



(22) Date de dépôt/Filing Date: 2000/06/02

(41) Mise à la disp. pub./Open to Public Insp.: 2001/12/02

(51) Cl.Int.⁷/Int.Cl.⁷ G06F 17/30

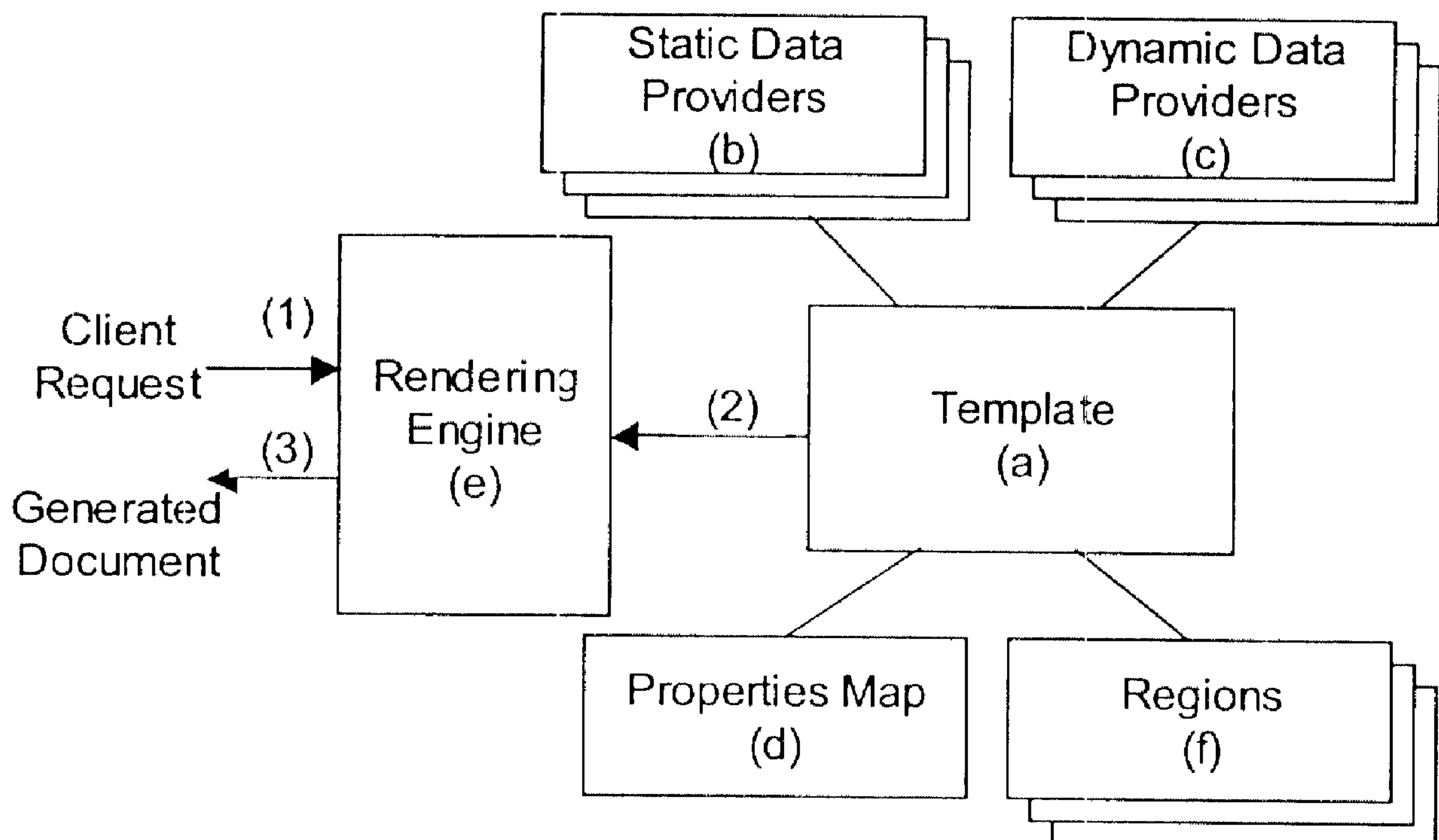
(71) Demandeur/Applicant:
SERVIDIUM INC., CA

(72) Inventeurs/Inventors:
SIKORSKY, MICHAEL J., CA;
SHAW, ROBERT, CA;
ZHANG, HONGWEN, CA;
BULMAN, JOE, CA;
RASMUSSEN, JONATHAN, CA

(74) Agent: SMART & BIGGAR

(54) Titre : METHODES, TECHNIQUES, LOGICIEL ET SYSTEMES PERMETTANT D'OBTENIR DES OUTILS
INDEPENDANTS DU CONTEXTE, PORTATIFS INDEPENDANTS DU PROTOCOLE OU REUTILISABLES DE
DEVELOPPEMENT

(54) Title: METHODS, TECHNIQUES, SOFTWARE AND SYSTEMS FOR PROVIDING CONTEXT INDEPENDENT,
PROTOCOL INDEPENDENT PORTABLE OR REUSABLE DEVELOPMENT TOOLS



Methods, Techniques, Software And Systems For Providing Context
Independent, Protocol Independent, Portable Or Reusable
Development Tools

5 Field of the Invention

The invention relates to Methods, techniques, software and systems for providing context independent, protocol independent, portable or reusable development tools. Different embodiments of the invention will each be described separately
10 below.

Polymorphic, Inheritance and Template Based Data Rendering
embodiment

This embodiment provides a technique, system, and computer program for using a polymorphic, inheritance-based
15 approach to render multiple sources of input data into a single output data source. This embodiment is hereby sometimes referred to as the "Template mechanism".

20 Background of this embodiment of the Invention

This embodiment relates to information presentation in a computer system, and deals more particularly with a technique, system, and computer program for applying a polymorphic, inheritance based approach to render multiple sources of input
25 data into a single output data source. This mechanism provides a clear methodology for the separation of presentation logic and disparate data sources.

One of the principles of computer science is to separate an application's presentation logic, business logic and
30 data sources. Accordingly, a need exists for a technique by which multiple loosely coupled input data sources can be rendered into a single output data source.

With this separation, the complexity of the system is reduced and its ability to adapt is increased, thereby reducing the effort for development, maintenance and future enhancements of the system.

5 Accordingly, it would be desirable to:

provide a technique whereby input data sources can be rendered into a single output data source;

provide a technique whereby this rendering technique allows efficient use of multiple and disparate data sources, such
10 as: plain text documents, HTML documents, and Software objects;

provide the required rendering technique in a way that enables different data retrieval or formatting implementations to be quickly and easily substituted into the computing
15 environment, to quickly adapt to changes in data or formatting requirements;

provide a technique that enables additional sources of data to be quickly and easily added into the rendering technique, providing greater flexibility in the manner in which the
20 data sources can be rendered;

provide a unique method of separation of presentation logic from data sources;

allow the separation of dynamic content generation (Dynamic Object Providers) from static content (Template and Static
25 Object Providers) providing a useful separation of disciplines;

provide a polymorphic content replacement algorithm providing a paradigm for selective replacement of Data Providers;

provide an inheritance based content replacement algorithm by
allowing Data Providers to hierarchically add regions
within regions; and

provide a value replacement algorithm for use across multiple
5 Templates.

Summary of this embodiment of the invention

An aspect of this embodiment provides a software
implemented process for use in a computing environment that may
10 or may not have a connection to a network comprising: one, a
client request; two, one or more data sources, each comprising:
a subprocess for receiving a client request; a subprocess for
parsing the data; a subprocess for marking up named regions in
the data; a subprocess for transforming the data with named
15 regions into a format expected by the rendering subprocess; a
subprocess for determining named placeholders in the data; and
three, a data source determined by the client request that
drives the rendering subprocess; and a subprocess for returning
the newly rendered output data as a response to the client
20 request.

This embodiment will now be described with references
to Figure 1, in which like reference letters denote the same
component throughout, whereas like reference numbers denote the
same process step throughout.

25

Detailed Description of this embodiment

Component Description:

Figure 1. illustrates the high level components which
comprise the Template mechanism. The following is a description
30 of these components beside the indicated component letter:

- a) Template - The Template provides the detail on how the various input data sources are to be merged. The ordering of the linking of Data Providers to the Template determines the priority of which providers are invoked for a given Region (f).
- b) Static Data Provider - The Static Data Provider provides a mechanism by which a data source can be described statically, in the form of a document for example. Static Data Providers may also introduce new Regions producing a hierarchical nesting of regions. These inherited regions are also resolved by the Rendering Engine (e).
- c) Dynamic Data Provider - The Dynamic Data Provider provides a mechanism by which a data stream can be constructed for each client request. This is often accomplished, but not limited to, using a programmatic language to retrieve the data.
- d) Properties Map - The Properties Map provides a global list of values for the Template mechanism, centralizing commonly used and shared values between multiple Templates.
- e) Rendering Engine - It is the duty of the Rendering Engine to merge the various data sources into a single document based on the rules indicated by the Template (a). Depending on the ordering of the Data Providers added to a given Template, the rendering engine will call on the providers in last in, first out (LIFO) order. This allows providers to be added which may override the regions previously provided for, giving an inheritance behavior. The polymorphic behavior is realized by the ability to have regions that provide content dependant on the context of the request.

- f) Region - A Region represents a unit of content that may be provided by a Data Provider. A Template may consist of several regions, each of which may provide content.

5 Operational Description

Further to Figure 1. the following steps occur in the operation of the Template mechanism:

- 1) The Client Request is received by the Rendering Engine.
- 10 2) The rendering engine locates the appropriate Template, linked Data Providers and associated Properties Map.
- 3) The Rendering Engine traverses the regions contained in the Template and queries the data Providers in LIFO ordering to determine which provides content for that
15 region. Any new regions introduced by Static Data Providers (b) are then resolved in a recursive manner. Once all regions have been satisfied, the content is then merged into the template to render the completed document.

20 Configuration Description

The following is a description of the configuration of the Template mechanism:

1. Construct a template document: Figure 2 illustrates an outline of how the final document is to be structured,
25 including title, modified date, body and salutation.
2. Figure 3 illustrates a mark-up the template's document content to specify its regions and placeholders: The purpose of this step (which could be implemented to be accomplished

manually or automatically) is to identify all of the content regions of the template, allowing the Rendering Engine (e) to merge the information from associated data providers into the completed document. Notice that all of the regions share the same mark-up schema; there is no differentiation between which regions will include static or dynamic content regions.

3. Construct and mark-up the template providers:

10 i. Static Data Provider, in this case a static document: As shown in Figure 4 we see that two regions are being provided by this static data provider, the Title and the document's contents. Additionally, the Static Data Provider is in the form of a document; hence, we'll refer to it in this example as a Document Data Provider.

15 ii. Dynamic Data Provider: In this example, the data modified region is being provided by a software program. For the sake of simplicity, as shown in Figure 5 this program simply returns a date.

20 4. Construct a properties map to define placeholder values: As shown in Figure 6, We've decided to add the name using a properties map. Properties maps are useful for itemizing commonly used, but relatively static information used across several Templates.

5. Figure 7 illustrates the linking of the providers and the properties map to the template:

6. Invoke the rendering process on the template to produce the output: As shown in Figure 8, the rendering engine will call on each of the object providers and properties maps to provide content for the template. Once the content is obtained, the engine merges it into the indicated regions, producing the final document.

The rendered document contains content obtained from the Static and Dynamic Data Providers, as well as the Properties Map.

Protocol Independent Service Handler Mechanism embodiment

The Service Handler (SH) mechanism embodiment provides a technique, system and computer program which abstracts the specific services of an application from the network communication protocols used by using a unique event handling mechanism.

Background of this embodiment:

Traditionally, systems have mapped services such as email, web applications, and cell phone applications, for example, to their unique protocol on a one-to-one basis. This means that for N services using M protocols, there are on the order of $M \times N$ components required. This creates a complex system which is both difficult to build and costly to operate. Also, when shared services such as security, logging and metrics collection are required by the system, their integration becomes equally as complex and difficult to manage.

Accordingly, it would be desirable to:

- reduce the complexity of the design and implementation of systems which provide multiple services for multiple network protocols;
- 5 provide abstraction of protocol specific implementations away from the services provided by an application in a unique and powerful paradigm;
- allow greater flexibility in system configuration and operation due to an ability to treat all requests as events in
10 the Service Context;
- reduce the number of components required to build a multi-protocol, multi-service application, and reduce the time required for new application development;
- provide the ability to manage the system centrally at the broker
15 thereby allowing holistic security, metrics, and auditing capabilities to be implemented in what are traditionally thought of as heterogeneous systems;
- allowing rapid integration of aging applications into new systems using new protocols; and
- 20 allowing legacy protocols to utilize functionality of new services with minimum effort.

Description of this embodiment

Figure 9 illustrates the software components and their
25 relationships in the architecture of an aspect of the Service Handler mechanism embodiment. The main components of the system are the Protocol Proxy, Broker, Service Handler and Service Context. Arrows between components indicate a "uses" relationship.

- a. The Protocol Proxy provides the protocol abstraction layer for the SH mechanism. When a request arrives at the Proxy, the request is decomposed into its atomic elements and a Service Context (b) is created. When the response is generated by the specific Service Handler (d), the Proxy is responsible for transforming the response back to the original protocol. Generally in a system, there is one Protocol Proxy per protocol type.
- b. The Service Context enables the loose coupling required between the SH mechanism components, as well as additional state information required by stateless protocols. The Context encapsulates the atomic request as events, and also the resulting response elements returned by the Service Handler (d). The request elements are initially added by the Protocol Proxy (a) and the response elements are generated by the Service Handler. The lifetime of a Context is protocol specific and is determined by the specific Protocol Proxy for that protocol.
- c. The Broker is the central controller for the SH mechanism. The Broker's main responsibilities are to provide a central registry for Service Handlers (d), to query Service Handlers for a volunteer to handle the current request, and to call on the Service Handler's authentication mechanism (not described) to ensure the request is from an authenticated source.
- d. The Service Handler is the worker of the SH mechanism. The Service Handler manipulates the request parameters in the Service Context (b) and, based on the type of Service, performs an action and/or returns a set of response elements in the Service Context. The response elements, if required, may include stream based content.

Details, as depicted in Figure 10:

1. A request is received by the Protocol Proxy.
Protocols may include, but are not limited to: HTTP, Jini,
5 FTP, DCOM, MAPI etc.
2. The Proxy determines if this request belongs to an
existing Service Context, or if one does not exist, a new
Service Context is created.
3. The Proxy breaks the request down into its atomic
10 elements and stores them to the Service Context as an event.
4. The Proxy passes the Service Context on to the Broker.
5. The Broker queries each of its registered Service
Handlers, in an ordered fashion, to find a volunteer to handle
this event type. The Service Handlers may determine whether
15 or not to accept a request based on a multitude of parameters
in the Service Context, which may include, but are not limited
to: protocol type, request parameter value(s), time of day,
user/role info, number of requests etc.
6. Once a Service Handler accepts the request, the Broker
20 requests the Service's authentication mechanism (not
described) to authenticate the request.
7. Assuming the request is authenticated, the Broker
calls on the Service Handler to handle the request.
8. The Service Handler may perform a specific action and
25 optionally return request parameters or streamed data via the
Service Context response mechanism. Examples of Service
Handler types may include, but are not limited to: mail
services, directory services, legacy services via SH wrappers,
content rendering services etc.

9. The response parameters are passed back to the Proxy for translation into their native protocol.

10. The response is returned to the requestor.

The Page Routine mechanism embodiment

5 In a web based application, a sequence of pages may be used to accomplish a task, e.g. a sequence of pages that allow a user to register for an online service. In many situations, the same task may be invoked from several different places. An advantage of an aspect of the *Page Routine* mechanism embodiment
10 is to make the sequence of pages, including the server side software code, for a certain task reusable.

In order to accurately describe an aspect of this embodiment, we give the following definitions:

- Page: A web page that can be viewed in a browser.
- 15 • Page instance: The actual viewing of a page in a browser.
- Page state: the complete state information of a page instance.
- Task: a Task is defined as a set of pages, including the back end server code, that fulfill a certain function.
- 20 • Task instance: The actual execution of a task.
- parent page: the page that has a button to a reusable task.

Figure 11 illustrates the particular advantages of the *Page Routine* approach. The same task, *Task X*, is invoked by both *Page A* and *Page B*. If the task is invoked from *Page A*, then the
25 termination of the task will cause the browser to display *Page A*. All the input fields of *Page A* will be populated with the values that are prior to the invocation, i.e. if a field has a

value ABC, upon returning from the task, the same value ABC should still be there. The task can also be invoked by *Page B* in the same way.

Figure 12 shows how the *Page Routine* mechanism can be implemented in a Java Servlet environment, such as Jaydoh. Each page instance is assigned a unique id. The id is displayed as a hidden field on the page instance. A task is invoked by the invocation of the server module *Task Invoke* which stores the parent page's state in a HTTP session with the parent page instance's unique id as the key. Each page instance of the task instance will carry, as a hidden field, the parent page's id. Hence, when the task instance is cancelled or returned on a page instance, the parent id is used by the server module *Cancel-To-Parent* or *Return-to-Parent* to retrieve the page name of the parent page instance. The control is then redirected to a new instance of the parent page. If the redirection is caused by the returning from the task, then the result of the task will be stored in the session with the parent id as the key. The new instance of the parent page will be displayed with the following steps:

1. Retrieve the page state from the session.
2. For each field on the page, if page state and field state is not null, if there is no task result associated with the field, then display the field state else display the task result.
3. Clean out the page state from the session.

Examples:

1. An aspect of this embodiment provides for the reusing of a sequence of web pages along with the server side code from many different pages and returns the browser to the

invoking page when the task is done. The software remembers the name and the state of the invoking page and will send the invoking page to the browser with the state of the page prior to the invocation upon finishing the invocation.

5 2. Another aspect of this embodiment provides a servlet based system that implements the previous example. The system, as depicted in Figure 2, assigns a unique identifier to each page and saves the state of the invoking page into a HTTP session with the unique identifier as the
10 key. Each page in the invoked task holds the parent page's unique identifier as a hidden field. Hence, when the task terminates, the parent page's state can be retrieved from the session by the system. The system then sends the browser the content of the parent page with the retrieved
15 state.

EJB abstraction embodiment

An aspect of this embodiment describes how developers may be shielded from the nuances of various Enterprise Java Bean (EJB) containers.

20

Background of this embodiment of the invention

The widespread adoption of Java as an enterprise application development language has lead to a large offering of commercial products from vendors. The market place is filled
25 with various tools and products all vying for market share. Because of the immaturity of the Java technology and Enterprise Java Bean specification, many vendors offer proprietary solutions when offering tools to developers. These proprietary services aid the developer at the expense of locking the
30 developer's application to that vendor's tool. This is counter to the open spirit of Java where vendor lockin is undesirable.

An EJB container is the runtime environment that holds a developer's EJB. With the proliferation of commercial EJB containers in the market place, the chances of developers at some point migrating from one vendor container to another is 5 very high.

As the EJB container market matures, we can expect more consolidation in this market as large vendors take over larger portions of the market.

Migrating EJBs from one container to another can be 10 difficult. There are often subtle differences in implementation. In the best of circumstances, considerable code rewriting will be required to modify the EJB and its respective application in such a way that it is able to make use of another EJB container's feature set.

15 Further, many applications developed today give little thought regarding how to best implement and architect their design in anticipation of moving the application from one EJB container to another. Hence EJB applications are developed without much thought on how to avoid vendor lockin.

20 Vendor lockin manifests itself in software by having EJBs developed that only run within one vendor's runtime environment. Hence the application is totally dependent on that vendor's EJB implementation. This practice is not advocated and should be avoided. Vendor lockin ties an application with a 25 particular vendor making the application extremely vulnerable to the state of the vendor. For instance, if the vendor decided to no longer support certain features with their EJB container, the developers would be forced to use older unsupported versions of the tool. If the application needs new features offered by a 30 newer version of the container, then considerable effort is required to rewrite the application in such away that it is no longer dependent on the proprietary older features.

In view of, among other things, the considerable effort currently involved with moving an application from one EJB container to another, if it is possible at all, abstracting away the difference between various EJB containers is desirable for the longevity of EJB applications.

Accordingly, it would be desirable to abstract away the various differences between EJB container implementations, to enable developers to more easily port their EJBs from one container to another, and for the longevity of EJB applications. It would also be desirable, from the point of view of developers that their applications not be locked into one vendor's EJB container.

In other words, it would also be desirable, from the point of view of developers, to be able to migrate their legacy EJBs from one EJB container to another.

Summary of this embodiment of Invention

As noted above, from the point of view of a developer, it is desirable to ease the migration of applications from one application server to another by abstracting away their differences. By isolating the nuances and differences between various EJB containers, applications can be deployed and run on a variety of EJB containers with no impact on developer code. Figure 13 illustrates this concept.

By inserting a layer of indirection between the application and the EJB container, developers protect their applications from the different EJB container implementation. If an application needs to be migrated from one EJB container to another, none of the application business logic would have to change - only the property that defines which EJB container the

application is running against would have to change. Figure 14 illustrates this concept at the component level.

Applying preferred software design practices, the invention is implemented by using the Stage/Strategy design pattern. An application requiring the services of an EJB container may instantiate the specific vendor container required as an EJB Implementation interface. This interface can now be used throughout the application without any regard from which EJB container the concrete class actually represents. By using Java polymorphism, one object, EJB Implementation, can take many forms.

If at some time in the future the developer needs to utilize the runtime environment of another EJB container, then the application simply instantiates that EJB container's concrete class and as the EJB Implementation interface.

At the implementation level, developers would implement this abstraction as shown in the following sequence diagram figure 15.

Each concrete EJB container implementation would hide the details of how that EJB container implements its runtime environment. For instance, the JNDI package name, how the EJBs reference database resources, security details, and the references to others EJBs can all be implemented by the various EJB container vendors in many ways. The layer of indirection (EJB Implementation) could be realized as either an interface or abstract class. An abstract class would be desirable if there were some features that were common amongst all EJB containers. Those implementations could then be implemented in the EJB Implementation abstract class. If there is no commonality between the EJB Implementations, then an interface would offer more flexibility for polymorphism. The recommended implementation would be for an abstract call as the 'is a'

relationship between an EJB Implementation and the Concrete implementation is quite strong. This suggests inheritance via an abstract class.

Once the EJB Implementation has been decided upon, the 5 concrete classes can be realized for each respective EJB container. Each container would hide the details of how that container realizes its service or operation. By hiding the details of each concrete implementation behind the EJB Implementation, developers may realize another EJB vendor 10 container when needed and deploy their EJBs to the new container once its concrete implementation has been realized.

What we claim as our invention is:

1. A technique, system, and computer program for using a polymorphic, inheritance based approach to render multiple
5 sources of input data into a single output data source.
2. A software implemented process for use in a computing environment that may or may not have a connection to a network comprising: one, a client request; two, one or more data
10 sources, each comprising: a subprocess for receiving a client request; a subprocess for parsing the data; a subprocess for marking up named regions in the data; a subprocess for transforming the data with named regions into a format expected by the rendering subprocess; a subprocess for determining named
15 placeholders in the data; and three, a data source determined by the client request that drives the rendering subprocess; and a subprocess for returning the newly rendered output data as a response to the client request.
- 20 3. A technique by which multiple loosely coupled input data sources can be rendered into a single output data source.
4. A technique where input data sources can be rendered into a single output data source.
- 25 5. A technique allowing efficient use of multiple and disparate data sources, such as: plain text documents, HTML documents, and Software objects.

6. A rendering technique to enable different data retrieval or formatting implementations to be quickly and easily substituted into a computing environment, to quickly adapt to 5 changes in data or formatting requirements.
7. A technique to enable additional sources of data to be quickly and easily added into a rendering technique, providing greater flexibility in the manner in which the data sources can 10 be rendered.
8. A method of separation of presentation logic from data sources.
- 15 9. A technique to allow the separation of dynamic content generation (Dynamic Object Providers) from static content (Template and Static Object Providers) providing a useful separation of disciplines.
- 20 10. A polymorphic content replacement technique for selective replacement of Data Providers.
11. An inheritance based content replacement technique allowing Data Providers to hierarchically add regions within 25 regions.

12. A value replacement technique for use across multiple Templates.

13. A technique for reducing the complexity of the design
5 and implementation of systems which provide multiple services
for multiple network protocols.

14. A technique for providing abstraction of protocol
specific implementations away from the services.

10

15. A technique for providing abstraction of protocol
specific implementations away from the services and allowing
greater flexibility in system configuration and operation
comprising an ability to treat all requests as events in a
15 Service Context.

16. A technique for reducing the number of components
required to build a multi-protocol, multi-service application,
and drastically reduce the time required for new application
20 development comprising a technique for providing abstraction of
protocol specific implementations away from the services and
allowing greater flexibility in system configuration and
operation comprising an ability to treat all requests as events
in a Service Context.

25

17. A technique for reducing the number of components
required to build a multi-protocol, multi-service application,
and drastically reduce the time required for new application
development comprising a technique for providing an ability to

manage the system centrally at the broker thereby allowing holistic security, metrics, and auditing capabilities to be implemented in what are traditionally thought of as heterogeneous systems.

5

18. A technique for allowing rapid integration of aging applications into new systems using new protocols.

19. A technique for allowing legacy protocols to utilize
10 functionality of new services with minimum effort.

20. A technique for reusing of a sequence of web pages along with the server side code from many different pages and returning the browser to the invoking page when the task is done
15 comprising a method for remembering the name and the state of the invoking page and sending the invoking page to the browser with the state of the page prior to the invocation upon finishing the invocation.

20 21. A servlet based system comprising: assigning a unique identifier to each page and saving the state of the invoking page into a HTTP session with the unique identifier as the key; each page in the invoked task holding the parent page's unique identifier as a hidden field; wherein when the task terminates,
25 the parent page's state can be retrieved from the session by the system; the system then sending the browser the content of the parent page with the retrieved state.

22. A technique for abstracting away the various differences between EJB container implementations, to enable developers to more easily port their EJBs from one container to another.

5

23. A technique to facilitate migration of legacy EJBs from one EJB container to another comprising isolating nuances and differences between various EJB containers, thereby allowing applications can be deployed and run on a variety of EJB
10 containers with no impact on developer code.

Smart & Biggar
Ottawa, Canada
Patent Agents

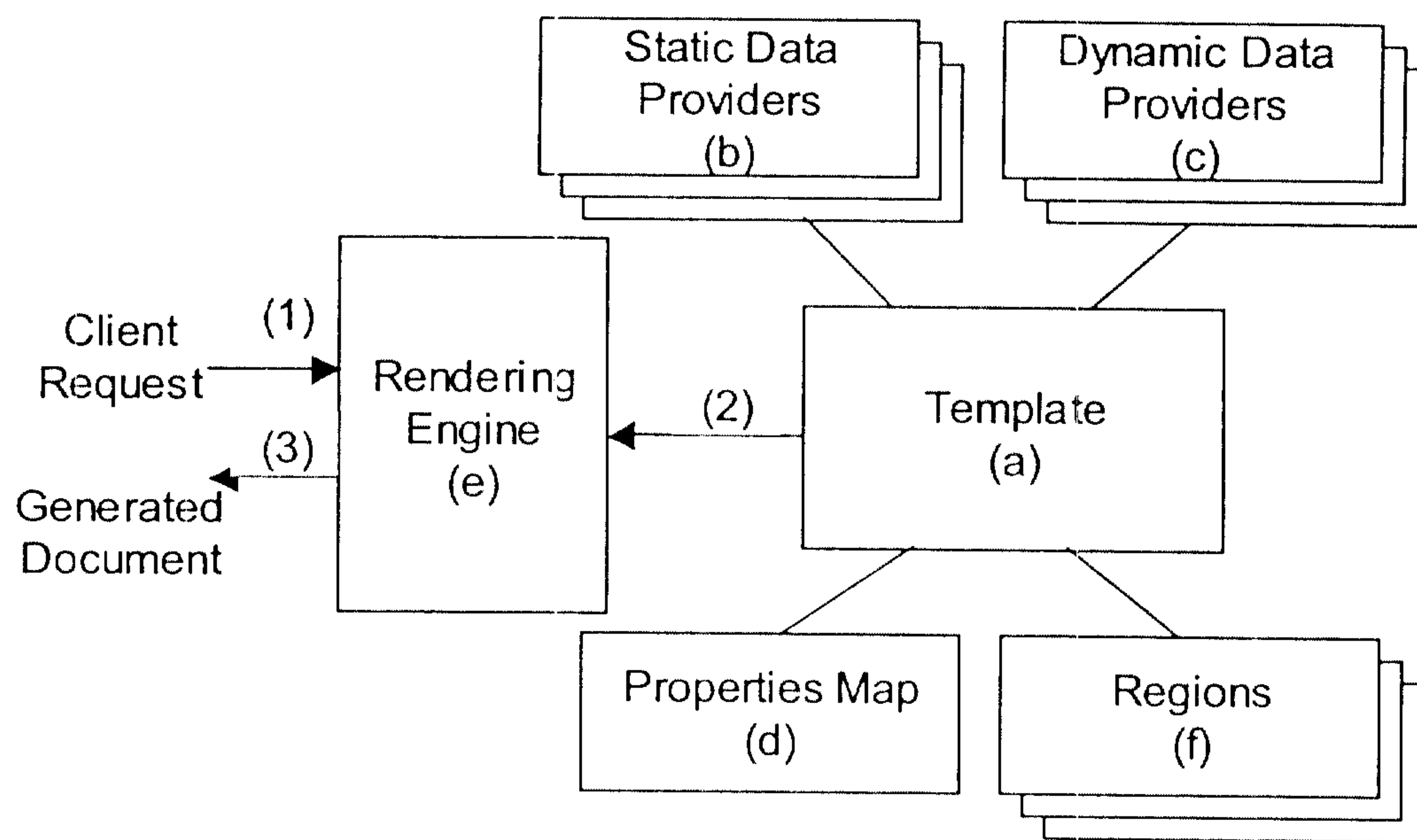


Figure 1: The Template Components

<div>Document Title</div> <div>This document was last modified on: some date</div> <div>add contents of document here</div> <div>Thank you, some name, for viewing this document</div>
--

Figure 2

Figure 3

```
<region name="DocumentTitle">Document Title</region>

This document was last modified on: <region name="DateModified">some
date</region>

<region name="DocumentContents">some contents</region>

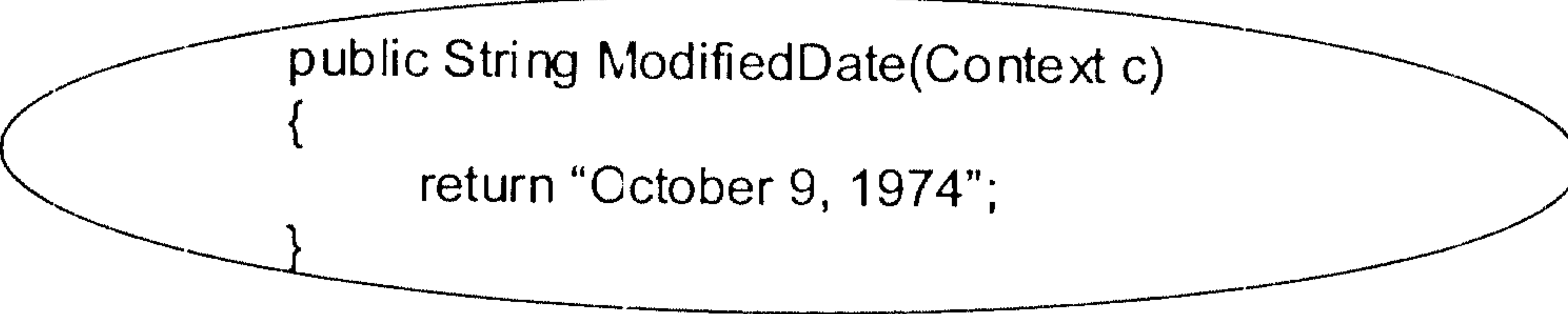
Thank you, $user.name$, for viewing this document
```


Figure 4

```
<region name="DocumentTitle">RKO 281: Deconstructing Citizen Kane</region>

<region name="DocumentContents">
This document deconstructs Well's Citizen Kane...
This is just another sentence from the document provider...
</region>
```

Figure 5



```
public String ModifiedDate(Context c)
{
    return "October 9, 1974";
}
```

Figure 6 of this embodiment:

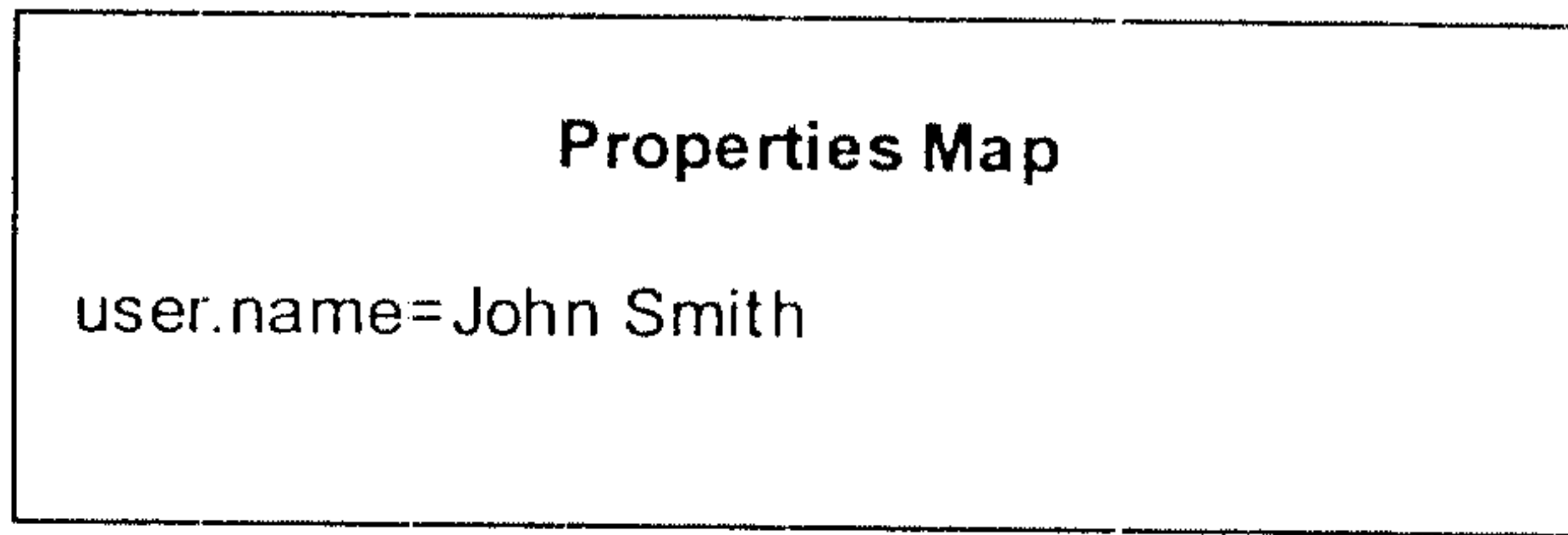


Figure 7 of this embodiment:

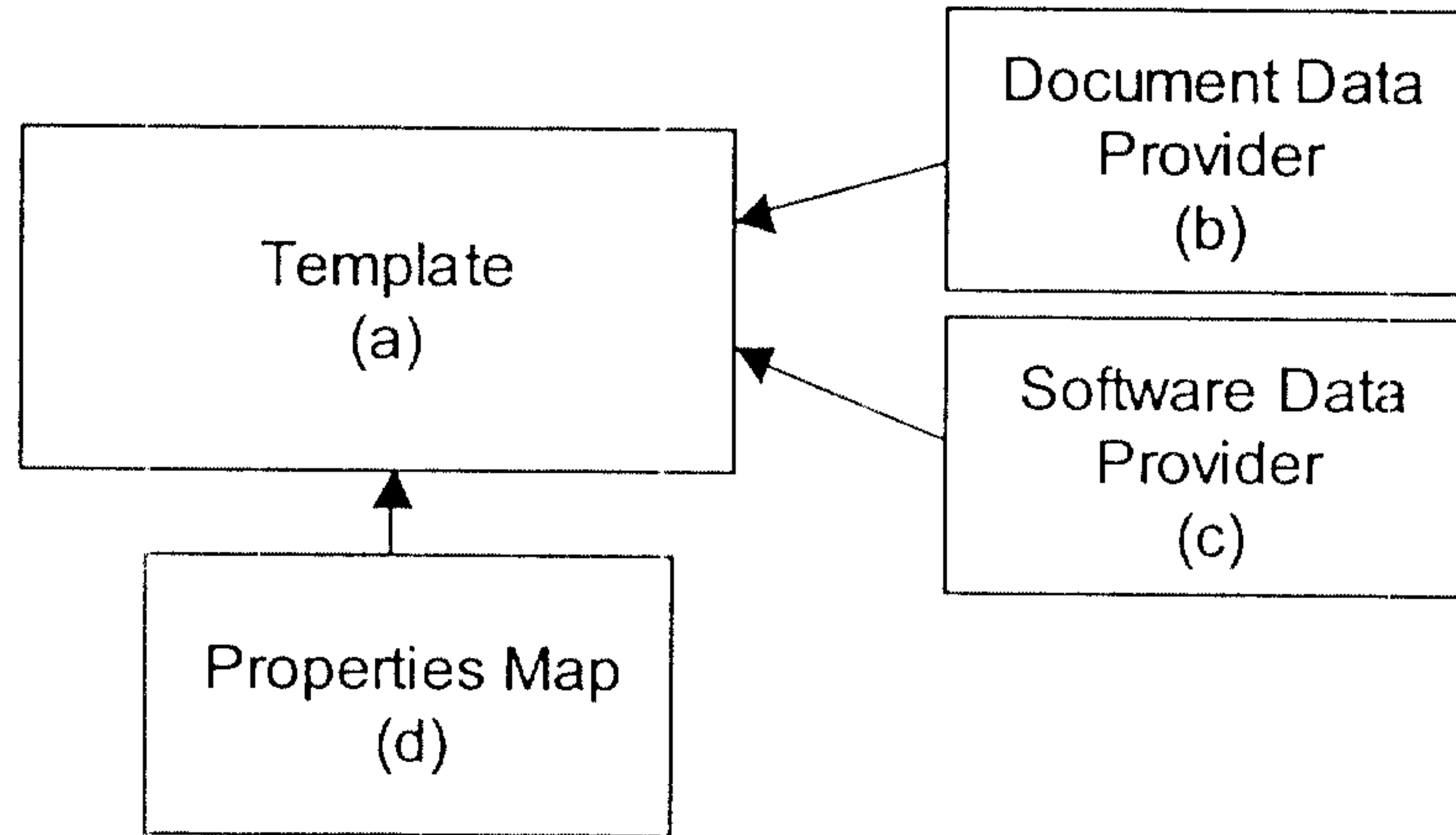


Figure 8 of this embodiment:

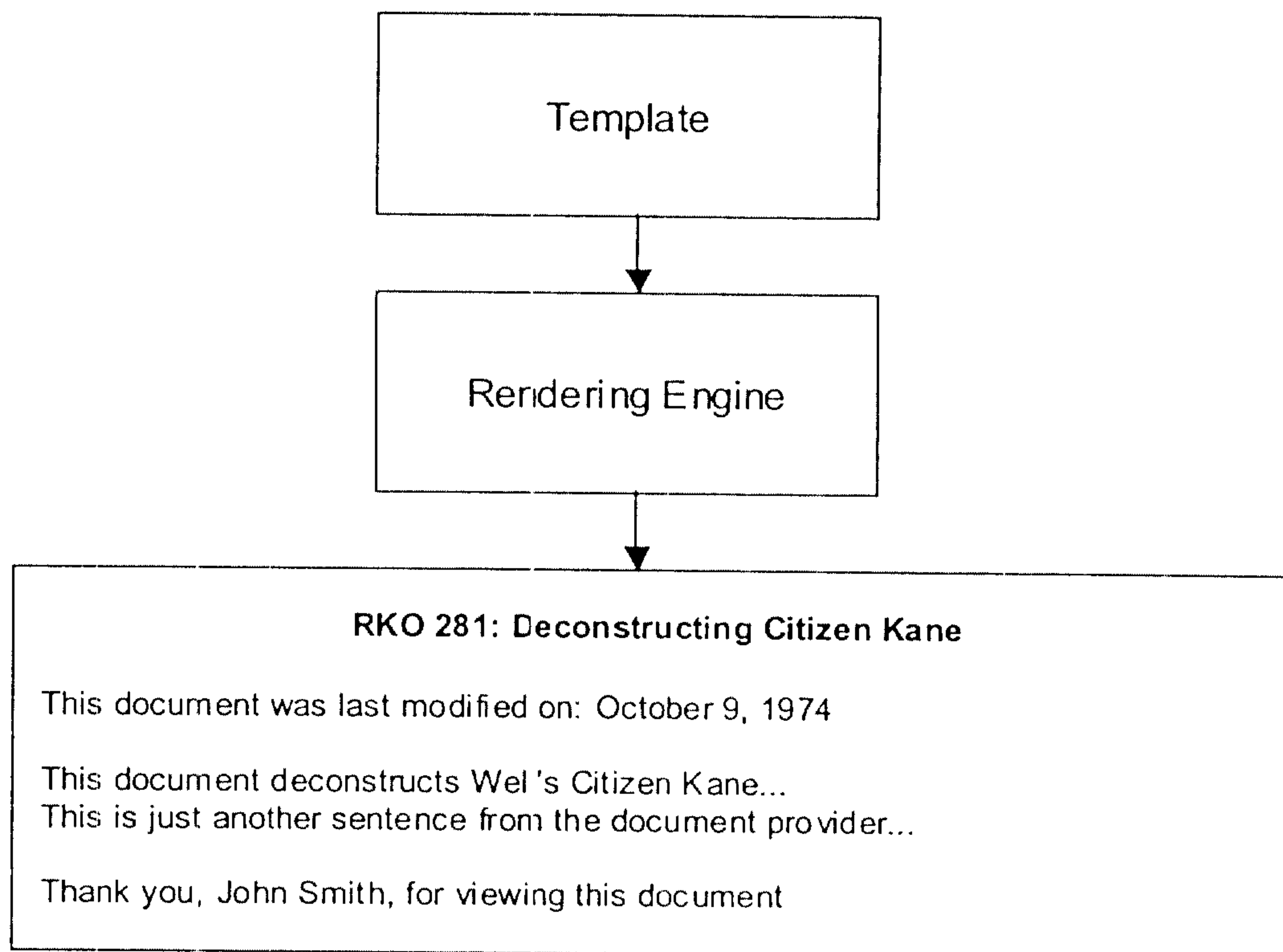


Figure 9. Basic Architecture

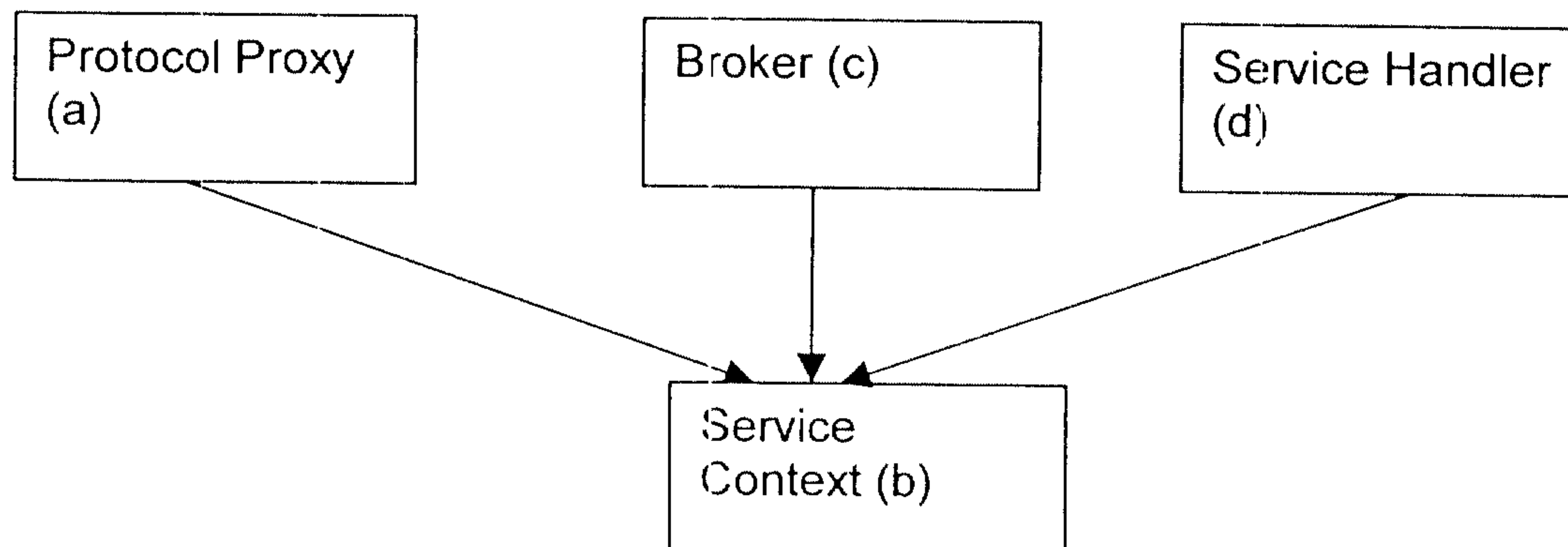
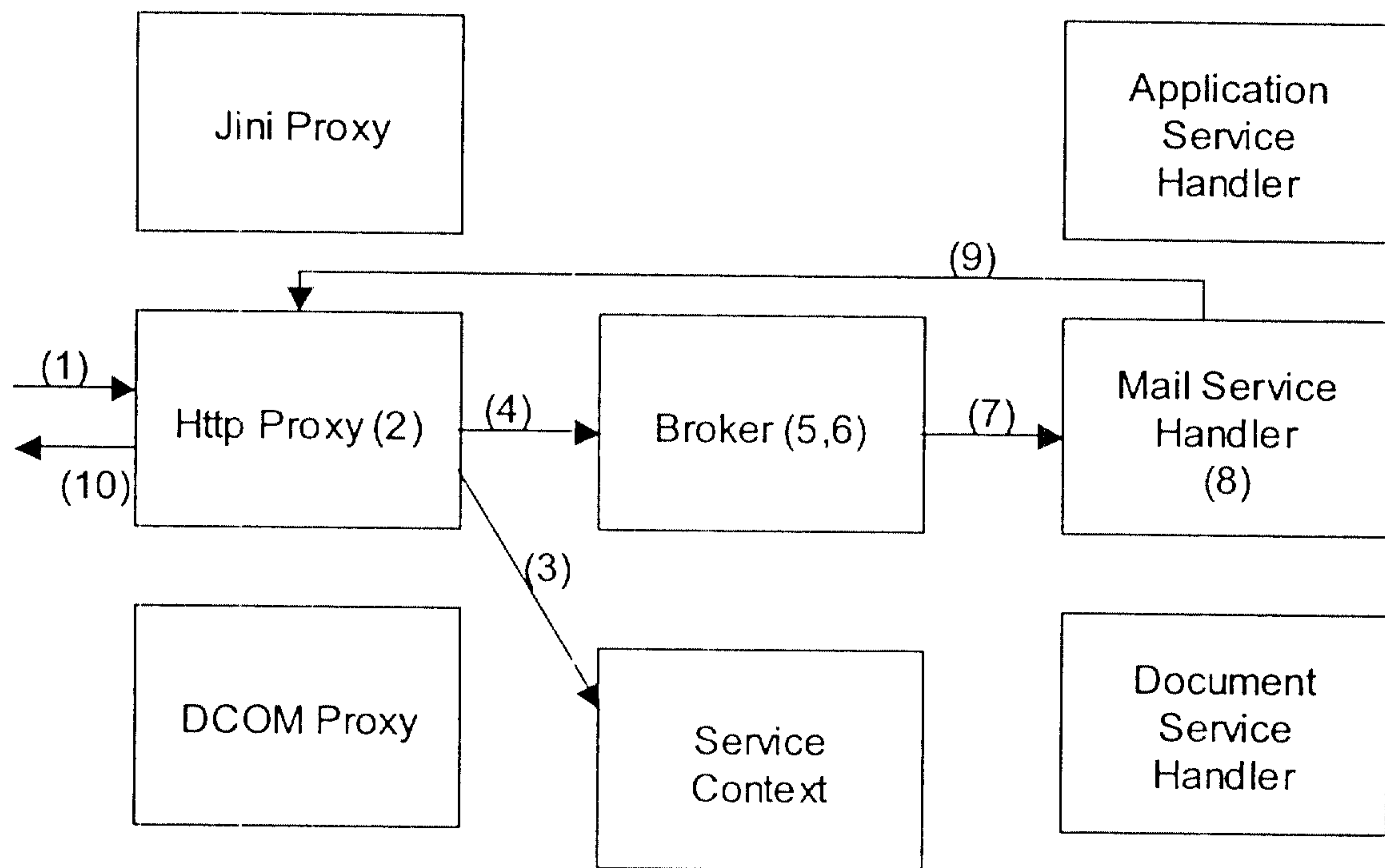


Figure 10: Example Scenario



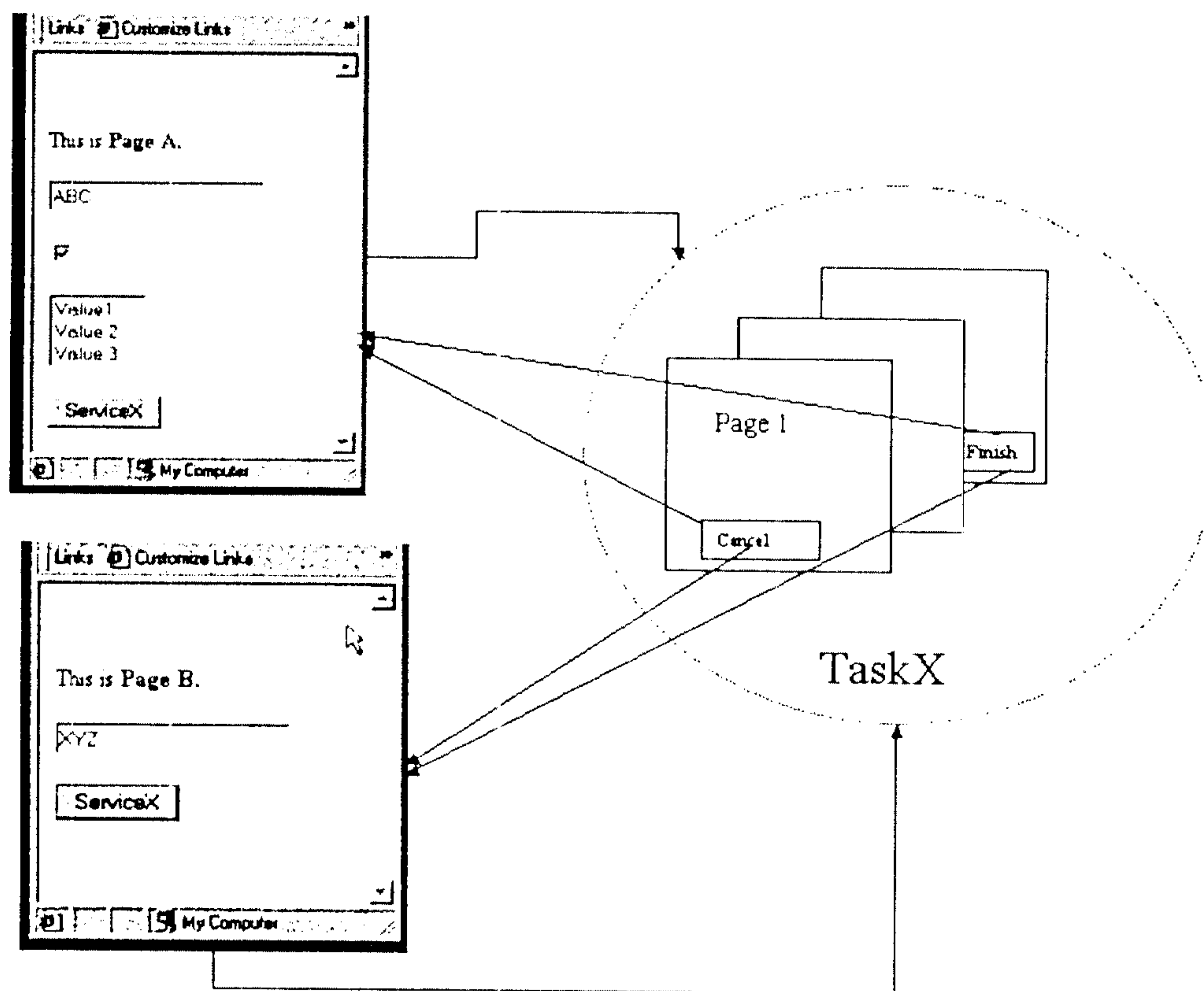


Figure 11

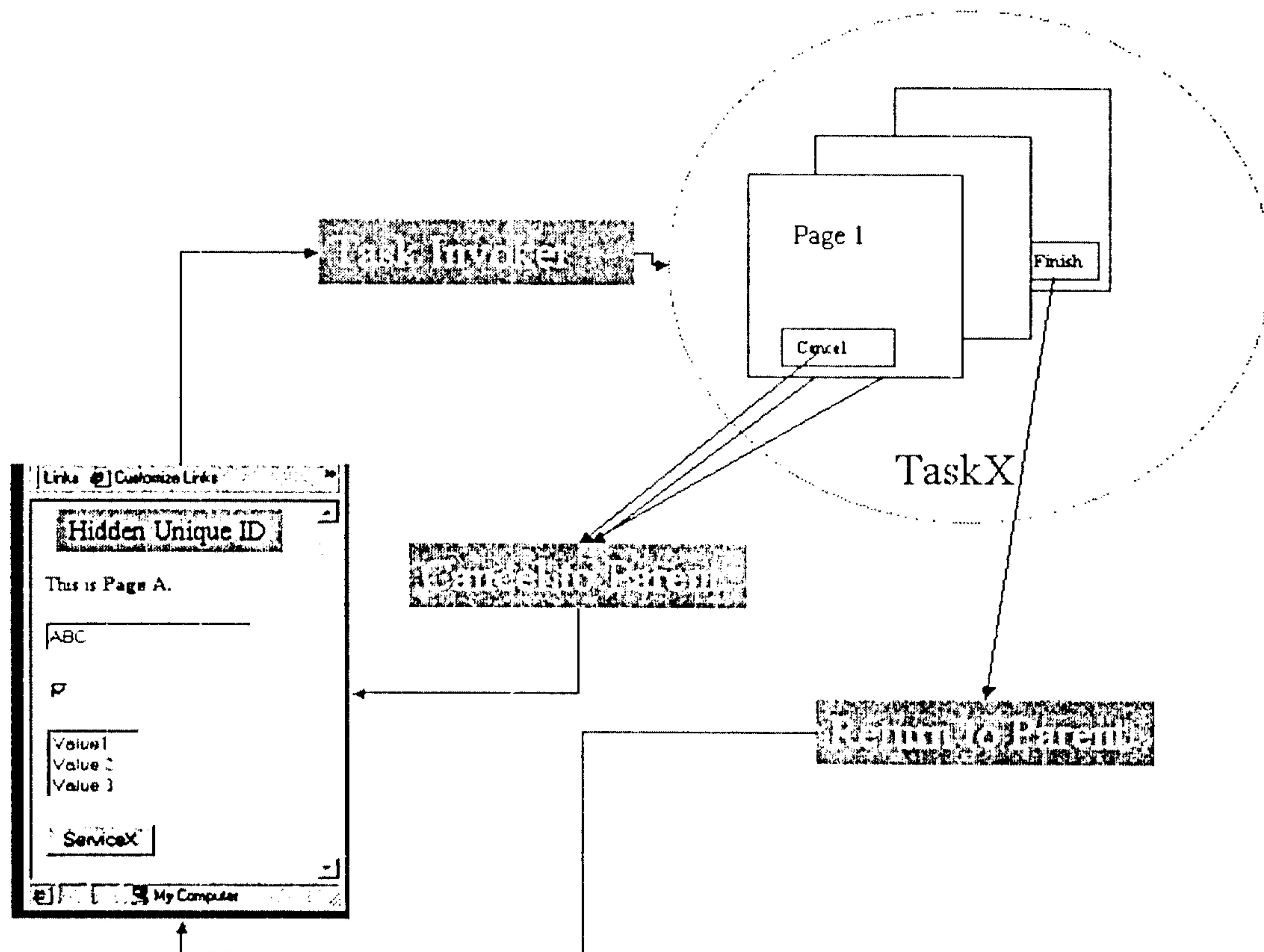


Figure 12

UNSCANNABLE ITEM

RECEIVED WITH THIS APPLICATION

(ITEM ON THE 10TH FLOOR ZONE 5 IN THE FILE PREPARATION SECTION)

#2310943

DOCUMENT REÇU AVEC CETTE DEMANDE

NE POUVANT ÊTRE BALAYÉ

(DOCUMENT AU 10 IÈME ÉTAGE AIRE 5 DANS LA SECTION DE LA
PRÉPARATION DES DOSSIERS)

