From Passive to Active Electronic Healthcare Records

J. Bisbal1, J. Grimson2, W. Grimson3, D. Berry4, L. Hederman5
1Department of Computer Science, Trinity College Dublin, Ireland
2Centre for Health Informatics, Trinity College Dublin, Ireland
3Department of Electrical Engineering, Dublin Institute of Technology, Ireland
4Department of Electrical Engineering, Dublin Institute of Technology, Ireland
5Department of Computer Science, Trinity College Dublin, Ireland

1. Introduction

Healthcare generates large volumes of data at multiple locations (hospitals, laboratories, clinics, etc) for a wide variety of users. Caregivers often make their decisions based on this multi-sourced information, and may require access to highly heterogeneous and autonomous legacy systems containing relevant patient data. Integrating all the information to provide appropriate patient care poses organisational, technical and even political challenges. The Electronic Healthcare Record (EHCR) is central to the solution to this challenge.

Numerous approaches have been proposed as to how an EHCR must be built [1]. Some authors propose a static data model [2] and focus on a specific domain [3]. The Synapses paradigm [4], in contrast, allows healthcare professionals to structure the record to meet the specific requirements of an individual clinical domain.

Synapses is specifically aimed at supporting shared or integrated care which depends on the ability to share information between care providers simply and securely both within a single healthcare organisation and between different institutions. Synapses was initially based on a client-server architecture where clients send requests for clinical data to the Synapses server [5]. The server in turn is responsible for retrieving the required data from feeder systems (the data sources which store clinical data such as, for example, Laboratory or Hospital Information Systems) and shaping this data according to the client’s requirements (see Section 3). This represents a pull [6] approach to data dissemination – the Synapses server only sends data to a client in response to a specific request from the client. Presenting the client with integrated data in the form of a record potentially offers significant benefits to those responsible for delivering care to the patient. However, increasingly the record is being viewed as more than simply a passive repository of patient data. The real benefits of the EHCR are extended when it is directly linked to clinical guidelines and knowledge to provide real-time decision support [7–9].

An active Electronic Healthcare Record, which sends data to its clients as it becomes available, is central to the next generation of EHCRs [10].

This paper extends the Synapses paradigm with a push approach to data dissemination [11]. In this case the Synapses record server can initiate the transfer of information, in advance of any request from the client. Neither pull nor push approaches alone are considered to fit the requirements of any sophisticated dissemination-based information system [11], such as is required in healthcare. The work reported here resulted in a system which combines both pull and push technologies, potentially leading to a more flexible and scaleable approach.

The remainder of this paper is organised as follows. The next Section identifies the needs for active Electronic Healthcare Records, justifying the developments of this paper. Section 3 briefly describes the main concepts underlying the Synapses paradigm. Then the architecture of the current implementation of the Synapses server is analysed in Section 4. The design of the
active module added to Synapses in order to extend it with push technology is detailed in Section 5. Then Section 6 identifies some similarities between the design decisions made to implement this active functionality and those studied in the field of active databases. The final section suggests some future directions for this research.

2. Medical Applications for Data Push

The data push approach has been used in various contexts in healthcare, for example general medicine clinic [12] and glaucoma therapy [13] and is sometimes used to support protocols that react to changes in the record. Indeed it has been recognised that active push elements such as alarms and alerts modules are a useful enhancement to the more traditional ‘passive’ record system [14, 15].

In critical care environments it is desirable to get patient information to care-givers and decision support systems immediately it becomes available. Nevertheless, some hospitals use batch-driven results reporting systems that only return test results to the care-givers two or three times a day. Other systems are based on request-retrieve approach to data acquisition where information must be requested before it is transferred. While these approaches have their merits and are appropriate for routine results reporting, push technology is a more appropriate delivery mechanism for time critical parts of the electronic health record.

The point-of-care industry consortium, POCIC, has produced a message-based open specification for passing data from critical care instruments to data repositories [16]. This is a promising initiative and will make it possible for such information to be incorporated in real-time into the EHCR. Another useful initiative is the Medical Information Bus which provides “vendor-independent interconnection and interoperability of medical devices and computer systems” [17]. Once information has been updated from primary data sources to an information repository, the EHCR server will be in a position to immediately push this information to clinician or nurse workstations or wherever it is needed.

The active record approach also paves the way for real-time decision support functionality. The guidelines and protocols can either be linked to the record at the server or in the client application.

3. The Synapses Approach to Electronic Healthcare Records

The Synapses software is a middleware component that federates multiple data sources used to populate the record, and it allows each client to shape the record according to its specific needs. This maximises flexibility and allows the content and presentation of the information in the record to be tailored to meet clinical needs. This is achieved by the use of the three concepts that form the basis for the Synapses paradigm, namely the Synapses Object Model (SynOM), the Synapses Object Dictionary (SynOD), and the Record.

- SynOM: the object model used in Synapses defines a set of ‘building blocks’ and association rules that can be used to specify the particular shape of a patient record [4, 5]. Briefly, the building blocks in the SynOM, also called record components, can be classified as those used to describe the structure of the record, and those used to describe actual patient data. The former are referred to as record item complexes (RICs), and the latter as record items (RIs). A record component particularly relevant to the purposes of this paper (see Section 4) is the so-called communication record item complex or ComRIC, which describes a set of record items that must be kept together (to preserve meaning) when information is communicated using Synapses [5].

- SynOD: the specific record shape or template that a healthcare professional requires is defined by using the building blocks and aggregation rules of the SynOM. A SynOD does not contain actual patient data, but information about how to retrieve relevant data that must be used to populate the record, and how to organise this data according to the clinician’s requirements.

- Record: actual patient data can be retrieved by the Synapses server according to a particular template defined in the SynOD. Based on a template definition, the server can access relevant clinical data from the feeder systems as specified in the SynOD, and shape it accordingly.

Synapses has been validated in several clinical scenarios, with broad coverage of clinical domains [5]. The verification sites were the following: St. James’s Hospital, Dublin (Intensive Care), Royal Marsden Hospital NHS Trust (Oncology), Academic Medical Centre, Amsterdam (shared care – diabetes), Central Hospital of Akershus, Oslo (Internal medicine and general surgery) and Geneva Canton Hospital (General).

Synapses represents a flexible approach to delivering information without using a rigid data model, or being tightly related to a specific application domain (e.g. cardiology, diabetes). It also leverages the investment that healthcare organisations have made in their existing systems. Using the Synapses server the data used to populate the record remains in the current feeder systems, and it is the server that must adapt to retrieve this data. This is an important strength of Synapses if it is to become part of (healthcare) organisations that may not be in a position to migrate their systems to having full electronic record capability. Synapses assumes that feeder systems are legacy systems and cannot easily adapt to meet new requirements [18]. It is important to note that this assumption underlies the work reported in this paper, and has strongly influenced the design of the original functionality and now the active functionality of the Synapses server as described in Section 5. The next Section describes the current architecture of the Synapses server, and places the concepts analysed above in their context.
4. Synapses Server Architecture

Figure 1 shows the architecture of the current implementation of the Synapses server. It follows a three-tier architecture [19]: clients (e.g., web browser) request patient data according to a specified record template; then the Synapses server using the template’s definition decomposes this request and accesses all feeder systems that contain relevant data; the feeder systems represent the last tier of this architecture and are essentially data stores as far as Synapses is concerned. The feeder systems are considered to be autonomous and heterogeneous. For example, the current implementation of the Synapses server provides interfaces to access ODBC data sources (e.g., relational databases), image files, and an implementation of the HISA standard (http://www.cente251.org) called Distributed Health Environment (http://www.gesi.it/dhe/). Following this approach the Synapses server itself could be used as a feeder system for another Synapses server leading to a highly distributed and scalable architecture. This interaction between Synapses servers lies at the heart of the inter-institutional shared care proposed by the Synapses paradigm.

The Synapses server itself can also be divided into three layers. The first one is referred to in Figure 1 as the CORBA wrapper, and provides the distribution mechanism used by the server. It is based on the CORBA approach to building distributed applications [19]. Clients always connect to the Synapses server through this layer which provides them with network and implementation transparency [5]. The CORBA wrapper layer isolates the clients from changes in the implementation of the Synapses interface. Section 4.1 justifies the use of CORBA as the distribution mechanism for the current implementation of the Synapses server, and also analyses the use of other possible alternatives in the context of the work discussed in this paper.

The Synapses Server Kernel implements the Synapses interfaces, and is responsible for populating patient records according to the definition of the SynOD with patient data extracted from the connected feeder systems. Section 5 will briefly describe the Synapses interfaces.

Finally the Generic Adapter layer is responsible for interfacing with the numerous feeder systems that store clinical data needed to populate the Record. This component isolates the Synapses Kernel from the idiosyncrasies of individual feeder systems. Feeder systems could store data of diverse nature (demographics, images, etc.), and require very different connection/retrieval mechanisms (ODBC, binary file, proprietary). This functionality is encapsulated in the Generic Adapter. This module provides a uniform data access interface to all feeder systems, and transforms the resulting data into the appropriate record components (see Section 3) that will be used by the Server Kernel in order to populate the record [20].

Figure 1 also shows the extensions added to the Synapses architecture in order to extend the Synapses server with active functionality. Modules drawn in this figure with dotted lines denote new components. A different type of client is now required in order to allow the Synapses server to send a notification when new data becomes available. This notification is performed via a CORBA callback as will be described in Section 5.1. The Synapses Kernel has also been extended with an Active Server Component that is responsible for detecting relevant changes to a particular record component. The design of this module is described in Section 5.3.

4.1 Distribution Mechanism

As briefly outlined in the previous Section the distribution mechanism used in the current implementation of the Synapses server is OMG’s CORBA [19]. CORBA, as well as other competing technologies currently available, offer network as well as implementation transparency. In this context, the developers do not have to code the low-level details of the underlying computer networks. Similarly, the client applications are hidden from the implementation details of the services they request, which allows this implementation to change without affecting its clients.

There are other alternative distribution mechanisms which could have been used. Currently, both Sun Microsystem’s J2EE and Microsoft’s .NET are receiving much attention from the industrial sector, and are considered the most serious alternatives to
CORBA. Nonetheless, it could be claimed that CORBA is still the most generic and neutral alternative, particularly when compared to J2EE [21] and .NET [22]. In order to use J2EE the system must be developed or at least very strongly tight to Java technology, which may not be the technology chosen in many hospital IT departments. Likewise, .NET is only a viable alternative if the system is running on Microsoft’s Windows platform, which is not, to date, considered a robust enough environment for mission critical applications. In contrast, CORBA is programming language independent, and is available in a wide range of platforms.

The discussion above justifies the choice of CORBA as the distribution mechanism for the implementation of Synapses server described here. However, the principles on which the developments of this paper are based are also available in any of the alternative technologies discussed above. Particularly the active functionality, which forms the core of the work reported in this paper, relies strictly on the use of callbacks through the CORBA layer. This same concept is also readily available in J2EE and in .NET, as it is very commonly used when developing distributed applications. Therefore, although the discussion in this paper will focus on an implementation based on CORBA, the concepts described here can easily be applied to other technologies.

5. Push Technology in the Synapses Server

This Section describes the design of the active functionality as implemented in the current version of the Synapses server. Firstly, Section 5.1 outlines the mechanism used to notify a client of relevant changes to the data in the record. This requires extending the underlying CORBA layer that wraps the Synapses server. These changes are analysed in Section 5.2. Finally, Section 5.3 describes the extensions made to the Synapses Server Kernel in order to provide this functionality.

5.1 Active Mechanism

As outlined in Section 4 the distribution mechanism used by the Synapses server is that proposed by OMG’s CORBA [19]. According to this paradigm, a server must publish an interface with the services it provides, and clients request these services using this interface through the CORBA layer. The actual connection of client and server is transparent to both tiers and is the responsibility of the CORBA implementation in use.

The active mechanism has been implemented using another of CORBA’s features known as client callback. In order for the Synapses server to notify a client when relevant changes occur in the record, the client must first register with the server identifying what changes are relevant to it. Relevant changes are defined in terms of the contents of a particular ComRIC(s) (see Section 3) in the record(s) a client is interested in. The Server will monitor these ComRICs for changes and notify the client when their contents change. Recall from Section 3 that a ComRIC represents an atomic part of a patient’s record which must always be transmitted in its entirety. Therefore the ComRIC has been chosen as the unit of information over which the active record is defined.

Different active semantics have been implemented, depending on which kind of changes in the contents of a ComRIC (insertions, updates, etc.) a client considers relevant. Monitoring a ComRIC for any kind of change has high computational cost. A client can tune the way the ComRICs contents are monitored using domain specific information (at the ComRIC level) in order to reduce this complexity. For example, if the client knows that for a specific ComRIC the contents are sorted by date and that only insertions will occur, such as for example with laboratory test results, it can register this ComRIC with active_insert semantics so that the Synapses server only checks for new record components at the end of the contents of the ComRIC. This approach makes the active record more flexible, efficient, and scaleable. Similarly, the current implementation allows for both thin-client and fat-client architectures [19]. Upon registration, a client can specify whether the server must notify that a change has occurred in a particular ComRIC, without specific information about the actual change. This will lead to a fat-client architecture, as it is then the client’s responsibility to determine what are the new contents. On the other hand, the client may require the Synapses server to send the actual changes that have occurred in record. This leads to a thin-client approach. All these active semantics can be combined in different ComRICs for the same or for different patient’s records. See Section 5.3 for a detailed explanation of these semantics.

5.2 Extending the Synapses IDL

The Synapses server implements a CORBA interface designed to provide generic access to an electronic healthcare record. Figure 2 shows a UML class diagram for the Synapses server [24]. Refer to [5] for a detailed description, with data flow diagrams, sequence diagrams and use-cases, of these interfaces. Briefly, from a client’s point of view, the entry point to accessing information provided by the Synapses server is to bind to InterrogateInterface. Then it must use the service GetInterfaceRef() in order to establish a Session. Using the Session interface it can first retrieve all existing patient records with FindRecords() and then bind to one Record interface using the GetRecord() service. Once the client has accessed to a particular record it can retrieve its contents either as CORBA objects, using the ExternalObjectRetrieval interface, or as XML documents using the XmLEntityRetrieve interface. The description here will focus on the CORBA version of this interface; the XML version is analogous.
As described in Section 4 a Synapses-compliant healthcare record is shaped using the building blocks of the SynOM, namely FolderRICs, ContextRICs, and ComRICs. The ExternalObjectRetrieval interface allows the client to unfold the shape of the record by navigating through the tree structure of FolderRICs and ContextRICs using services RetrieveFolderRIC() and RetrieveContextRIC(). Actual patient data specified in a ComRIC can be retrieved using RetrieveComRIC().

All interfaces described above assume a pull approach to data dissemination, that is, data is sent to a client only when the client requests it. Therefore the Synapses server is referred to as being passive; it provides data only upon request.

In order to provide the clients of the Synapses server with active functionality, as described in the previous Section, this interface had to be extended so that:

1. Clients can register with the server for relevant changes in specific parts of a Record.
2. The Server can notify clients of such changes.

Interfaces shown shaded in Figure 2 represent the extensions required to the Synapses interface in order to allow the server to be active, that is, to send relevant data to a client when it becomes available, without waiting for the client to request it.

As shown in Figure 2, clients register with the server using the ExternalObjectRetrieval interface, as it provides a service called RegisterComRIC(). This interface is always associated with one particular Record object, therefore the actual patient data being registered for is implicit. Method RegisterComRIC() takes three parameters:

1. ComRIC number that identifies the ComRIC within the record.
2. Instance of interface ActiveObjectRetrieval, see Figure 2.
3. Active semantics for this particular active ComRIC.

Interface ActiveObjectRetrieval is used by the Synapses server to notify the client of relevant changes. Before registration, clients must instantiate an object of this class and then pass it as a parameter to RegisterComRIC(). When a relevant change occurs, the server will call a method in this object (e.g., either ComRICChange() or ComRICInsertion()). This call is passed back to the client via the CORBA wrapper and it gets executed at the client side. This mechanism is referred to as a callback in CORBA terminology\(^2\) [25], and it is transparent to the (client and server) applications. Which of these two methods, ComRICChange() or ComRICInsertion(), the server calls depends on the active semantics being registered for and is detailed in the next Section.

\(^2\) As discussed in Section 4.1, this is a concept commonly used in the context of distributed applications development, and is also available in those technologies considered as alternatives to CORBA, like for example J2EE and .NET.

### 5.3 Design of the Active Server Component

Figure 3 shows the design of the active server component shown in Figure 1 used to implement the active functionality for the Synapses server as described in the previous sections.

Class ActiveRecordManager is used to wrap all this functionality, and is also the container of the set of active ComRICs (Section 5.1) the Synapses server manages at any given time. As described in Section 5.2, active functionality is provided at the ComRIC level, that is, each client registers with the server the set of ComRICs it wishes to make active. The server will monitor the contents of these ComRICs, and when changes occur it will notify the appropriate clients.

The container class, ActiveRecordManager, offers methods to add and remove active ComRICs. These methods are used by
the implementation of services RegisterComRIC() and RemoveActiveComRIC() published by the extended IDL described in Section 5.2. Whenever an active ComRIC is registered, a new instance of class ActiveComRIC is created and it is added to the collection of active ComRICs maintained by the server. If two different clients register for the same ComRIC, two different instances of ActiveComRIC will be created. This simplifies the case in which two clients register for the same ComRIC at different times, and therefore the appropriate notifications may be different for each of them (see Section 6).

Method checkChanges() in class ActiveRecordManager is executed by an independent thread. At predefined time intervals, this execution thread goes through the collection of ActiveComRICs currently registered and calls method checkChange() in each of them. This method retrieves the current contents of this ComRIC which, as described in Section 3, contains the patient data. The contents this ComRIC had previously, in the last execution of method checkChanges(), would have been stored in data member called snapshot in Figure 3. Then method compareComRICs() is used to compare these two contents and check whether any change has occurred. If so, the set of new record components is returned.

If a change has been detected in the contents of a ComRIC, the corresponding client must be notified. Data member notify_to contains a reference to the client, which has been passed to the server when registering a ComRIC (see Section 5.2). This is a reference of type ActiveObjectRetrieval (Figure 2), and thus two services are available, namely ComRICChange or ComRICInsertion:
- ComRICChange: this method is called when a ComRIC has been registered as having active change semantics. This is used to implement a fat-client architecture, as referred to in Section 5.1. With this active semantics, the server will only count how many record components (e.g., tuples if a ComRIC represents data coming from a relational database) this active ComRIC contains. If this number changes, the client is notified. However, only the ComRIC ID is passed back to the client, not its contents. It is the client’s responsibility to find out what has been changed in that ComRIC, if this information is necessary.
- ComRICInsert: this method is called in case of active insert semantics. The Synapses server will send back to the client the sets of new record components as they become available in the ComRIC. As described above, this requires the contents of the ComRIC to be compared at predefined time intervals, using method compareComRICs(). If the client knows that new record components are added only at the end (or the beginning) of a ComRIC, then the identification of these new record components can be done efficiently. This is the default behaviour of compareComRICs(). If this is not a valid assumption, and new record components are added to a ComRIC in any order, then active update semantics should be used upon registration of the ComRIC. With this semantics, the Synapses server will also notify clients by calling method ComRICInsert(). However, the detection of changes in the record is computationally more thorough.

5.4 Exception Support

Due to the computational cost involved in monitoring active ComRICs for possible changes in their contents, special attention has been paid to detecting active ComRICs that no longer need to be considered. This is the case when the associated client process has died. Two cases are considered:
- As described in Section 5.3, when a change is detected in the contents of an active ComRIC, method checkChange() of class ActiveComRIC will notify the appropriate client. If the client has died, the server will at this stage remove this instance of ActiveComRIC from the collection of active ComRICs under consideration.
- It is possible that no changes are detected in a particular active ComRIC for a large number of successive calls to checkChange(). There is a possibility that these checks are unnecessary because the client may have died, even though this has not yet been detected because no notification has been sent to the client. For this reason, the Synapses server will, after a predefined number of calls to checkChange() without changes detected, contact the client to ensure that it is still alive.

If the collection of instances of ActiveComRIC maintained in class ActiveRecordManager becomes empty, the dedicated thread that executes method checkChanges() will terminate. It will be started again only when a new active ComRIC is registered. Therefore the Active Server Component (Figure 1) does not pose any overhead to the Synapses server when no active functionality is required.

---

3 In this case, the comparison has linear time complexity, $O(n)$ [23], with the number of record components in the ComRIC.

4 As described in footnote 1, this case has quadratic complexity.
6. Active EHCR and Active Databases

Some of the design decisions made to extend the Synapses server with active functionality can be compared to design alternatives used when building an active database system. This section places the design of an active EHCR described here in the context of the general framework for comparing active database systems reported in [26]. Most of the decisions made when extending the Synapses server have been dictated by the fact that the feeder systems are considered autonomous (legacy) systems, and cannot be expected to adapt in order to meet the requirements of the Synapses server. For example, requiring all feeder systems to provide, themselves, active capabilities was not an acceptable option.

In the active databases framework presented in [26] the knowledge model is based on so-called ECA-rules, which are constructed from up to three components: an event, a condition and an action. The event describes what makes a rule relevant. The condition specifies the context in which the rule may be fired. And the action describes the task to be performed if the event has occurred and the condition is satisfied. A large number of approaches have been proposed to support ECA-rules, leading to widely different functionalities depending on how each of the three components are added to the system.

- Event: the nature of the event and how it can be detected depends largely on the source of the event. Alternatives include structure operation (e.g. inserting a tuple), transaction (e.g. commit), exception, user-defined, external, clock, etc. In case of the active EHCR described here, the event is always raised at some point in time, i.e. clock event detection. As described in Section 4.3, method checkChanges() in class ActiveRecordManager monitors all active ComRICs at predefined time intervals.
- Condition: the conditions for the events that are used in the design of the active EHCR analysed above are always concerned with whether a given active ComRIC contains any new data. Its context, that is, the settings in which the condition is evaluated, is determined by the contents of the ComRIC at the point in time that the event occurred as well as the contents of that same ComRIC at the last point in time where its contents were monitored.
- Action: the actions associated with the events described here are always the notification to the client of a relevant change in the contents of the record. Several different notifications are possible, depending on the active semantics as described above, and they are likely to be extended in the future.

The comparison of the active EHCR functionality with the active database research area is useful to identify the issues addressed, the assumptions made and areas where future work may be necessary. However, the scope and requirements are more limited in the case of the active EHCR. For example, the active EHCR functionality can be identified with a set of rules that compare the contents of a set of ComRICs and notify the user of changes. In this scenario there would not be interference between rules. This simplifies the execution model used in a general active database system [26], in which rules’ actions may have an impact on each others events or conditions, and therefore the rule execution order may determine the final database state. In case of the execution model described in Section 5.2, the order in which method checkChange() of class ActiveComRIC is called for each registered active ComRIC is not relevant (from the Synapses server point of view).

A significant restriction placed to the active EHCR implementation relates to the fact that the autonomy of all feeder systems must be preserved. This forces a Net-Effect Policy approach [26], that is, if several changes occur during two time events, all changes will be considered together. Feeder systems cannot be enforced to notify the Synapses server of individual changes. This restriction has also forced what is referred to as a Layered Architecture [26] approach, by which the active component is added on top of the underlying (unchanged) passive feeder system.

Clinical guidelines have been shown to be most effective when they are automatically triggered in response to patient-specific prompts. Thus changes to the record (directly or via feeder systems) notified using the active functionality described in this paper, can cause guidelines rules to fire which notify clients of further actions required.

7. Conclusions and Future Work

This paper has presented a general approach to adding active functionality to a Synapses-based electronic healthcare record system. The final goal is to make the system pro-active so that it notifies clients of relevant changes in the underlying data sources (feeder systems) when they become available, as opposed to providing patient data only when clients request it.

No assumptions have been made regarding the active capabilities of feeder systems. The Synapses server is responsible for retrieving relevant data in order to detect whether changes have occurred. This approach ensures that this active functionality can be used in any environments where the Synapses server itself has been installed, without additional requirements to...
the underlying IT infrastructure. However, this approach also has high memory and computational requirements. A more sophisticated approach needs to be investigated in cases where a feeder system itself provides active capabilities, in which case part of the functionality described in this paper would be shared between the Synapses server and the feeder system. This may prove challenging as record components (ComRICs) requested by clients may federate data which spans across several feeder systems, some of which may have active capabilities and others may not.

The mechanism described in Section 5.3 to detect changes occurred in the record has high memory requirements as it stores the latest two versions of the contents of each active ComRIC. Materialising the record (i.e. storing its current contents in a local database) could be an alternative in order to reduce the current memory usage. There are other advantages associated with caching the records locally in terms of increased efficiency especially where the data comes from remote feeders (e.g. a different healthcare institution) and is accessed frequently. In such a scenario, records created ‘on the fly’ by the Synapses server could be stored locally in a database/warehouse. Also, as outlined in Section 5.3, the current approach should be optimised for the case in which several clients register for the same active ComRIC. Since this situation could raise inconsistent notifications to clients, they are currently treated as being independent active ComRICs.

The Synapses server that resulted from the extensions reported in this paper allows for both client pull and server push data dissemination strategies. Future work will focus on extending the system towards the other two characteristics considered as central in any dissemination-based information system (DBIS) [11,27]: Aperiodic vs. Periodic notifications, and Unicast vs. 1-to-N notifications. In particular, multicasting will become increasingly important in the context of shared care, where several clinicians may need to be notified of relevant changes to patient records (e.g. new laboratory results). The current system would perform multiple individual (unicast) notifications. This approach does not scale in case of high network traffic and overloaded servers. Population-based Electronic Healthcare Records [10] will require broadcasting as part of their data dissemination strategies.

The current approach has also assumed that clinical data is not deleted or updated from an electronic patient record. This is in line with best practice and medico-legal requirements that state that erroneous or updated data should not be removed or changed from a record but rather that new record components should be added to describe those corrections. Therefore the assumptions made here do not seem to pose a limitation in practice.

The work described in this paper is a necessary pre-requisite to the broader objective of linking clinical guidelines and EHCRRs to provide real-time decision support and is being undertaken as part of a wider research initiative in Health Informatics, MediLink [28, 29]. MediLink is a collaborative programme which is developing an open and generic approach to the integration of the active EHCRR with clinical guidelines, case bases, workflow, and modelling and simulation tools.

Acknowledgements

The authors gratefully acknowledge the contribution of members of the Dublin Health Informatics Group and of the Synapses and MediLink consortia to the work discussed in this paper. This research has been supported by the European Commission (Synapses HC1046) and the Irish Higher Education Authority (MediLink, PRTLI Cycle 2).

References


Correspondence to:
Professor Jane Grimson
Vice-Provost
Trinity College
Dublin 2
Ireland
E-mail: jane.grimson@tcd.ie

Methods Inf Med 5/2003