Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID-19. The COVID-19 resource centre is hosted on Elsevier Connect, the company's public news and information website.

INTERNATIONAL JOURNAL OF MEDICAL INFORMATICS 75 (2006) 741–754

journal homepage: www.intl.elsevierhealth.com/journals/ijmi

The ALPHA Project: An architecture for leveraging public health applications

Cameron Turner, Hany Bishay, Bo Peng, Aaron Merifield*

Information Technology Management Section, Public Health Agency of Canada, 130 Colonnade Road (A.L. 6501J), Ottawa, Ont., Canada K1A 0K9

ARTICLE INFO

Article history:
Received 14 April 2005
Received in revised form
20 October 2005
Accepted 20 October 2005

Keywords:
Public health surveillance systems
Application architecture
Reusable components
Infectious diseases

ABSTRACT

Objective: Public health surveillance applications are central to the collection, analysis and dissemination of disease and health information. As these applications evolve and mature, it is evident that many of these applications must address similar requirements, such as policies, security and flexibility. It is important a software architecture is created to meet these requirements.

Methods: We outline the requirements for a public health surveillance application, and define a set of common components to address these requirements. These components are configured to produce services used in the development of public health applications.

Results: A layered software architecture, the ALPHA architecture, has been developed to support the development of public health applications. The architecture has been used to build eleven surveillance applications for the Public Health Agency of Canada in the areas of disease surveillance, survey, distributed data collection and inventory management.

Conclusions: We have found that a software architecture that addresses requirements on policies, security and flexibility facilitates the development of configurable public health applications. By creating this architecture, key success factors, such as reducing cost and time-to-market of applications, adapting to changing surveillance targets and increasing user efficiency are achieved.

1. Introduction

Public health surveillance is the ongoing, systematic collection, analysis, interpretation and dissemination of data regarding a health-related event for use in public health action to reduce morbidity and mortality and to improve health [1]. A public health surveillance application is a software system designed to assist in these activities.

The ability to accurately monitor and track emerging and previously identified infectious diseases, such as Severe Acute Respiratory Syndrome (SARS), Avian flu, Bovine Spongiform Encephalopathy (BSE) and HIV/AIDS, is key to preventing outbreaks and epidemics. In the shrinking global village, an outbreak can quickly become a public health problem, so an effective public health system is necessary to protect the health of the population it serves [2]. Public health systems must be able to rely on modern surveillance applications to respond to an emerging epidemic by giving public health officials the information they need to make accurate decisions. The need for rapid comprehensive analysis means that antiquated and outdated systems are unsatisfactory for surveillance of modern diseases [2,3]. Technology systems within the public health community have generally been applied to narrow, categorical applications not easily integrated into functional applications that can monitor the health of communities [4]. Furthermore, public health surveillance applications unable to adapt to new surveillance targets become outdated, as data is collected with little

* *Corresponding author.*
E-mail address: Cameron_Turner@phac-aspc.gc.ca (C. Turner).
1386-5056/$ – see front matter
doi:10.1016/j.ijmedinf.2005.10.006

or no analysis and use of the corresponding information [5].

There are many public health surveillance systems in operation. A review of current surveillance systems reveals that there are over 66 detection systems in use in the U.S. and elsewhere [6]. At the Public Health Agency of Canada (PHAC), there are 35 different surveillance system applications in production, ranging from surveillance databases, survey applications, inventory and laboratory systems. Many of these PHAC applications are customized to be disease-specific, or are developed for a particular surveillance function (e.g., subject identification encryption). Furthermore, many of these PHAC applications are developed using different technologies. Maintaining these heterogeneous systems are fiscally and resource expensive.

Rapidly changing environments require the delivery of timely surveillance information [7,8]. In response, the ALPHA Project is an initiative that started in late 2002. The purpose of the ALPHA Project is to develop a software application architecture based on the philosophy of configuring and reusing common components to produce services that would be used to enable faster development of robust, maintainable public health applications. Public health is practiced through complex relationships of organizations (e.g., local, federal) and functionally organized units (e.g., health departments, disease programs) [9]. As such, the emphasis of the project is not on creating one monolithic application to handle all public health surveillance needs, but rather on creating customizable applications with the same underlying architectural or component structure. The goal is to reduce the amount of new development work required for each new application, in order to reduce its time-to-market. Configuration plays an increasingly larger role in the development cycle, and leads to more flexible applications responsive to new and emerging public health needs. New components or services that have to be built are designed in such a way that other applications can use them, and contribute to the architecture. Consequently, software becomes more maintainable, since applications share many of the same components.

This paper defines a software architecture that is used to build public health surveillance applications. It focuses on common components, which are configured to provide different services that are integrated into each application.

This paper is outlined as follows. In Section 2, we summarize the requirements identified for a public health surveillance application. In Section 3, the ALPHA architecture is described which meets these requirements. In Section 4, three applications, which have been built using the ALPHA architecture, are explained. Section 5 provides a description of how the data collected within these applications is analyzed. Related work is outlined in Section 6 while Sections 7 and 8 summarize the paper.

2. Requirements of a public health surveillance application

At the Public Health Agency of Canada, many different surveillance system applications have been developed using different technologies to collect data for specific diseases. By studying these applications, the following important, common requirements have been extracted:

- *Flexibility*: Tracking and monitoring diseases is a dynamic activity. Emerging diseases, such as SARS can appear very quickly, requiring public health officials and surveillance applications to respond just as quickly [10]. When a disease outbreak occurs, it is essential that the technology tools be in place to track cases and contact information. Public health officials do not have the luxury of time to develop systems in response to each disease outbreak. An application must be flexible to handle new diseases [10]. In addition, given the possible links between outbreak events, epidemiologists analyzing data collected by a surveillance system may need to gather different data as they discover new information. A surveillance application must be able to adapt to these changing data and analysis requirements.
- *Maintainability*: Applications are inherently expensive to maintain. The cost of supporting a deployed software product can be between 60 and 80% of its lifetime cost [11,12]. The greater the degree of heterogeneity that exists in the application suite available for an organization, the more expensive it is to maintain. For example, different applications tend to use different technologies and different versions of third party software. Keeping track of all the applications' technology version matrices can be complex, and the licensing costs can be expensive. Making viable the sharing of common architectural components among applications reduces support complexity and licensing costs.
- *Jurisdictional configurability*: Each jurisdiction collecting public health surveillance data has its own policies or business rules governing the content and format of data to be collected. For instance, larger jurisdictions may be able to collect last names of their subjects, while smaller jurisdictions may only be allowed to collect their initials. Similarly, some jurisdictions may want to include the complete date when collecting a birth date, while others can only record the month and the year. An application distributed to different jurisdictions must be able to handle these variations in policy. Furthermore, since policies can change within jurisdictions, established or already implemented applications must be readily configurable.
- *Alert notifications*: A core requirement of a surveillance application is to send alert notifications to designated people indicating that a certain event has occurred. An event could be the occurrence of two or more cases of a similar disease being reported in potentially multiple jurisdictions within a certain time frame (i.e., outbreak). Another event may raise attention to data anomalies within a case that requires attention. An event could even be a user accessing an application after hours. Events such as these help users to manage their applications and understand the data being stored. Alerts could take the form of an email notification, a message sent to a Personal Digital Assistant (PDA)/cell phone, or a posting to a web site.
- *Security*: Due to the sensitive nature of public health surveillance data, security is of the utmost concern. One area of concern within security is authorization. Particular individual users of applications may only see a subset of the data, and only certain functions of the application depending

upon their role. For instance, if an application collects data from different jurisdictions, a public health official in one jurisdiction should not be able to see data collected in another jurisdiction. Furthermore, access to different parts of the application should be restricted only to those who are authorized. Auditing functions can also assist in monitoring what applications and which data are being accessed by a specific user.

- *Usability*: Prior to the analysis and reporting of public health information, data must exist in a repository. Currently, data collection poses the single most resource intensive component of surveillance cost, since data must be manually entered into a repository. A disease case can require the collection of, potentially, a few hundred data elements. Based on our experiences, data entry clerks often complain of user interfaces being crowded with irrelevant data fields since not all data elements need to be collected for a case. To reduce this overhead associated with data collection, the presentation of a surveillance application interface must optimize data entry and filter out irrelevant data.
- *Data sharing*: Public health surveillance applications collect data about different diseases from different demographics. This data is usually stored in different databases. Valuable information could be discovered if field entries from these databases were cross-referenced, and in line with existing legislation and privacy controls stored in a central database. The Pan-Canadian Electronic Health Record project by Canada Health Infoway is a multi-year initiative focussed on integrating Canadian surveillance systems from a data perspective [13]. To support this important objective, a surveillance application must have the capability of sharing data with other applications.
- *Statistical analysis and reporting tools*: Once data has been collected and entered into the system, epidemiologists and other public health professionals must be able to analyze the data and report on its contents. A number of commercial off-the-shelf (COTS) statistical analysis and reporting tools adequately support this requirement. Thus, PHAC has elected to integrate COTS products into the surveillance environment to support this need.

3. ALPHA software architecture

This section describes the ALPHA software architecture proposed to address the requirements previously identified. We have found that as different applications evolve and mature, a pattern of solutions has been discovered within the software that can be abstracted and reused to address these requirements. This bottom-up approach forms the core of the architecture's components. These components can be instantiated with concrete data and other code to produce a usable service. Applications then use these services to provide the necessary functionality. Fig. 1 shows the ALPHA architecture.

3.1. Architectural layers

3.1.1. Component Layer

The Component Layer contains the building blocks that provide the framework for producing application functionality, but not the content. Therefore, a Component is not entirely usable on its own—it must be given a specific implementation.

For instance, a *Profiler Component* provides an access control framework. The Component, itself, is not concerned with the specifics as to what it is controlling access. The same is true with the *Business Rules Component*. This Component only provides the framework, language and inference engine to enable rules-based logic. It does not provide any actual rules. It is through the use of these common components that the maintainability requirement is addressed; these components form the core of the underlying structure of each application.

3.1.2. Service Layer

The Service Layer provides a set of common services for use in the Application and Configuration Layers. These Services encapsulate a specific set of functions, which can be easily integrated into an application. These services are also used between themselves.

For instance, a *Disease Access Service* uses the Profiler Component to provide the access control functionality specifically for case information on different diseases. For this service, an instantiation of the Profiler Component with the addition of disease-specific data creates a Service.

3.1.3. Configuration Layer

The Configuration Layer provides a set of common tools that can be combined with applications in the Application Layer to deliver a fully functioning application suite. These tools handle certain configurations of the system.

For instance, the *Business Rules Manager* facilitates the creation, modification and deletion of business rules and policies. The *Access Manager* permits the configuration of the authorization privileges for organizations, regions and users. Finally, the *Application Manager* allows for viewing and configuring logs and audits. These tools require very little modification to be adapted to be used in different applications.

3.1.4. Application Layer

The Application Layer consists of the different applications comprising the entire system. Applications, such as Infectious Disease Surveillance System (IDSS), Anti-Microbial Resistance Surveillance System (AMRSS) and Enhanced Surveillance of Canadian Street Youth (ESCSY) exist at this layer. These three examples are presented later.

3.2. Architectural components

This section outlines the building blocks of the Component Layer, which provides the core of the architecture.

3.2.1. Profiler

Public health is practiced through complex relationships of organizations (e.g., local, federal) and functionally organized units (e.g., health departments, disease programs) [9]. As a direct consequence, public health applications must be flexible, secure and able to handle differences between jurisdictions. The Profiler Component is designed to meet these requirements.

The Profiler is a generic component customized to manage the authorization of an entity across different organizations.

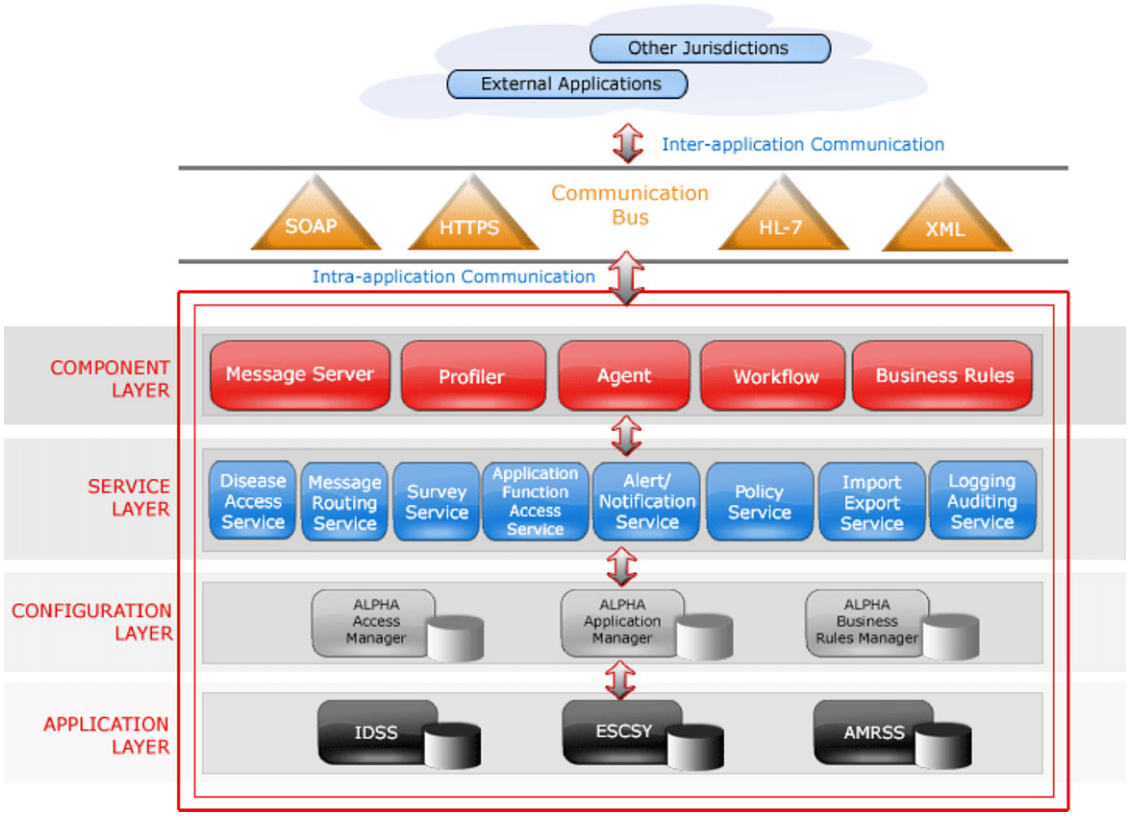


Fig. 1 – ALPHA architecture.

An entity is defined as anything that should be restricted at some level (e.g., a disease data element or an application function). The configuration of accessibility to these entities creates a *profile*. Profiles are handled through the Access Manager in the Configuration Layer, and this tool is used to assign multiple profiles to a user.

We model a profile using a hierarchical data model where the entity structure is based on categories, properties and attributes, and the organizations are based on groups and units within the organization. In our model, a category can have ***n*** properties. Each of these properties can have ***n*** attributes. For instance, a disease (category) can have multiple sections to its data entry forms (properties) that, in turn, can have multiple data elements (attributes). Similarly, an organization can have ***n*** regions (groups). Each of these groups can have ***n*** districts (units), and each district can contain many users. Fig. 2 shows the data model for a profile.

As an example, this data model provides an authorization service for diseases, and so it is possible to control access for every data element for each user. This addresses the security requirement as it prevents a user in one district from viewing or modifying public health data in another district.

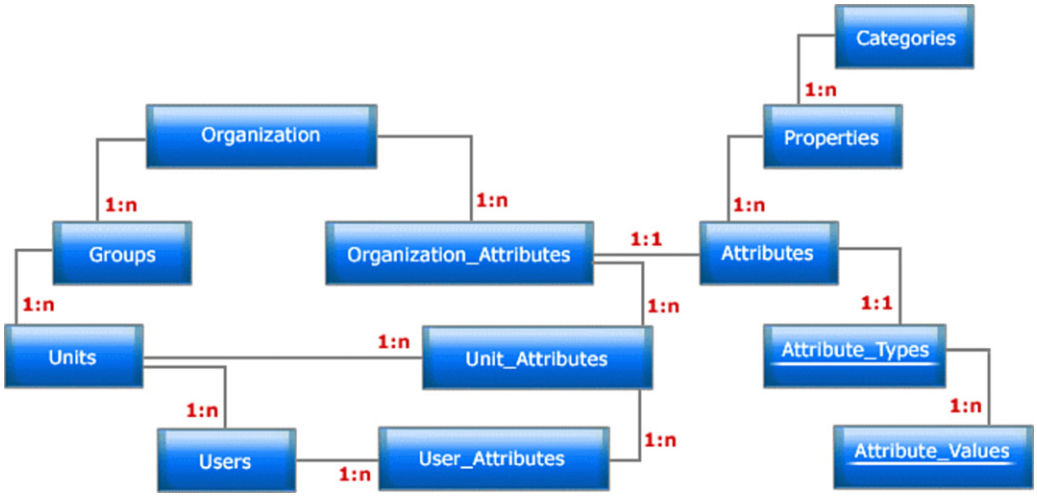


Fig. 2 – Data model of a profile.

Furthermore, we can also use this data model to provide an authorization service for application functions. Access can be easily configured to accommodate some users in the same district from accessing certain functions within an application, while allowing another user in the same district to have full access.

Since the Profiler Component is a framework that is customized using data, new entities such as emerging diseases can be added or modified easily by inserting data specific to the new requirement, thus addressing the flexibility requirement. An application that can be adapted to handle the addition of emerging diseases solves one of the problems that hindered the fight against SARS [2,3].

Finally, the Profiler Component provides the ability to handle differences between jurisdictions. Some jurisdictions may want to collect surveillance data that others may not need. The non-required data elements are then 'switched off', so they do not appear in the application. As a result, data collection is streamlined and application navigation is optimized at no additional cost to the development process.

3.2.2. *Message Server*

Software systems are typically comprised of disparate applications, each executing in different process spaces. These applications can reside entirely on one machine, or be distributed over several machines. One of the requirements identified in the previous section outlines that there are times when applications need to share information. For this reason, a communications component is required to handle inter-application communication. The *Message Server* is designed to meet this requirement.

Messages are transmitted between applications over a secure HTTP link using the Simple Object Access Protocol (SOAP). These messages are modeled using XML.

We use a Service Locator to identify whether the service to be invoked should be done locally or remotely. For instance, a service to retrieve a patient's current active cases may be done within an application itself (i.e., locally). However, a service to retrieve a patient's case history may require sending a message to a central application, which, after authorizing the request, sends back the required information (i.e., remotely). The ability to configure where services are invoked gives flexibility to re-route messages based on current needs.

Once the decision has been made for a service to be invoked remotely, a Message Creator creates the XML message using the correct schema. This message is transmitted to the receiving application that uses a Message Parser to interpret the XML message, and a Message Handler routes it to the correct service. Since SOAP is based on Remote Procedure Calls (RPC), results are returned to the calling application. Fig. 3 shows the Message Server Component Framework.

3.2.3. *Agents*

An Agent is an autonomous entity that is assigned specific tasks to perform. These tasks, typically performed as a background process, can assist in addressing jurisdictional configurability and alert notification requirements through the use of business rules. Therefore, an Agent's duties involve the periodic collection of rules and data and applying these rules to the data.

We model an Agent using the observer design pattern [14]. In this design pattern, there exists a 1:***n*** relationship between a subject and its observers. When the state of the subject is changed in any way, the subject's observers are notified so they can take the correct course of action. In our Agent model, the subject is a *gatherer* that retrieves information from a data source (e.g., surveillance data stored in a relational database). Once the gatherer has retrieved the information it requires, the observers are notified of the event. The observers then retrieve the information from the gatherer, analyze it and take the appropriate course of action. A Notification Receiver is informed of the results. We have implemented the Agent using a *thread* so that it can operate on a sleep–wakeup schedule.

For example, an Export Agent wakes up at a predetermined time and retrieves information from an application's database instance. One observer is setup to export this data to a centralized repository, while another observer is

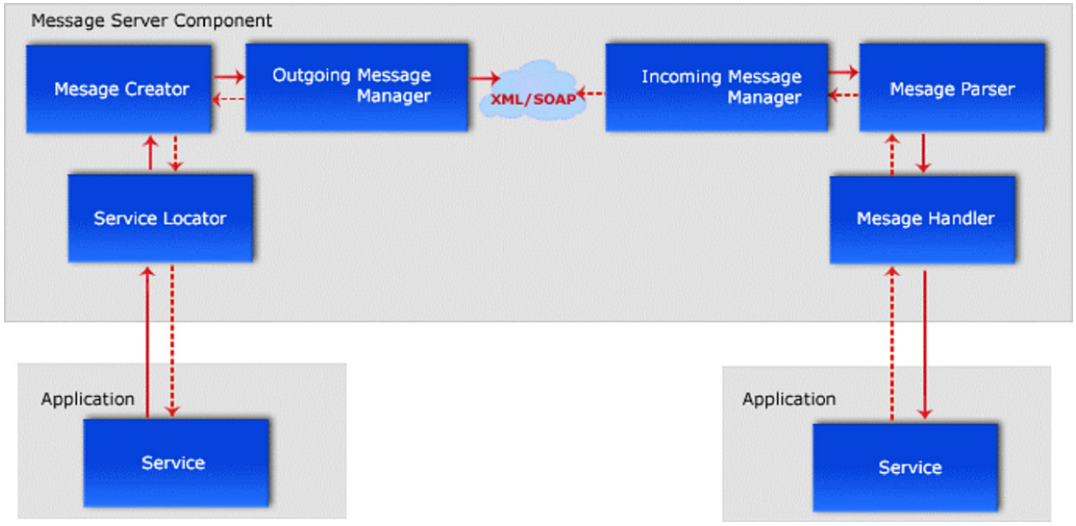


Fig. 3 – Message Server Component framework.

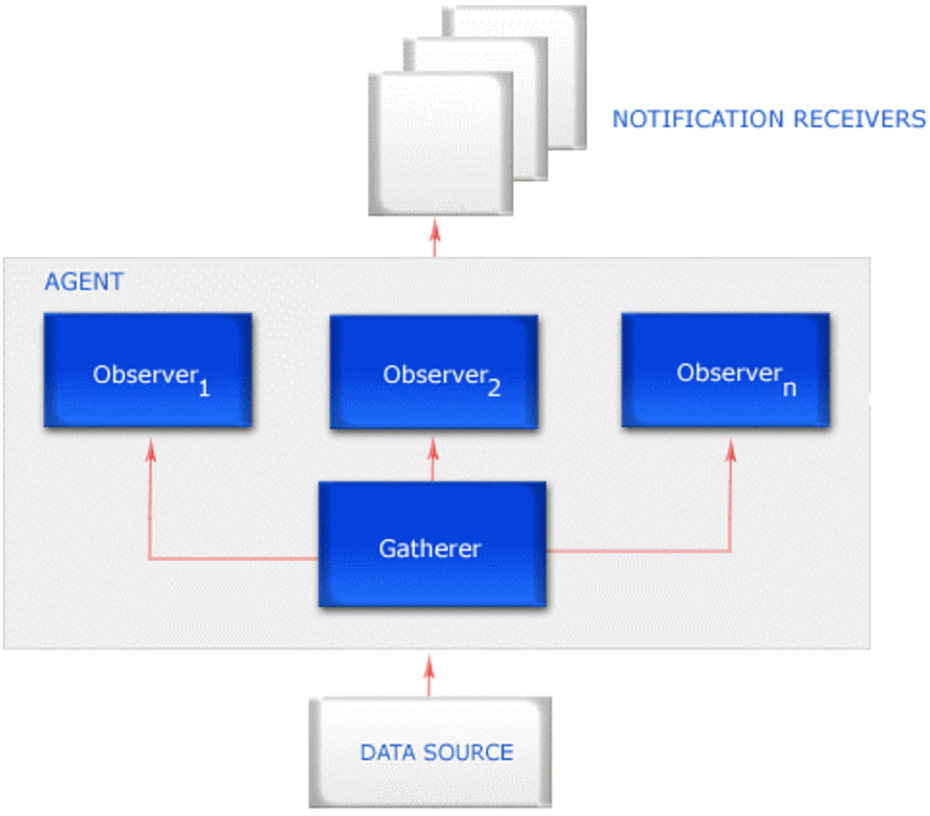


Fig. 4 – Agent component framework using the observer design pattern.

setup to export this data to another application. Fig. 4 shows the Agent Component Framework.

3.2.4. *Workflow*

The *Workflow* Component defines a set of tasks to be completed, and the order in which they should be completed. These tasks can be a set of screens to display, or a set of automated work items to perform. One example of a workflow is an *escalation* scenario, whereby each task performs a higher-degree work item than its predecessor.

The Workflow Component assists in addressing the jurisdictional configurability and usability requirements. Workflows are configured based on the jurisdiction in which the application is deployed without affecting the underlying code.

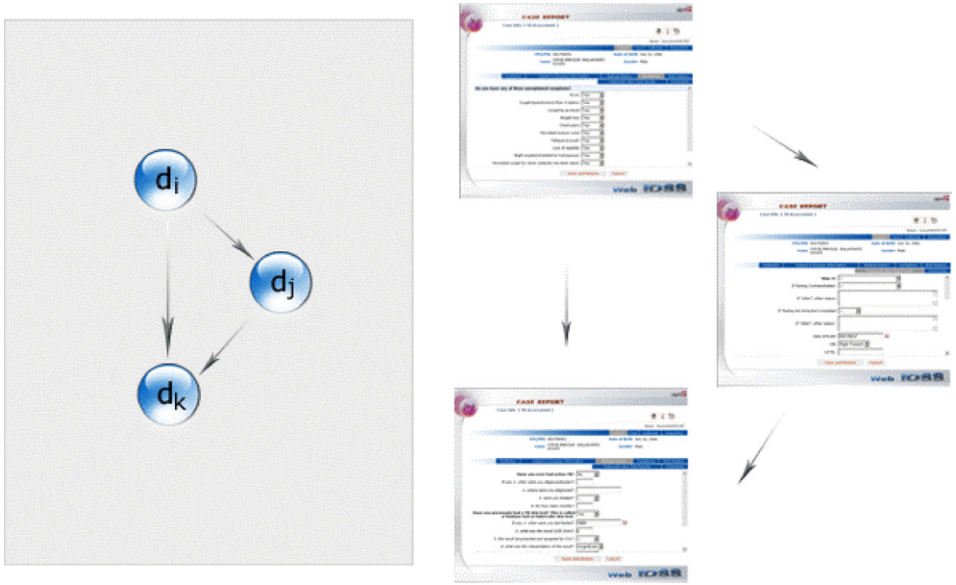


Fig. 5 – Modeling a workflow using a graph.

```
IF Exists (DateOfBirth) &
[
     Empty({DateOfBirth()} |
     !ValidFormat ({DateOfBirth}, ####/##/##) |
     !isDate({DateOfBirth})
]
THEN SetSessionVariable(Subject Data Error, {Subject Data Error}
                        \n- Date of birth is invalid. Expected format: yyyy/mm/dd)
```

Fig. 6 – Business Rule to validate date of birth.

Also, data entry is facilitated as a user is guided through the entire process.

We model a workflow as a graph, $G = (V, E)$, where V represents the tasks and E represents the action required to move from one task to another. For any, $t_i, t_j \in V$, there exists $(t_i, t_j) \in E$ if t_i is a task that must be completed before completing t_j (i.e., t_j depends on t_i).

For example, assume three data entry screens exist, d_i, d_j, $d_k \in V$. Selecting a value from a data element's drop-down menu in the first screen, d_i, may lead to the second data entry screen, d_j, to get more information. In this case, $(d_i, d_j) \in E$. After completing data entry on the second screen, d_j, the user is next shown the third data entry screen, d_k. Therefore, $(d_j, d_k) \in E$ exists. If the value on the first data entry screen, d_i, which leads to the second data entry screen, d_j, was not chosen, the user is shown the third data entry screen, d_k. Therefore, $(d_i, d_k) \in E$ also exists. Fig. 5 shows this example's workflow.

Modeling a workflow using a graph allows us to conduct a depth-first search in order to determine all paths in the graph from one node to another. This assists in determining what data needs to be removed based on revisiting a previous node, and choosing an alternative course.

We enforce certain rules, since workflow graphs can become quite complex. A workflow must have a single start node and a single end node. Furthermore, while navigation can occur between a node and the node that preceded it, in order to avoid infinite loops an action cannot directly point to a preceding node.

3.2.5. *Business Rules*

Business Rules are capable of monitoring data stored in a database or memory, and making intelligent decisions based on that data. Since databases house large amounts of data in distributed tables, valuable information can reside undetected among these tables. Information such as a user logging in at an abnormal time, or the frequency of a person visiting different clinics within a 24-h period, could be critical knowledge to some application users. Therefore, monitoring and extracting information from data stores is of great importance.

We model Business Rules using a rules-based expert system. Expert systems encode the knowledge of domain experts in order to solve particular problems. In a rules-based expert system, these problems are solved using production rules. A production rule is stated in the form: $P \rightarrow Q$, which states *IF* P *THEN* Q, where P represents a set of *premises*, or *conditions*, and Q represents a set of *conclusions*, or *actions* [15]. The conditions can be linked together using Boolean logic (i.e., AND, OR, NOT) and organized into sub-conditions using parentheses (i.e., order of operations). Testing these production rules and firing, or executing, those rules whose conditions are satisfied, transform an initial state of knowledge into a new state of knowledge. Problems can therefore be solved in a logical manner.

The rules and the data upon which those rules act are stored in a *knowledge base*. The knowledge within the knowledge base can constantly change as data is added, modified or deleted when different rules are executed. Furthermore, the rules can also be added, modified or deleted, without modifying the source code, in order to dynamically evaluate the rule set. This helps reduce code complexity, as increasing the amount of logic needed to handle many different scenarios can lead to difficult to read code, or *spaghetti* code. Spaghetti code can be difficult to thoroughly test and debug [16]. Furthermore, hard-coding logic inevitably leads to multiple software configurations of the same application, which further complicates maintenance [12]. Therefore, part of a business rules system's power is not locking an application into a predetermined set of rules.

Due to the dynamic nature of the Business Rules Component, the jurisdictional configurability and alert notification requirements are addressed. Policies can be encoded as rules are deployed to different jurisdictions as seen fit. Furthermore, alerts and notifications can be configured depending on the data to be monitored and the behaviour to detect.

We use forward chaining for the business rules inferencing engine. Forward chaining systems start with an initial state, where certain facts are known, and the rules are used to infer an end result, where the conclusions are initially unknown [15]. For example, Fig. 6 shows a Business Rule to validate date of birth.

This Business Rule evaluates a data element to ensure that the integrity of the information collected is intact. In this case, the Business Rule ensures that if the date of birth has been specified, it cannot be empty, it must be in the format yyyy/mm/dd, and it must be a valid date. Such a policy would not be acceptable in a small jurisdiction where the collection of a subject's full birth date would violate legislation. The policy deployed in that jurisdiction would be changed to only accept four digits. In both cases, the underlying application code remains untouched.

4. Case studies

This section outlines case studies of three applications built using the ALPHA architecture. In the past two and a half years, we have built 11 applications for the Public Health Agency of Canada using the ALPHA architecture. The three applications

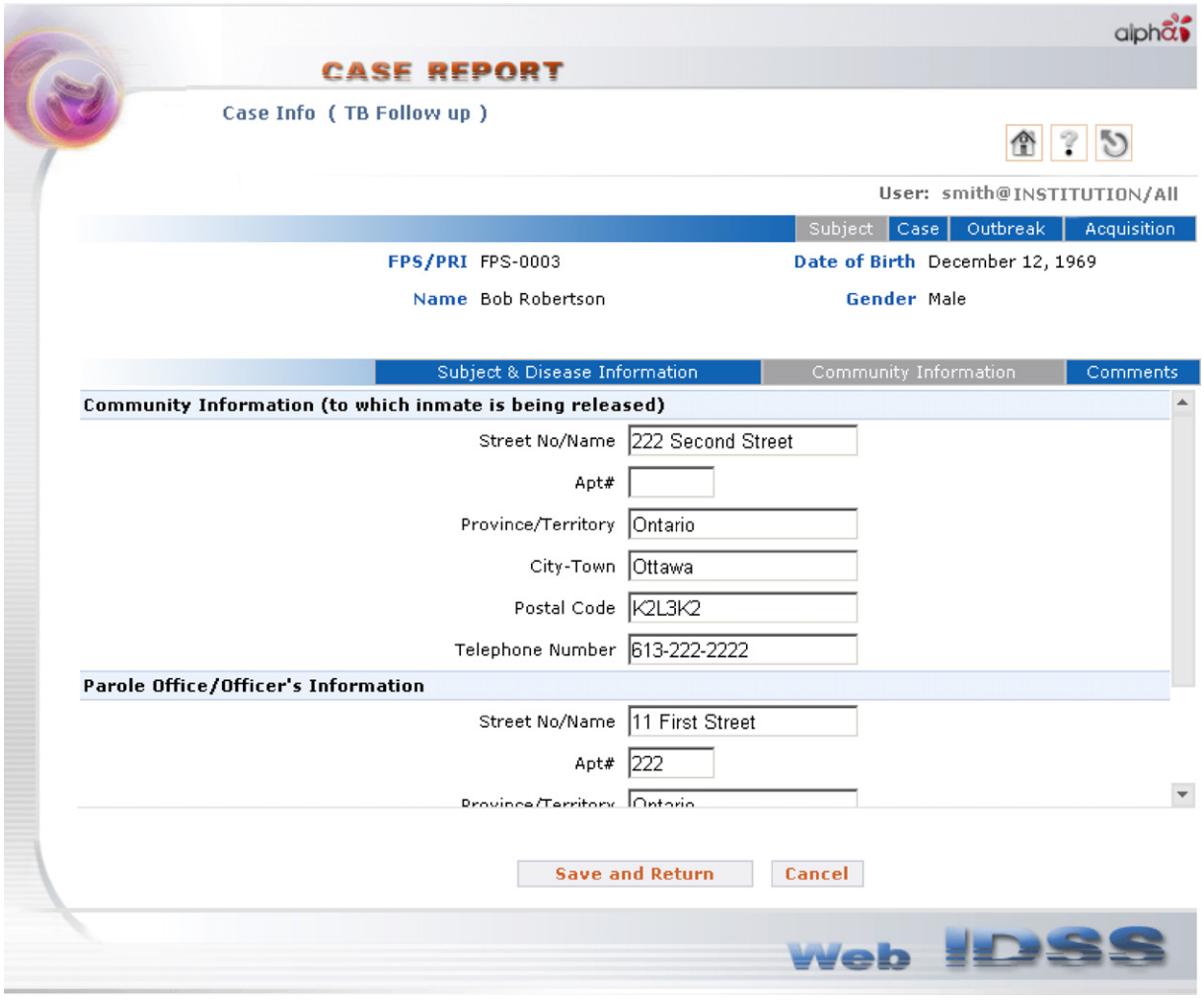


Fig. 7 – Smith—partial access to Robertson's data.

presented in this section were chosen since they show a wide range of services used.

4.1. *Infectious Disease Surveillance System*

The Infectious Disease Surveillance System is a web-based application that collects case information on tuberculosis (TB) and sexually transmitted diseases (STDs) in Canada's federal penitentiary system. This data is sent to Correctional Service Canada (CSC) for data entry and analysis.

The following central services are found in IDSS.

4.1.1. *Disease Access Service*

The Disease Access Service is a service that controls users access to case information. The access to information is both at the presentation level (what the user can view), as well as at the data level (what the user can retrieve, update or report).

The Disease Access Service uses the Profiler Component to implement its service. Users in different roles and different locations can only access data they are permitted to see. Therefore, a user in one institution cannot view data about subjects in another institution.

An administrator, when creating a user, controls the level of access to the disease case information. The administrator, for example, can restrict access to individual data elements on a form, a section of the form, or the entire form. Therefore, it is possible for users to see tests that have been run on a particular subject, but not the test results.

Figs. 7 and 8 show how a user in an institution can have different views of the data depending on the user's role.

4.1.2. *Jurisdictional Policy Service*

The Jurisdictional Policy Service is a service that handles the different policies of jurisdictions. Each jurisdiction may have different requirements for its data. For instance, in one jurisdiction only an initial can be stored for a first name, while no such policy exists in another jurisdiction.

The Jurisdictional Policy Service uses the Business Rules Component to implement its service. Rules are encoded into the Business Rules Component, and when a user attempts to save data, these rules are tested for data violations. Since these rules are not hard-coded into the application, rules are created, deleted or modified as seen fit. Therefore, if one policy rule is not applicable in a jurisdiction, it can be deactivated or removed.

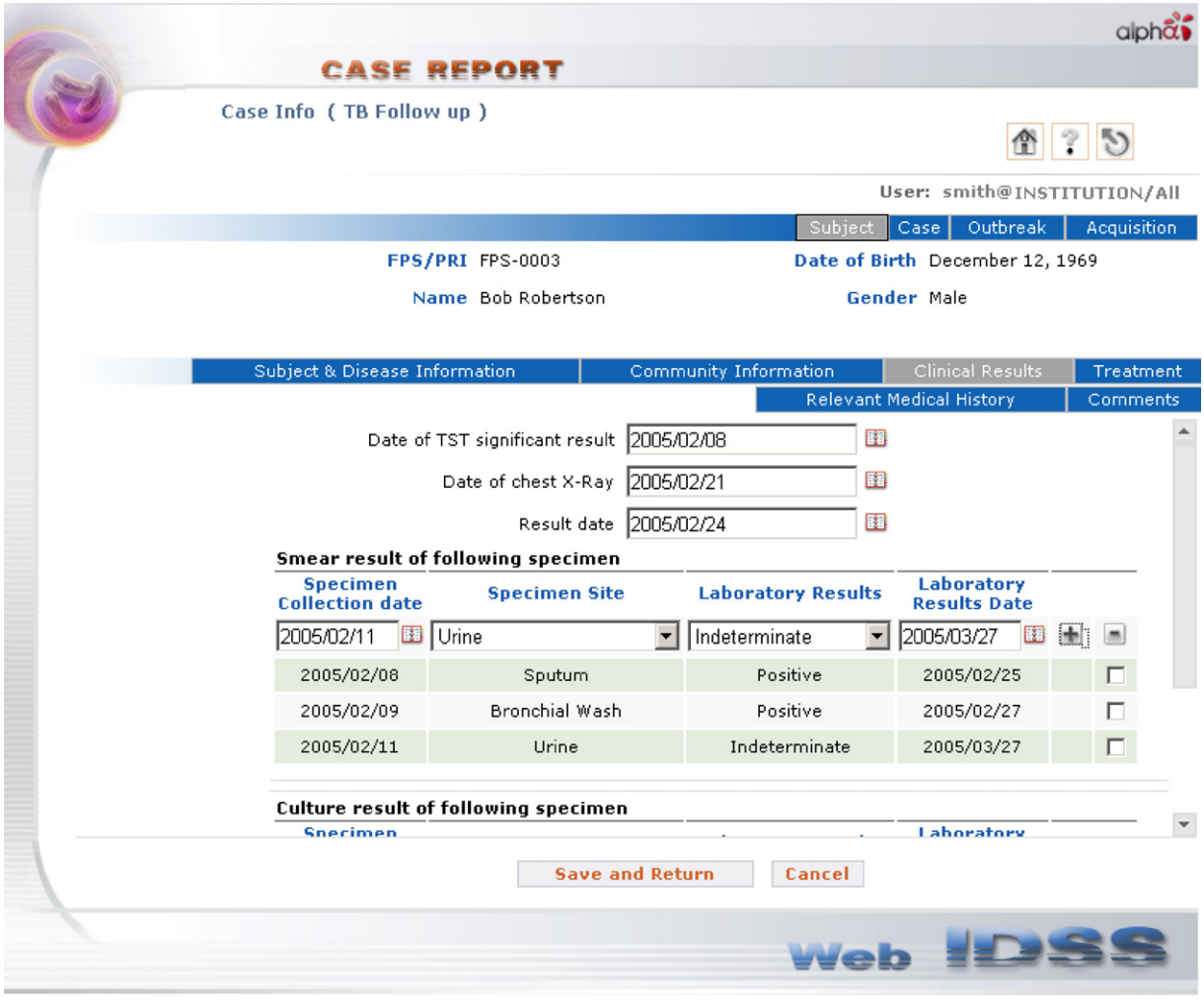


Fig. 8 – Smith—full access to Robertson's data (part of clinical results shown).

A *Jurisdictional Policy* rule was shown in Fig. 6.

4.1.3. *Message Routing Service*

The Message Routing Service is a service that handles the communication between different applications. Applications may need to share data with other applications, or get data from other data sources. Therefore, a communications service is required.

The Message Routing Service uses the Message Server and Business Rules Components to implement the service. This service extends the Message Server Component by implementing an XML schema and parser to send generic retrieval and update requests. These requests allow the application to use local as well as remote databases as its data source.

4.2. *Anti-Microbial Resistance Surveillance System*

The Anti-Microbial Resistance Surveillance System is a distributed, data collection system designed to gather information into one central repository for reporting and analysis purposes.

There are three separate locations, Guelph, Ontario; St. Hyacinthe, Quebec and Winnipeg, Manitoba that currently input their disease and biological data from animals, food and humans into their respective local databases. This data is then exported periodically (e.g., every 15 min) into the central repository in Ottawa, Ontario.

The creation of a central repository enables the assessment of event relatedness, detection of time trends and geographical patterns. Furthermore, resources at the separate locations have the ability to analyze their own data, and also the capability to perform analysis on the integrated data.

The data is simultaneously accessible by selected resources in different office locations, and within different departments.

The following central services are found in AMRSS.

4.2.1. *Message Routing Service*

The Message Routing Service uses the Agent, Message Server and Business Rules Components to handle communication. This service defines an Agent to periodically send messages containing disease and biological data collected from animal, food and human specimens from the local databases to the central repository. This service extends the Message Server Component by implementing the necessary XML schemas and parsers. Business Rules are used to route these messages to the correct service.

Fig. 9 – ESCSY survey screen.

4.2.2. *Alert and Notification Service*

The Alert and Notification Service is a service that handles raising events of interest to a designated person or persons. An event of interest could be an error condition, a policy violation, or some other event of which a person should be notified.

The Alert and Notification Service uses the Business Rules Component to implement its service. Rules are created to look for data transformation, data transfer or data integrity failures. For instance, if a record is not imported correctly into the central repository, the local Administrator must be notified of this event. The local administrator must also be notified if a record does not contain the correct mandatory fields. If such errors are detected, an error report is created and sent via email to the local Administrator. The Administrator can then use the report to remedy the situation.

4.2.3. *Application Function Access Service*

The Application Function Access Service is a service that handles access control to various functions of an application. The privileges a user has dictates what that user can and cannot do within an application.

The Application Function Access Service uses the Profiler Component to implement its service. The data managed within the component is organized into the applications (e.g., Administration Tool, Data Viewer), properties of these applications (e.g., User Management, Code Management), and attributes of these properties (e.g., Create User, Delete User). A local Administrator is permitted to create user accounts for those at their location; however, they are not permitted to change the values of the data element pick lists. The central Administrator has access to all application functions.

4.3. *Enhanced Surveillance of Canadian Street Youth*

The ESCSY system is a web-based application that collects information on sexually transmitted diseases in street youth. Data is collected from youth in their teens and early twenties by public health nurses who use paper-based surveys. These surveys are then sent to PHAC for data entry and analysis. The following central services are found in the ESCSY system.

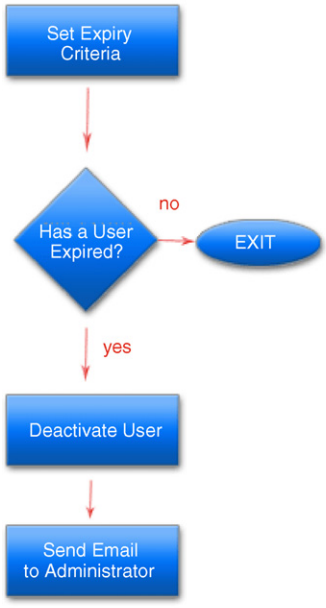
4.3.1. *Survey Service*

The Survey Service is the central service in the ESCSY application. It is responsible for loading questions, saving responses and assisting in navigating through the survey.

The Survey Service uses the Workflow Component to implement its service. The Workflow Component permits this service to implement features such as Branch/Skip Logic to bypass questions that are based on answers to previous question; Conditional Logic to control such things as gender-specific questions from being answered by the wrong gender; Full Navigation to go to the first and last questions, the previous question, the next question, as well as being able to specify a specific question and Data Integrity to prevent navigation to those questions that cannot be viewed. Fig. 9 shows an example of a question.

4.3.2. *Jurisdictional Policy Service*

As in the IDSS application, the Jurisdictional Policy Service uses the Business Rules Component to handle policies. The ESCSY application has certain security policies to enforce. One of the policies is user access expiry. Survey data is entered using data entry clerks, typically students. In order to prevent access to those that have not logged onto the system in




```
IF   !Exists(Expired Users) | !Exists(Expiry Criteria)
THEN SetSessionVariable(ExpiryCriteria,
                        last_access_date < sysdate - 60); // days

     SetSessionVariable(Expired Users, select username
                         || ' last accessed the system on '
                         || last_access_date from users where
                         (Expiry Criteria))

IF   Exists(Expired Users) & !Empty({Expired Users})

THEN update users set status = 'Inactive' where
     (Expiry Criteria);

     Email ("The following users' access have expired",
            {Expired Users}, "admin@domain")
```

Fig. 10 – Rules implementing the user access expiry policy.

a certain timeframe, a policy is in place to prevent access to those not involved with the project anymore. This is a policy that is not required by our other survey applications that use permanent staff.

Fig. 10 shows the two rules involved to implement this policy. The rule engine, operating in stand-alone mode, tests these rules every day. If the rule finds an expired user, the rules will deactivate that user and send an email to the Administrator explaining the action taken.

4.4. *Management Services*

Two other services used by these applications are outlined below. These services assist in the management of the application environment.

4.4.1. *Logging and Auditing Service*

Logging and auditing are fundamental to all applications, since they record exceptions, errors and other events of interest. The Logging and Auditing Service provides a level of security by monitoring, and capturing application and network access, messaging requests, failure events and system misuse.

The Logging and Auditing Service uses the Agent component to implement its service. Applications built using the ALPHA architecture write all their logs and audits to the file system. The Logging and Auditing Agent periodically collects the messages stored in these files and stores them in a database.

4.4.2. *Application Management Service*

The Application Management Service works with the Logging and Auditing Service to provide application-level management. The Application Management Service is based on the FCAPS model (Fault, Configuration, Accounting, Performance and Security) [17].

The Application Management Service uses the Agent and Business Rules Components to implement its service. The Application Management Agent periodically analyzes the log messages stored in the database from the Logging and Auditing Service to determine if policy violations have occurred. For instance, the Application Management Agent can determine if a fault exception has occurred which needs immediate attention (e.g., an application has produced a critical fault), or that some other potential faults have occurred (e.g., 10 warnings from an application within 1 min). Furthermore, the Application Management Agent also determines if a security breach has occurred (e.g., an application has produced an audit trail of a user logging in after work hours to retrieve sensitive information).

5. Analysis of Public Health Surveillance Data

The applications we have developed using the ALPHA architecture provide surveillance officers and analysts the tools they need to do their jobs. Preliminary analysis is done using the Jurisdictional Policy Service to perform frequency analysis and correlation of variables relevant to that jurisdiction. For instance, in IDSS, business rules have been created to run a frequency analysis on variables such as *Tests Completed* or *Positive Tests* in order to generate a summary report on HIV/AIDS, Hepatitis A/B/C and Tuberculosis. Monthly graphic or textual reports can include region, institution and disease form-specific information.

More in-depth analysis and statistical reporting were deemed outside the scope of our work based on the number of readily available commercial applications. However, the surveillance officers and analysts are provided a Data Export Service that extracts the information from their application into a text file based on their specific variables and formats.

These text files are then imported into a tool such as SAS (SAS Institute Inc., Cary, NC), and analyzed using various statistical analysis methods [18].

Results from data collected within the ESCSY application have been presented at a variety of forums [19–22].

6. Related work

A number of surveillance systems currently exist, as well as a few initiatives to create an inter-operable network of coordinating systems. We discuss one of these surveillance systems and two of the initiatives under way.

The Real-Time Outbreak and Disease Surveillance (RODS) system [23–25] is a syndromic surveillance system developed in the RODS Laboratory at the University of Pittsburgh. A syndromic surveillance system is designed to identify outbreaks based on reported symptoms that precede a diagnosis [26]. In the RODS system, data is collected from patients' chief complaints during emergency department visits, as well as patient registration at acute care clinics. After removing patient identifying information, the data is automatically sent to the RODS system using a Health Level-7 (HL7) message. The RODS HL7 listener parses this message and routes it to a Bayesian text classifier, which assigns it to a syndromic category. The data is then stored in the database for other applications to use. This data can then be analyzed to detect disease and bioterrorism outbreaks. The RODS system is deployed within several health systems in the United States.

Two of the initiatives to create a network of surveillance systems in order to enhance surveillance in a larger domain are the Canada Health Infoway project, and the Public Health Information Network (PHIN) project at the Centers for Disease Control (CDC).

The Canada Health Infoway Blueprint [27] is a government funded program to create a Pan-Canadian Electronic Health Record System (EHRS). The conceptual architecture of the blueprint outlines how Point-of-Service applications (e.g., case management applications at clinics, and hospitals) send information using standardized messages (e.g., HL7) to a Health Information Access Layer (HIAL). The HIAL, which defines services that can be used by inter-operating networks, stores information it receives into the appropriate repositories and registries.

The CDC is working on a framework to implement a standards-based network of inter-operable public health care systems [9]. The PHIN Functions and Specifications [9] outline components for those Intranet and Internet-based health systems that transmit data with their public health partners (e.g., laboratories, local public health agencies). These components are focused on detection and monitoring, data analysis, knowledge management, alerting and response.

The Canada Health Infoway Blueprint and PHIN Framework are primarily interested in building an inter-operable architecture for a cross-jurisdictional network, whereas the purpose of our work is to build an architecture for creating applications that would live in this network. Both the Canada Health Infoway Blueprint and PHIN Framework work on the principle that a secure, standards-based network of public health systems lead to better public health management and response.

7. Discussion

A key success factor for creating an architecture for public health applications is that the application must be useful to a surveillance officer. If surveillance officers cannot change the surveillance targets in a timely manner, they cannot meet their public health objectives. Therefore, the architecture, in order to be successful, must be flexible enough to allow both existing and new applications the ability to adapt to new surveillance targets so that surveillance officers can collect and analyze their data efficiently. In the ALPHA architecture, the data elements and data types collected within IDSS were easily modified after the surveillance officer required different data elements. Also, in another application, the Canadian Tuberculosis Reporting System (CTBRS), we were able to adapt to a set of different disease data forms. In both cases, the changes were made through configuration without modifying the underlying code.

Another key to the success of an architecture for public health applications is that the applications should assist users to enter or retrieve data efficiently so they can spend less time on administrative tasks and more time on their primary work (e.g., health care professionals spend more clinical time with patients, surveillance officers and analysts spend more time analyzing data). In the ALPHA architecture, feedback from the surveillance analysts has indicated this objective has been met, since they now have the ability to configure access to specific disease elements.

Finally, one more key to the success of an architecture is that each application's software development cycle should be reduced as time goes on. Consequently, there will be a drop in costs associated with each new application. In the ALPHA architecture, the ESCSY survey application took three developers 5 months to design and develop. By the time the third survey application was built, it only took one developer 1 month to configure the necessary Survey, Alert/Notification and Policy Services. Furthermore, IDSS took five developers 9 months to design and develop, but a similar surveillance application, CTBRS, took one developer only 3 months to configure the necessary Disease Access, Alert/Notification and Policy Services. Once again, the increased reliance on application configuration rather than development allowed the ALPHA architecture to meet these goals.

Furthermore, the work we have done on the ALPHA architecture has provided us a lot of information on building public health applications. These lessons are being applied to extend our work in the future.

The usability of an application provides one of the biggest sources of frustrations for our clients. We have addressed this issue using the Profiler Component and, more specifically, a service such as the Disease Access Service in IDSS. This Service only presents the necessary data elements to a user based on their role, privileges and interest. Therefore, a data entry clerk does not have to sort through a screen full of irrelevant data elements in order to enter data into one or two fields. Furthermore, most of the field surveillance officers enter data onto paper forms that data entry clerks enter into the system. We are currently investigating the feasibility of using Personal Digital Assistants (PDAs) that can be distributed to the field in

order for the data to be entered once and synchronized with the application at the end of each day. This will help automate the data entry process. An issue still to be resolved is the privacy concerns of information stored and transferred using a PDA.

We continue to investigate the extension and addition of new components and services through analysis of our software as it is produced. For instance, we are currently expanding our Data Entry Workflow Service for a new project that involves directing the user to different screens (tasks) based on information they provide. This service, like the Survey Service in the ESCSY application, uses the Workflow Component. To assist in this activity, we are also developing a Workflow Manager tool that will reside in the Configuration Layer to automatically create, edit and evaluate workflows.

Our latest project, a prototype for a Mobile Clinic System, is designed to be used for special events. This system, which is a combination of the IDSS and AMRSS systems, link up distributed, mobile clinics to a centralized repository to present aggregate data. Business rules are created dynamically to monitor for anomalies, such as a high frequency of symptoms from one particular clinic or from all clinics.

Although the applications we have implemented so far have been based on infectious diseases, they can easily be adapted to track chronic diseases and other conditions. Integrated systems that monitor and track a wide range of public health concerns can lead to a better understanding of certain diseases [28].

Finally, we continue to monitor the activities and advances made by the Canada Health Infoway and PHIN projects so that our applications can integrate with these networks using messaging standards and protocols such as HL7.

8. Conclusion

In this paper, we have described an architecture that can be used to build public health applications. The architecture is based on four layers: a Component Layer, a Service Layer, a Configuration Layer and an Application Layer. We have outlined the components that form the core of the architecture. We have presented as examples three applications built using the architecture and the common service elements. These services illustrate how the components were configured in order to produce the application products.

Acknowledgements

This work is supported and funded by the Centre for Infectious Disease, Prevention and Control (CIDPC), Public Health Agency of Canada; Information Technology Management Section (ITMS), Public Health Agency of Canada and Correctional Service Canada (CSC).

REFERENCES

[1] CDC. Updated Guidelines for Evaluating Public Health Surveillance Systems: Recommendations from the Guidelines Working Group. MMWR, 50 (No. RR-13), 2001.
[2] D. Naylor, D. Butler-Jones, S. Basrur, M. Bergeron, R. Brunharn, G. Dafoe, M. Ferguson-Paré, F. Lussing, A. McGeer, K. Neufeld, F. Plummer, Renewal of Public Health in Canada, October 2003, pp. 28–29, 65, 92, 97.
[3] A. Campbell, The SARS Commission Interim Report: SARS and Public Health in Ontario, April 2004, pp. 101–111.
[4] W.A. Yasnoff, J.M. Overhage, B.L. Humphreys, M. LaVenture, A national agenda for public health informatics; summarized recommendations from the 2001 AMIA Spring Congress, J. Am. Med. Inform. Assoc. 8 (2001) 535–545.
[5] WHO, An integrated approach to communicable disease surveillance, Weekly Epidemiol. Record 75 (1) (2000) 1–8.
[6] V. Dato, M.M. Wagner, A. Fapohunda, How outbreaks of infectious disease are detected: a review of surveillance systems and outbreaks, Public Health Rep. 119 (September/October) (2004) 464–471.
[7] L. MacLehose, M. McKee, J. Weinberg, Responding to the challenge of communicable disease in Europe, Science 295 (5562) (March 2002) 2047–2050.
[8] N.H. Bean, S.M. Martin, Implementing a network for electronic surveillance reporting from public health reference laboratories: an international perspective, Emerg. Infect. Dis. 7 (5) (September–October 2001).
[9] Public Health Information Technology Functions and Specifications, Version 1.2. Centers for Disease Control, December 18, 2002.
[10] A. Campbell, The SARS Commission Interim Report: SARS and Public Health in Ontario, April 2004, pp. 97–100.
[11] J. Koskinen. Software Maintenance Costs, September 2003, http://www.cs.jyu.fi/~koskinen/smcosts.htm (last accessed February 4, 2005).
[12] S.R. Schach, Software Engineering, Richard D. Irwin, Inc./Aksen Associates, Inc., 1990.
[13] E. Maldoff, L. Lizotte-MacPherson. Canada Health Infoway/Vancouver Stakeholder Forum, March 2002, http://www.canadahealthinfoway.ca/pdf/CHI-Report-Vancouver.pdf (last accessed February 4, 2005).
[14] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, 1995.
[15] G.F. Luger, W.A. Stubblefield, Artificial Intelligence Structures and Strategies for Complex Problem Solving, second ed., The Benjamin/Cummings Publishing Company, Inc., 1993.
[16] J.A. Whittaker, J. Voas, Toward a More Reliable Theory of Software Reliability, IEEE Comput., December 2000, pp. 36–42.
[17] R.L. Freeman, Telecommunication System Engineering, third ed., John Wiley & Sons, Inc., 1996.
[18] S.A. Shields, T. Wong, J. Mann, A.M. Jolly, D. Haase, S. Mahaffey, S. Moses, M. Morin, D.M. Patrick, G. Predy, M. Rossi, D. Sutherland, Prevalence and correlates of chlamydia infection in Canadian street youth, J. Adolescent Health 34 (5) (May 2004) 384–390.
[19] M. Gully, S. Shields, C. Bowman, J. Phelan, T. Wong, Hepatitis B in Canadian street youth: trends in immunity between 1999 and 2003, in: Poster Presentation Presented at The International Society for Sexually Transmitted Diseases Research (ISSTDR), July 2005.
[20] M. Gully, S. Shields, C. Bowman, J. Phelan, T. Wong, STI and hepatitis C in Canadian street youth 1999–2003: What are the rates in this population? in: Poster Presentation Presented at the International Society for Sexually Transmitted Diseases Research (ISSTDR), July 2005.
[21] J. Phelan, R. Kropp, C. Bowman, S. Shields, T. Wong, Canadian street youth: sexual behaviours and self-perceived

risk, in: Canadian Public Health Association 95th Annual Conference, June 2004.
[22] J. Phelan, S.A. Shields, T. Wong, J. Mann, D.A. Haase, S. Mahaffey, S. Moses, M. Morin, D.M. Patrick, G. Predy, M. Rossi, W.D. Sutherland, Enhanced surveillance of Canadian street youth: an overview, in: Poster Presentation Presented at the British Association for Sexual Health and HIV Conference (BASHH), May 2004.
[23] F.-C. Tsui, J.U. Espino, V.M. Dato, P.H. Gesteland, J. Hutman, M.M. Wagner, Technical description of RODS: A real-time public health surveillance system J. Am. Med. Inform. Assoc. 10/5 (September/October) (2003) 399–408.
[24] W.W. Chapman, J.N. Dowling, M.M. Wagner, Fever detection from free-text clinical records for biosurveillance, J. Biomed. Inform. 37 (2004) 120–127.
[25] M.M. Wagner, J. Espino, F-C. Tsui, P. Gesteland, W. Chapman, O. Ivanov, A. Moore, W. Wong, J. Dowling, J. Hutman, Syndrome and Outbreak Detection Using Chief-Complaint Data—Experience of the Real-Time Outbreak and Disease Surveillance Project. Morbidity and Mortality Weekly Report, September 24, 2004, pp. 28–31.
[26] J.W. Buehler, R.S. Hopkins, J.M. Overhage, D.M. Sosin, V. Tong, Framework for Evaluating Public Health Surveillance Systems for Early Detection of Outbreaks. Morbidity and Mortality Weekly Report, May 7, 53 (RR05), 2004, pp. 1–11.
[27] M. Francis, D. Giokas, EHRS blueprint: an interoperable EHR framework (1.0), Canada Health Infoway (July 2003) 2–205.
[28] S.A. Hearne, M.A. Hamburg, L. Segal, SARS and its implications for U.S. public health policy: “we’ve been lucky”, Biosec. Bioterrorism: Biodefense Strategy Pract. Sci. 2 (2) (2004) 127–131.