form and connects the initial task of the application to a Web page.

Of course, the information systems may be incomplete or there may be bugs because not all dependencies have been defined. If there are bugs in the service frameworks, they are dealt with outside the context of the system development.

As can be seen from the previous description our approach clearly separates structure (workflow) and services applied to entities in the data model. Further, the structure is captured in a data structure rather than a programming structure, which makes the structure easier to identify and modify. If the services involve style, then that is easily separated from content as already been discussed.

Modification or maintenance of the workflow-based system is in two steps. If the structure of the system is to be changed, then the data structure representing the workflow is modified. If a service is to be changed then another service framework is substituted, the current one in use is modified, or values are changed in the corresponding adaptation interface. If a service framework is modified then it can be given a new identity within the framework repository. In this way, applications that use the old version of the service framework only use the modified version if desired.

The WIDE Toolkit and Service Frameworks

The Web Informatics Development Environment (WIDE) toolkit was developed to produce software systems from models based on enterprise service architectures. WIDE is primarily based on open source software technology and consists of a number of service frameworks. The frameworks are outlines of functionality for a specific service that are specialized using different approaches. For example if the frameworks are developed using object-oriented technology then they would be specialized using hotspots and similar concepts, although this information would still be captured using a declarative form-based adaptation interface.

Applications built using the WIDE toolkit can include input forms or reports containing extensive multimedia materials such as imaginative use of maps or any 2-dimensional diagram, databases, indexing and searching methods, agents, and push technologies. WIDE also contains a knowledge management system that supports documentation of technical information and best practices.

The adaptation interface for the service frameworks are expressed in a declarative XML-based DDSL. In the WIDE context, "programming" has effectively been replaced with a declarative methodology thus making it possible to provide a wizard or forms-based approach to building Web-based systems. Thus, WIDE has often eliminated the need for programming in the construction of complex Web-based systems.

Internally WIDE uses a bootstrap approach; its extensions are implemented using its own declarative technology. WIDE supports a rapid development paradigm and new applications can be built and demonstrated quickly. The structure of the WIDE environment also supports evolution thus addressing new concerns as they arise. The next few subsections contain descriptions of the service frameworks in WIDE.

A mapping services framework. This framework supports the creation of interactive maps which are delivered from a database-driven map-server. The interactive maps can provide zoom-in or zoom-out functionality and positioning over areas of interest. The mapping service framework does not use traditional geographic information system (GIS) software to support any of the mapping functionality. The mapping service framework minimizes complexity, support, cost, and time usually associated with creating and operating map-based applications.

The supported map services do support the new Canadian Geographic Data Initiative (CGDI) thus allowing exchange of information between the WIDE mapping services and most GISs. The mapping service can be used to:

- *provide multi-layered maps* when connected through the CGDI standard to a source of map layers such as a GIS;
- *display and interact* with the location of geo-referenced objects such as a building or park that is stored in a database;
- *find* geo-referenced data in databases or Web sites based on an area on the map defined by the user. The map area searches can be defined by a circle, rectangle or general polygon;
- *attach* an "electronic pushpin" and accompanying data to a map location or area and then store that data in a database or Web site.

The maps can represent any spatial or map-based concept including thematic maps such as ones showing environmental or demographic data, roadmaps or even floor plans. Combinations of maps can be displayed such as a road map with a superimposed thematic map. Vector-based maps can also be displayed where required.

The simple API (application program interface) to the maps provided by the server supports rapid development of map-based applications. The mapping functionality is based on the Scalable Vector Graphics (SVG) open W3C standard [24]. Data can be geo-referenced or located using various forms of geo-coding including Lat-long, UTM, address-range, or postal code.

The mapping framework also supports the OpenGIS® Web Map Service (WMS) [19] and Web Feature Service (WFS) [20] thus supporting two-way interchange of maps and features between instances of the mapping framework and operating geographic information systems that support the services.

A diagram and chart services framework. This framework manages and delivers specified interactive diagram and chart types upon request for presentation of data on the Web or in other formats. The diagram and chart services are also based on the Scalable Vector Graphics (SVG) open W3C standard and so provide similar functionality to the mapping service framework.

An XML-based metadata framework. The adaptation interfaces to databases, Web sites, agents, and applications including reports and input forms with maps and diagrams are described using XML. They are transformed into operating applications through the use of XSL "programs."

A report services framework. This framework supports the creation and management of interactive report and input form types and delivers them on request for presentation on the Web or in other formats. The user specifies the form type or service framework while creating an instance for an application. The instance is specified by completing an adaptation interface, which is part of the framework. The instance is stored with the application and is interpreted at runtime to produce the required report.

A content management services framework. This framework supports the management of text and multimedia information in a database where it can be viewed, searched, maintained and then published for use on the Web or in other formats.

A role-based access control service framework. Access to any content such as a database, Web site or other text and multimedia content can be provided with multi-level role-related access controls to determine who can read or change data. For example, each entry in a database can be accessed by the individual or role that "owns" the data and has the authority to make changes.

A Web and database searching service framework. This service framework contains an indexing agent and search engine that indexes known Web sites and databases and supports searching. The results from Web searches are categorized based on different search criteria such as the proximity of words in a phrase. The results of combined database and Web searches can be presented together. Temporal searches are also possible where the results of a search can be saved and the search re-executed at a later time. The results of the two searches can be compared to see if new results have appeared in the intervening time interval. The searching framework includes the ability to rank search results.

A push/notification service framework. The Web is naturally a pull medium. A user must look for information by pulling it out of the Web. The general push/notification service framework allows developers to create systems that allow users to specify conditions under which they wish to be notified or have information pushed at them. For example, someone who is buying a house could request notification if a house meeting his/her specifications becomes available.

An agent service framework. The agent service framework supports the description of agents that act autonomously to perform utility tasks within an application. Agents are often defined to manage redundancy. For example, agents could be defined to verify the content of "local" databases against authoritative sources or to allow a user to type information once while submitting the data to multiple databases or Web sites.

A knowledge management framework – the academy. The academy framework is used to support widespread dissemination of "documentation" and knowledge describing how applications can be built from the WIDE Toolkit. Documentation includes best practices as well as principles of proper design for the Web and the Internet. The academy uses the StudySpace [25] educational interface that allows individual users to keep and manage personal annotations related to the document being read.

A workflow framework. **–** The workflow framework and its related workflow engine support the construction of an application from the previously described service frameworks. The application is formed by composing workflow patterns involving services. The workflow engine then "walks" over the composed patterns and interacts with the application developer to complete the tasks associated with the workflow.

Recent and Current Portal Projects by Sector

Complex web-based information systems sometimes called portals have been created to evaluate different versions of the WIDE toolkit and its constituent service frameworks as they have evolved. These designs have created the evidence-base to indicate the robustness of the approach. These portals cover several different areas including urban and rural community information, economic development, health, environment, and cultural heritage.

All of the portals are still operational, but may not be accessible to external users at this time, either because they are still waiting final publication approval, are experimental, or use older versions of the WIDE toolkit that are no longer supported. However the URIs are given for portals that can be accessed. This section provides a list of many of the systems that have been developed.

Urban and Rural Community Information

There are several web-based systems that have been developed to support urban and rural community information. We describe them in the order in which they were developed.

The Waterloo Information Network (WIN) – This portal was developed in 1998-99 as a portal for the City of Waterloo. It had a complete business directory for the City, which was searchable by keyword, name of business and similar properties and also provided timed advertising of local events. Mapping, which was not based on a geographic information system (GIS), was used to provide the functionality described previously.

County of Oxford Online (COOL http://www.cooloxford.ca/) – The County of Oxford adopted an early version of the WIDE toolkit to develop their portal. The COOL portal has features similar to the WIN portal, but has used a different organization. The COOL portal chose to use a mapping system based on GIS and developed by the GeoGraphics Information Systems (GGIS) Group of the County of Oxford.

The Rural Switchboard (http://learningspace.uwaterloo.ca/rs) - The Rural Switchboard system contains information related to business, tourism, arts and culture in the rural communities in Waterloo, Perth and Oxford counties. The Rural Switchboard uses the mapping functionality described earlier.

Project NOW (http://www.newcomerswaterloo.ca/) – The Waterloo Public Library as the lead agency and our research team have been building a multi-lingual portal for newcomers to Waterloo Region, particularly immigrants under the title Project NOW. There are over 1400 web pages of information of interest to newcomers.

Waterloo Regional Arts Council (WRAC - http://learningspace.uwaterloo.ca/wrac) - The WRAC site provides a comprehensive overview of arts and culture in Waterloo Region. There are databases of all the individuals and organizations involved in the arts and all the venues that are available to artists for presentations, displays and similar activities. The portal contains profiles of some local artists with interactive multimedia presentations of their work, a database of press releases that can be automatically sent to subscribers who register and indicate their preferences, and a planning calendar. This calendar can be used to download to multiple local event calendars. The WRAC system incorporates some of the mapping functionality described earlier.

Social Planning Council of Kitchener Waterloo - The Quality of Life portal provides information from the Social Planning Council of Kitchener Waterloo. It contains a number of community based resources including the Blue Book, a detailed listing of all community resources in Kitchener Waterloo.

Volunteer Action Centre (VAC) of Kitchener Waterloo (http://volunteerkw.ca/) - The VAC portal provides individuals with an opportunity to discover and apply for various volunteer opportunities within the greater Kitchener-Waterloo community and to provide related support services.

Economic Development

Canada's Technology Triangle (CTT) – CTT is an economic development organization for the Region of Waterloo in Ontario Canada. The site uses the mapping functionality described earlier. The site contains a business directory of all its members and a directory of commercial real estate (buildings and land) and uses the mapping to create a geographically searchable database of available buildings and land.

Health

Rapid Visualization Environment for Health Data – This experimental project incorporates maps and databases that can be used to visualize and understand better the operation of the Ontario Health System. It is intended that this portal might be used as a management "dashboard" for operational assessment purposes.

A Web-Based Tool for Defining Bandwidth Requirements of PACS Networks - Hospital groups are often faced with the problem of transmitting digital images such as Xrays, MRI or CAT scans to expert teams of radiologists in different locations. Thus, it becomes necessary to calculate present and future loads to determine when there is a need to purchase additional network capacity. This portal incorporates databases of network capacity and a calculator that can be used to compute network traffic based on assumed network input.

'Point of View' Intelligent Middleware – Our research team worked with DataGlider, a Canadian company specializing in Enterprise Healthcare Portal technology. Using their own technology and parts of the WIDE toolkit, the two groups built a prototype health care portal to support the display of multiple medical reports (portlets).

Health Informatics (http://hi.uwaterloo.ca) – Our research team has produced a Web portal for the Waterloo Institute for Health Informatics Research (WIHIR) at the University of Waterloo that describes the institute and its members.

Health Informatics Competencies (http://learningspace.uwaterloo.ca/hi/) – A research program on assessing competencies for people interested in a career in health informatics has been underway for a number of years. Those interested in this type of career can access the Web portal to learn about the various types of positions available in applied health informatics. By completing a questionnaire an individual can be assessed on their preparation for a specific occupation in the field.

VON CareNet: VON Caregiver Resource Network – This experimental system sponsored by the Victorian Order of Nurses is conceived as an Internet-based Community of Practice support tool to assist informal caregivers in providing effective and sustainable care in the home setting and in achieving effective and efficient communications with professional personnel.

The Regional Physician Webspace Project (http://www.family-medicine.ca) - The Regional Physician Webspace project is intended to provide a virtual health community with a wide array of support services for physicians in the Kitchener-Waterloo-Cambridge (KWC) region. Among its purposes is to provide an information environment that links physicians to each other, to the University of Waterloo, to regional healthcare facilities, and to information resources.

Performance Indicators Monitoring System (PIMS) – PIMS involves gathering indicator data and work planning related to smoking cessation programs, sponsored by the Ontario Ministry of Health Promotion. Information about work planning and program indicators is gathered from the 54 separate units across Ontario. Each unit has their special characteristics, which make the system quite complex. There are currently over 150 different interfaces that must be constructed and managed.

Environment

COMWASH – The COMWASH experimental portal was created to understand how portal technology could be used to support environmental remediation in the developing world. Information about the quality of water in a district of the Republic of Malawi in Africa has been used for illustrative purposes. There is a database of all water sources and these are mapped using the mapping technology described earlier. Local citizens can monitor existing water sources and change values such as water bacterial content, and also add new water sources to the database.

The Muskoka Lakes Association (http://www.mla.on.ca/) – The Muskoka Lakes Association system presents information about the quality of water in the Muskoka Lakes, a tourist and vacation area in Ontario Canada. There is a database of all water testing sites and accompanying test data and these are mapped using the mapping technology described earlier. Local citizens can monitor existing water sources and change values such as water bacterial content, and also add new water sources to the database.

The Ontario Stewardship Tracking System - The Stewardship Tracking System supports inventories of ecosystem restoration projects and is being developed in partnership with the Ontario Ministry of Natural Resources, with the support of GeoConnections, the Oak Ridges Moraine Foundation and the Metcalfe Foundation and in collaboration with conservation authorities and non-government-organizations in natural heritage conservation.

Cultural Landscape

Mapset (www.mapset.com) - The Mapset portal contains a number of examples of portals based on the WIDE technology. One section illustrates the many buildings in the Galt section of Cambridge Ontario thus providing an overview of the cultural landscape of the community.

Related Work

There has been other work on generating implementation from models. The ideas behind this work are neatly captured in the book by Czarnecki and Eisnecker [28] in which they describe methods of generating programs through the definition of domain specific languages (DSLs). The work described restricts itself to a specific domain and only generates domain specific applications. The methods described do not produce an entire implementation from a model as we have described.

UML and xUML (executable UML) [29] have been described as producing an implementation directly from a model. However the translation only produces code outlines which must be completed by the programming team. Thus great discipline must be exercised to avoid architectural drift where the implementation gradually deviates from the model.

There has been other work on software factories [30] and software product lines [31] where object-oriented frameworks are created using design patterns and specialized to perform specific functions. These are still programming models that require a high degree of expertise to understand and modify the framework. In fact there has been substantial work on process languages to capture framework instantiation activities [32] so that they can be replicated at a later time to minimize the extensive learning curve required of the framework programmer.

There has been substantial research on developing web-based systems from models. OOHDM [36] and WebML [23, 33] are two examples. However they are strictly focused on the web and not the more general concept of enterprise service architectures as we have demonstrated in this paper.

Conclusions and Future Work

In this paper, we have outlined an approach to building and generating software systems from an enterprise service model and then shown how the approach can be used in a web context. The approach proceeds from a model based on workflow patterns to an operational application. There are no gaps in the derivation between the model and the application. Any changes to the application are first specified in the model and then carried over to the application through a well-defined process based on workflow. Thus, the problem of architectural drift between model and operating application can be avoided. In many cases, modifications to the model are straightforward and do not make the entire derivation process inefficient and therefore open to circumvention.

There are a number of directions to be explored in future research.

1. We need to exercise our approach on many more examples in different enterprise service-oriented domains to ensure that there are no serious gaps in our understanding of the process.
2. Each task in our approach could encompass a crosscutting concern such as logging or failure. We need to investigate how to handle these concerns in a manner similar to aspect-oriented programming. Preliminary work on aspects related to workflow has indicated a possible approach.
3. Although our approach describes the derivation of an application directly from the model, we still need to describe a deployment strategy. We believe that this strategy can be specified in a manner similar to the one described in this paper. Preliminary

results support this view. Deployment strategies have significant effect on non-functional requirements such as performance.

4. Model checking and other validation techniques are very important to ensure that the behaviour of a software system meets certain criteria. We believe that we can associate behaviour with each task and thus validate systems using the model with this associated behaviour. Work is continuing on this theme.
5. As service-oriented architectures evolve, there will be many situations where services will be obtained using technologies such as UDDI [37] and WSDL [38]. We will have to expand our model to handle these methods of obtaining services and to handle exceptions when services are not available. Preliminary investigation indicates that the model can be expanded to handle this method of accessing and using services.
6. Although the systems are highly interactive already, we have observed that a higher degree of interactivity is possible using Ajax [39] and should be incorporated into the approach.
7. There is now a recognized body of work on business patterns [5]. We should investigate finding a mapping between a representation of business patterns such as the Resource, Event Agent (REA) model and the workflow and service framework describe in this paper. Preliminary investigations look promising.

Acknowledgements

The research reported in this paper has evolved over a number of years and has been funded by many businesses and government agencies including: Bell Canada, Communications and Information Technology Ontario (CITO), Human Resources Development Canada (HRDC), Human Resources and Skills Development Canada (HRSDC), iAnywhere Solutions (Sybase), IBM, Natural Sciences and Engineering Research Council of Canada (NSERC), Rogers Cable, and Oracle. As well we wish to acknowledge our many partners who participated with us in developing web-based systems that have allowed us to test our approach.

References

1. Schmidt, D.C. “Guest Editor's Introduction: Model-Driven Engineering.” IEEE Computer Volume 39, Issue 2, pp 25- 31.
2. Balasubramanian, K.; Gokhale, A.; Karsai, G.; Sztipanovits, J.; Neema, S. “Developing Applications Using Model-Driven Design Environments.” IEEE Computer Volume 39, Issue 2, pp 33- 40.
3. Michael Jackson and Graham Twaddle “Business Process Implementation: Building Workflow Systems.” Addison-Wesley, 1997.
4. Gregor Kiczales et al, "Aspect-Oriented Programming," Proceedings European Conference on Object-Oriented Programming vol 1241, Springer-Verlag, Berlin, Heidelberg, and New York pages 220--242, 1997.
5. Hruby Pavel. Model-Driven Design Using Business Patterns Springer, 2006.
6. John Reynold’s Blog http://weblogs.java.net/blog/johnreynolds/archive/2005/04/process_driven_1.html (last accessed April 29, 2007).
7. Business Process Execution Language for Web Services version 1.1 http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ (last accessed April 29, 2007).

8. Rumbaugh, J., Jacobson, I., Booch, G., The Unified Modeling language Reference Manual Addison Wesley 2004.
9. James Carey, Brent Carlson Framework Process Patterns: Lessons Learned Developing Application Frameworks Addison Wesley 2002.
10. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley 1995.
11. Smith, H. and Fingar, P. "Business Process Management – the third wave" Meghan-Kiffer Press 2002, Tampa, Florida, USA. ISBN 0-929652-33-9.
12. C. Plesums. "Workflow in the world of BPM: Are They the Same?" from The Workflow Handbook 2005, Future Strategies Inc., 2005, ISBN 0-9703509-8-8.
13. Chen, P. P.-S. "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Database Systems, 1, Vol 1, pp 9-36.
14. XML Metadata Interchange (XMI), v2.1 http://www.omg.org/technology/documents/formal/xmi.htm (last accessed April 29, 2007).
15. Extensible Stylesheet Language (XSL) http://www.w3.org/Style/XSL/ (last accessed April 29, 2007).
16. P.S.C. Alencar, T. Oliveira, D.D. Cowan, D. Mulholland. "Towards Monitored Data Consistency and Business Processing Based on Declarative Software Agents, Software Engineering for Large-Scale Multi-Agent Systems – Research Issues and Practical Applications," Garcia, A., Lucena, C. et al. (Eds), Lecture Notes in Computer Science (LNCS), vol. 2603, pp. 267-284, Springer, 2003.
17. Boehm, Corrado, & Jacopini, Giuseppe (1966), "Flow Diagrams, Turing Machines, and Languages with only Two Formation Rules", Communications of the ACM 9(5): 366-371.
18. Ashcroft, E. & Manna, Z. The translation of go to programs to while programs, Information Processing 71, Proceedings of IFIP congress 71 (Amsterdam, North Holland, 1972) pp 250-255.
19. OpenGIS® Web Map Service http://www.opengeospatial.org/standards/wms (last accessed April 29, 2007).
20. OpenGIS® Web Feature Service http://www.opengeospatial.org/standards/wfs (last accessed April 29, 2007).
21. D. L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM, 15(12):1053–1058, 1972.
22. S. Robertson and J. Robertson. Mastering the Requirements Process. ACM Press books. Addison-Wesley, Harlow et.al., 1999.
23. M. Matera, A. Maurino, S. Ceri, P. Fraternali: "Model-Driven Design of Collaborative Web Applications".Software: Practice and Experience Volume 33, Issue 8, 2003, Pages: 701-732.
24. Scalable Vector Graphics http://www.w3.org/Graphics/SVG/ (last accessed April 29, 2007).
25. StudySpace http://learningspace.uwaterloo.ca/rs (last accessed April 29, 2007).
26. Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl (last accessed April 29, 2007).

27. Simple Object Access Protocol http://www.w3.org/TR/soap (last accessed April 29, 2007).
28. Czarnecki, K. & Eisenecker. "Generative Programming: Methods, Tools, and Applications." Addison-Wesley, 2000 http://www.swen.uwaterloo.ca/~kczarnec/ (last accessed April 29, 2007).
29. Chris Raistrick C., Francis, P., Wright, J., Carter, C., Wilkie, I., Model Driven Architecture with Executable UML Cambridge University Press March 2004.
30. Greenfield & Short. "Software Factories: Assembling Applications With Patterns, Models, Frameworks and Tools." Wiley, 2004 http://www.softwarefactories.com/ (last accessed April 29, 2007).
31. Clements, P., Northrop, L., Software Product Lines : Practices and Patterns Addison-Wesley Professional; 3Rev Ed edition (August 20, 2001).
32. Alencar, P.S.C., Oliveira, T., Cowan, D.,D., Filho, M., Lucena, C.J.P., Software Process Representation and Analysis of Framework Instantiation, IEEE Transactions on Software Engineering (IEEE TSE), vol. 30, no. 10, pp. 145-159, March 2004.
33. WebML Documentation at http://www.webml.org (last accessed April 29, 2007).
34. Martin Fowler. UML Distilled Third Edition. Addison Wesley 2004.
35. Jim Arlow, Ila Neustadt. UML 2 and the Unified Process Second Edition. Addison Wesley 2005.
36. Daniel Schwabe and Gustavo Rossi, "An Object Oriented Approach to Web-Based Application Design", Theory and Practice of Object Systems 4(4), 1998. Wiley and Sons, New York, ISSN 1074-3224).
37. UDDI - http://www.uddi.org/ (last accessed April 29, 2007).
38. WSDL - http://www.w3.org/TR/wsdl (last accessed April 29, 2007).
39. Ajax - http://www.adaptivepath.com/publications/essays/archives/000385.php (last accessed April 29, 2007).