LiSA - Live System Analysis: A robust approach for the real time monitoring of clinical environments

Issom, David-Zacharie; Lovis, Christian

Swiss Medical Informatics
SGMI SSIM SSMI
Schweizerische Gesellschaft für Medizinische Informatik
Société Suisse d'Informatique Médicale
Società Svizzera d'Informatica Medica
Swiss Society for Medical Informatics

LiSA – Live System Analysis: a robust approach for the real time monitoring of clinical environments

David-Zacharie Issom, Christian Lovis

University Hospitals of Geneva, Division of Medical Information Sciences, Switzerland

Abstract

The use of data produced in the University Hospitals of Geneva for steering and governance is becoming increasingly important. In this regard, many tools have been developed, both for aggregating and consolidating storage and also in terms of analysing Big Data. However, these tools are not generally designed to provide a real-time analysis. Indeed, there is an increasing amount of data available and it is easy to get lost. The potential uses of real-time monitors are numerous and can greatly increase the responsiveness of the institution in various situations by providing relevant data needed in that situation. The LiSA project aims to develop the conceptual and technical architecture for building real-time monitors, objects potentially useful in critical situations. A robust architecture is necessary to build a nearly real-time system. Management of alerts and events in a healthcare environment is so complex and scattered that it has become necessary to centralise. In software engineering, one of the most reliable architectures for such systems is the Publish-Subscribe Design Pattern. This architecture has been implemented in the project in order to give access to any kind of event in the Clinical Information System (CIS). Once data are processed and organised, a data stream that can be read by humans is sent to a web based monitoring interface. This new versatile monitoring model could lead to better and clearer management. We found that this system can stimulate managers to access the CIS events by creating meaningful indicators. It could help governance when it is necessary to obtain some information immediately, or to create statistical charts or gauges.

Key words: *pervasive healthcare, alert management, dashboard, monitoring, real time monitoring, clinical environment, healthcare environment, events, EHR alerts, clinical decision making, clinical decision support systems, Big Data*

Résumé

L'utilisation des données produites du HUG à des fins de pilotage et de gouvernance prend une place croissante. A cet égard, de nombreux outils ont été mis en place, tant en matière de stockage agrégé et consolidé, qu'en matière de capacité analytique de ces données. Ces outils ouvrent encore d'inestimables perspectives et ne sont encore qu'à l'aube de leurs potentiels. Aux HUG comme dans toutes les autres institutions, ces outils reposent sur une stratégie entièrement appuyée sur l'existence d'entrepôts de données institutionnels. Le principal défi, mais également l'avantage de ces entrepôts réside dans la représentation commune et standardisée de multiples sources de données, logistiques, cliniques, financières, etc. Le projet LISA vise à développer l'architecture conceptuelle et technique permettant la construction de moniteurs en temps réel potentiellement utiles dans les situations critiques. Une architecture robuste est nécessaire pour construire un système fonctionnant en temps réel. De plus la gestion des alertes et des événements dans les environnements de soins de santé est si complexe et dispersée qu'il devient nécessaire de la centraliser. En génie logiciel, l'une des architectures les plus fiables pour construire de tels systèmes est le *Design Pattern* Publish – Subscribe. Cette architecture a été mise en œuvre dans le projet afin de rendre l'échange d'informations fiable et instantané. Une fois que les données sont traitées et organisées en un flux de données lisibles puis ce flux est visualisé sur une interface Web. Ce nouveau modèle de surveillance polyvalent peut conduire à une meilleure gestion des institutions et peut stimuler la gouvernance à créer les indicateurs dont elle estime avoir besoin immédiatement, mais aussi à créer des graphiques ou des jauges statistiques.

Introduction

The use of data produced in the Geneva University Hospitals (HUG) for management and governance is becoming increasingly important. Making as much information as possible available is an important goal of Clinical Information Systems (CIS). Consequently, integration in healthcare facilities and interoperability between numerous subsystems are important priorities. However, this evolution has not always been positive [1–3]. The increase in available information must not be achieved at the cost of quality and pertinence; otherwise it will lead to regrettable consequences [4–7]. Disorganised data can lead to wrong decisions. In this regard, many tools have been developed for aggregating and consolidating storage, mainly for the data analysis capacity. These tools open invaluable perspectives and are still at the dawn of their potential. In the HUG, as in many other institutions, these tools rely on a strategy

chosen to support fully the existence of institutional repositories or data warehouses.
The main challenge, but also a benefit, of these warehouses is the common presentation and standardisation of multiple sources of data: logistic, clinical, financial, etc.
However, these tools are not generally designed to be real-time analytical tools. For example, analytical databases are separated from transactional databases owing to architectural aspects of design, structure and criticality. Thus, for instance, data warehouses are, at best, updated with a frequency of J-1, specifically, data produced up to the day before [1].
The potential uses of real-time monitors are numerous and can greatly increase the responsiveness of the institution in certain situations, including bed management, human resources or infections, to take only three examples, or situations such as a fire, a disaster plan or an epidemic.

Background

The University Hospitals of Geneva (HUG) is the major care-providing consortium of public and teaching hospitals in Switzerland. It covers primary, secondary, tertiary and ambulatory care. It has about 9,000 FTEs. HUG handles about 1 million outpatient visits and 50,000 inpatients a year. HUG uses a clinical information system developed in-house. This system tightly integrates commercial systems covering all clinics and care. HUG comprises 8 hospitals in 4 different campuses in the Canton of Geneva.

Foundation

The LiSA project aims to develop the conceptual and technical system for building real-time monitors, objects potentially useful in critical situations. To do this, the project will be built on a fundamental characteristic of the existing clinical information system: message-oriented middleware.
The clinical information system (CIS) of the HUG is based on architecture components, and service-oriented messages. The organisational component of the SIC means that each feature was developed as a small independent program named component. A business entity, such as prescribing, will consist of many specific components and will use other components, such as the laboratory.
To allow these components to communicate autonomously, components implement two communication tools: services and messages. The services of a component can be called by another component that needs to perform a function. Thus, a component of prescription may call a laboratory component for tests of renal function, and propose an appropriate dose. Moreover, when a component performs an action, it will send a message. Another component may receive this message and be informed of the action without having to call the initial component. For instance, in HUG, each change in the clinic route, specifically a patient movement, is managed by a component that sends automatically a message to any service interested.

Alerts and events in Healthcare requires centralisation

Nowadays, CIS share a large amount of patient data. Most are updates of current clinic route, clinical pathway, drug prescriptions, laboratory results, schedules and patient appointments, documents and discharge letters, diagnoses and operations, but some are administrative data.
They come as real-time messages and can be complex or simple. The system we have to create must be at least as effective. It is, therefore, necessary to provide a real-time view and analysis of this in-coming information. Availability 24/7 is a requirement. Consequently, some technical considerations should be taken into account. The system must not affect the upstream data, but only connect. Indeed, in our experience every day, and often at the same time of day, the system handles peaks of data. Clinical activities are critical so we cannot afford to alter the functioning of the CIS.
The CIS, as said before, contains a lot of information from different sources. Several component producers of data flows are involved so notifications are heterogeneous. The source of these messages can come not only from actions executed by doctors, nurses or others stakeholders, but also from devices such as card readers, as well as logistical information such as the activity of a doctor in a service, the availability of lift, the open/closed state of a door, and also radiology images. These data are exchanged wrapped in XML messages based on a Petri net of different types [8] *(lab.result.*, user.login, task.refresh, patient.criticaldata.update, etc.)*. This information is then embedded in a cloud of distributed and heterogeneous data and received by every component concerned. To handle this is complex and, therefore, it is necessary to unify and centralise the management of these data. The idea here is to use the existing system of message exchanges to give users a tool with which they can be informed of alerts and events that interest them.

A real-time connectivity to any data source

As the system will need to read large amounts of data of different types, we must decide on some features in order to develop the conceptual and technical basis:
- The system should interpret data independently of their type – use of an universal data format (XML)
- The system has to receive data in real-time and provide a tool for analytical instruments for project steering, so data should be chartable.
- There are a large number of notifications generated every second and they should be immediately treated. There may be a delay or loss in the treatment, but it should never stop the upstream components. The system should be able to operate continuously without slowing down the operation of the CIS. In cases of extreme slowness or problems, notifications can be untreated but the system must be notified or, at least, be aware.

Essential steps in the data processing

The system will handle the data in several ways. For greater clarity and precision, these steps have been clearly divided [9]:

– *Reading:* The first step is to read the data from the CIS upstream. For now, the program connects to the main flow of notifications. A connection bridge has been developed simply to choose a streaming feed and receive data. This components allows the user to choose a bridge, start or stop reading, limit the amount of incoming data, and choose a frequency interval for the data reception (for example, data by second, data by minute, etc.). The system must react such that when there is an error in the reading process notifications (incomplete, illegible or slow) the system simply ignores it and proceeds to the next.

– *Purging:* The purpose of this phase is to consider only the properly constructed notifications, and not treat invalid or unintelligible notifications. There is no specific treatment of errors: erroneous notifications are simply ignored. This makes the system available at any time and avoids problems going back to the previous process. The purge system must also send an acknowledgment to show which notification has been served and when.

– *Pump:* The system produces a stream of data and sends it to the next component. It must be able to pump the notifications received by the reading stream and transfer them to an external system. The notification must be standardised and easily treatable by the client system. The notification must be sent immediately upon receipt.

– *Routing:* The system receives the pumped notifications and offers referral opportunities to route them by category, arrival date, size of the message or others detailed properties. The referral notifications are redirected to queues (database) which are then exploited. In the case of error or low reactivity, the treatment is stopped. The system sends the current notification to a removal waiting queue and the following notification is processed [10].

– *Cleaning:* The system changes the notifications to keep only information relevant for a human. The semantics are not changed but technical data are deleted. In the case of error or low reactivity, the treatment is stopped, the current notification is also queued and the following notification is processed. The goal is to prepare notifications. Thus the important information is highlighted without unnecessary information and the size of messages is optimised.

– *Grouping:* The system has the ability to create clusters of notifications based on selected criteria (date, keywords in posts, or properties, type (default) or category, key/value). This aggregation step consolidates the received data and creates clusters by message type [11].

Real-time architecture

A robust architecture is necessary to build a nearly real-time system. Moreover, the system should be able to receive a significant amount of data from several sources. In software engineering, one of the most reliable architectures for such systems is the Publish/Subscribe [12] Design Pattern [13]. The philosophy of this pattern is to add an intermediate layer between the components. The purpose is to loosen coupling, decrease the direct dependency and share responsibilities. The component goes through the intermediary layer to subscribe to a producer of interest. This additional step is responsible for informing the subscribed components of any change on any data stream.

This architecture is implemented in LiSA in order to give availability to any kind of new event for receivers, as soon as data are processed and organised. The stream is sent to a sinking step that is the content provider or publisher. Thereby, services, applications or components interested in the data subscribe to this component and will receive the update in real-time. Once the system receives notifications, it puts them in a cache and empties the queue continuously. The system can create real-time streams of messages containing a definable number of notifications of various selected kinds. Notifications are afterwards sent to the content management, which avoids problems going back to the previous process that is the display interface. In the case of low reactivity, the treatment is stopped, the current notification is sent to a waiting queue for removal and the following notification is processed [14].

The following figure shows the conceptual model of the implementation.

Realisation and tests

After each step, some acceptance tests were made. This was to ensure that reports were well received and to strengthen the robustness of the software. The step was validated only if data were read and if each notification could

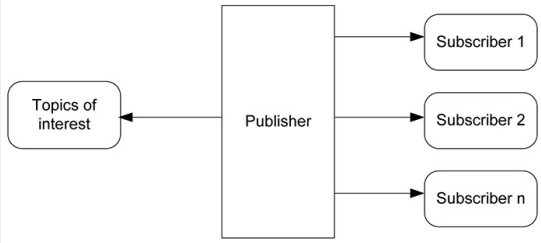

Figure 1

The Publish–Subscribe pattern.

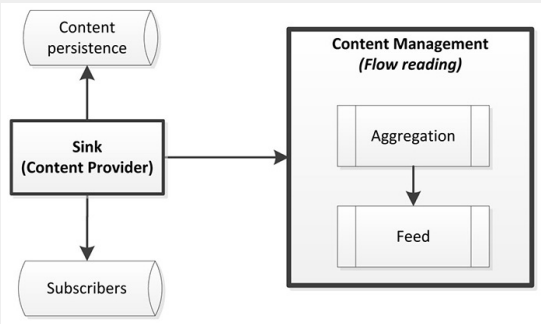

Figure 2

The implemented Publish–Subscribe architecture.

be broken down and shown separately with all its details, without any loss of information. The following figure illustrates the entire conceptual model.

A Web-based system for clinical decision support

At present, the interface is still a prototype, but it gives users an overview of notifications. It allows users to group them before a graphical visualisation [15] but does not yet allow them to choose different criteria to display (arrival date, keywords or categories).

Results and discussion

The establishment of such a system will enable the development of applications with strong benefits for managing a variety of critical situations, either locally or globally critical to the institution, and improve decision-making capacity and responsiveness in these situations. Moreover, the Publish-Subscribe design pattern is an efficient mechanism for those purposes.

Critical events and alerts can be widely used in a healthcare environment to communicate with management or notify caregivers.

This new monitoring model could also lead to better and clearer management. Indeed, the versatility of the system can stimulate managers to access CIS events by creating meaningful indicators. It could help them when they need to obtain some information immediately, create statistical charts or gauges.

However, several steps need to be finalised. It is still necessary to facilitate the choice of different data sources, to finish the web interface and consolidate the standardisation of XML flows and to make the connector bridge totally independent of upstream data.

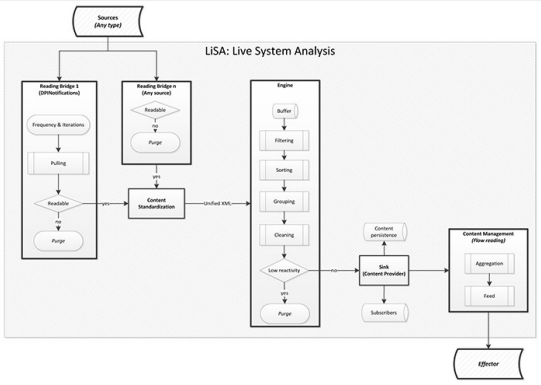

Figure 3

Conceptual model.

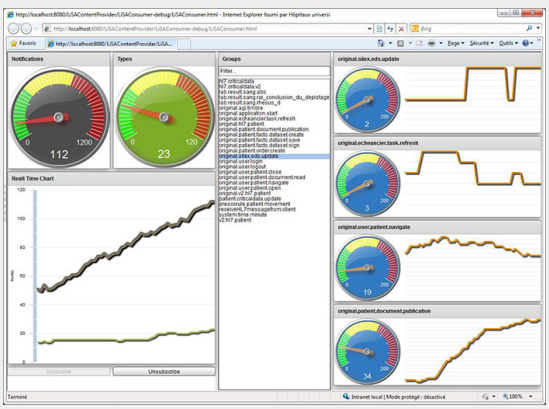

Figure 4

Web interface prototype.

Acknowledgments

Christian Lovis and many individuals of the Division of Medical Information Science of the HUG have contributed to the work. They are warmly acknowledged for their participation to the software.

Correspondence:
David-Zacharie Issom
University Hospitals of Geneva
Division of Medical Information Sciences
Rue Gabrielle-Perret-Gentil 4
CH-1211 Geneva 14
david.issom[at]hcuge.ch

References

1 Hirschtick RE. A piece of my mind. Copy-and-paste. JAMA. 2006;295(20):2335–6.

2 Hammond KW, Helbig ST, Benson CC. Are electronic medical records trustworthy? Observations on copying, pasting and duplication. Brathwaite-Sketoe BM. 2003:269–73.

3 O'Donnell HC, Kaushal R, Barrón Y, Callahan MA, Adelman RD, Siegler EL. Physicians' attitudes towards copy and pasting in electronic note writing. J Gen Intern Med. 2009;24(1):63–8.

4 Cicourel AV. Cognitive overload and communication in two healthcare settings. Commun Med. 2004;1(1):35–43.

5 Embi PJ, Yackel TR, Logan JR, Bowen JL, Cooney TG, Gorman PN. Impacts of computerized physician documentation in a teaching hospital: perceptions of faculty and resident physicians. J Am Med Inform Assoc. 2004;11(4):300–9.

6 Ash JS, Berg M, Coiera EJ. Some unintended consequences of information technology in health care: the nature of patient care information system-related errors. Am Med Inform Assoc. 2004;11(2):104–12.

7 Samaritan GA. Standard of care deviation results in patient's death. J Med Assoc Ga. 2010;99(2):32–3.

8 Sievering J, Lovis C, Geissbuhler A. Implementation of a publication-subscription environment within a multi-agents paradigm. Stud Health Technol Inform. 2002;90:310–5.

9 Waitman Lemuel R, et al. Adopting Real-Time Surveillance Dashboards as a Component of an Enterprisewide Medication Safety Strategy. Jt Comm J Qual Patient Saf. 2011.

10 Dugerdil Ph., Sennhauser D. Dynamic Decision Tree for Legacy Use-Case Recovery. 28th ACM Symposium On Applied Computing. 2013.

11 Dugerdil Ph., Jossi S. Computing Dynamic Clusters. 2nd ACM ISEC 2009, Pune, India, February 23–26, 2009.

12 Geissbühler A et al. Design of a General Clinical Notification System Based on the Publish-Subscribe Paradigm. Am Med Inform Assoc. 1997.

13 Rafe V, Hajvali M. Designing an architectural style for pervasive healthcare systems. J Med Syst. 2013;37(2):9927. doi: 10.1007/s10916-013-9927-6.

14 Donnelly N, et al. Development of a ubiquitous clinical monitoring solution to improve patient safety and outcomes. Conf Proc IEEE Eng Med Biol Soc. 2012;2012:6068–73.

15 Wicht A, Wetter T, Klein U. A web-based system for clinical decision support and knowledge maintenance for deterioration monitoring of

hemato-oncological patients. Comput Methods Programs Biomed. 2013;111(1):26-32. doi: 10.1016/j.cmpb.2013.02.007.

Figures (large format)

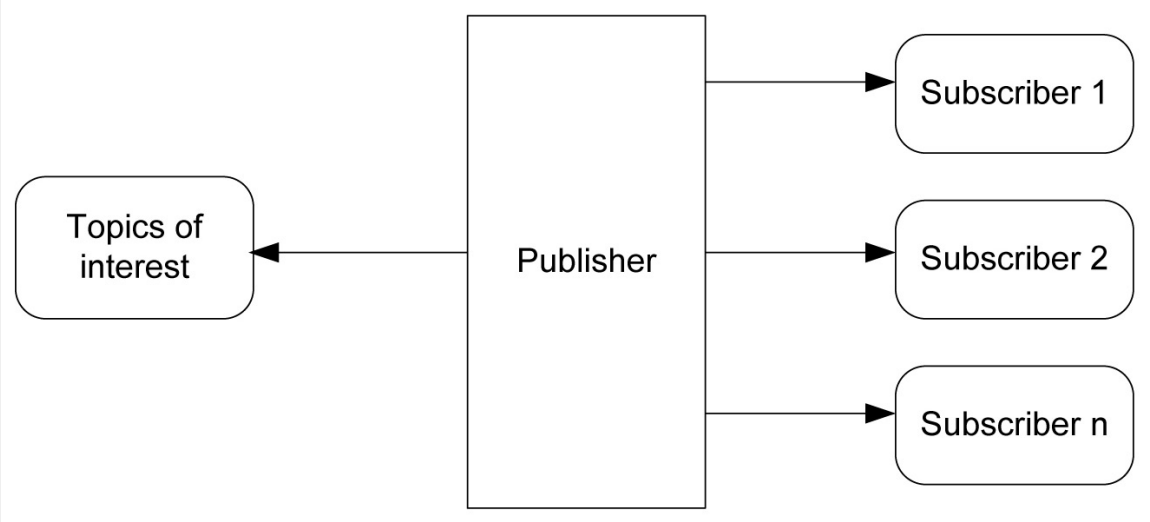

Figure 1

The Publish–Subscribe pattern.

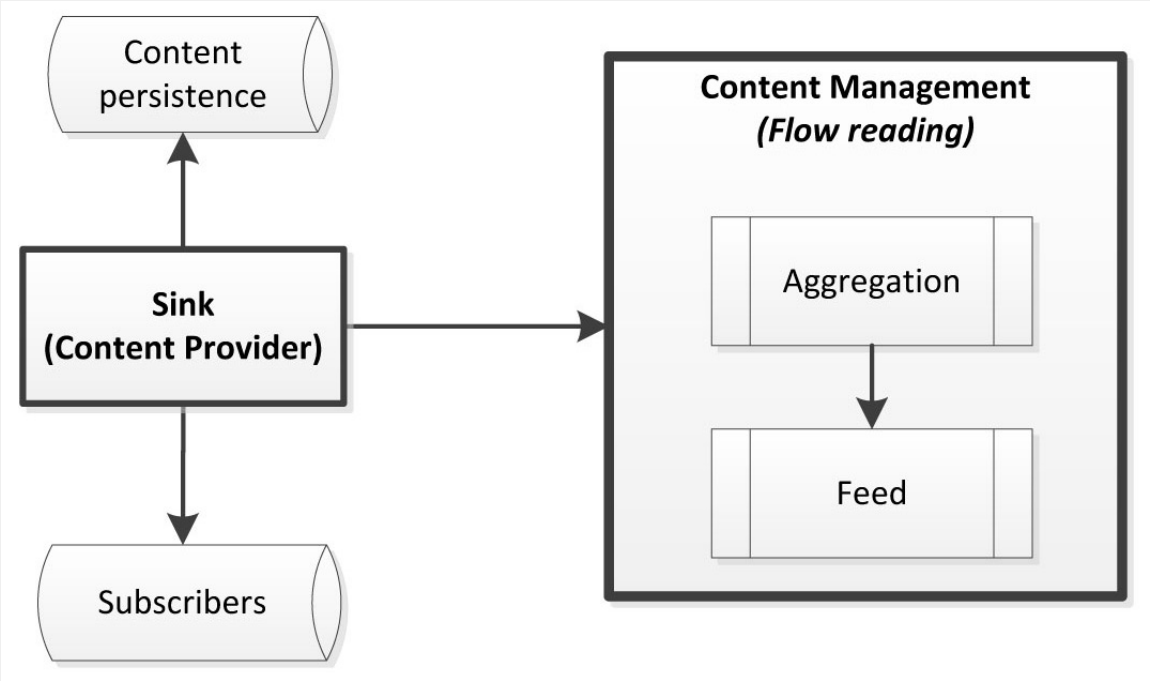

Figure 2

The implemented Publish–Subscribe architecture.

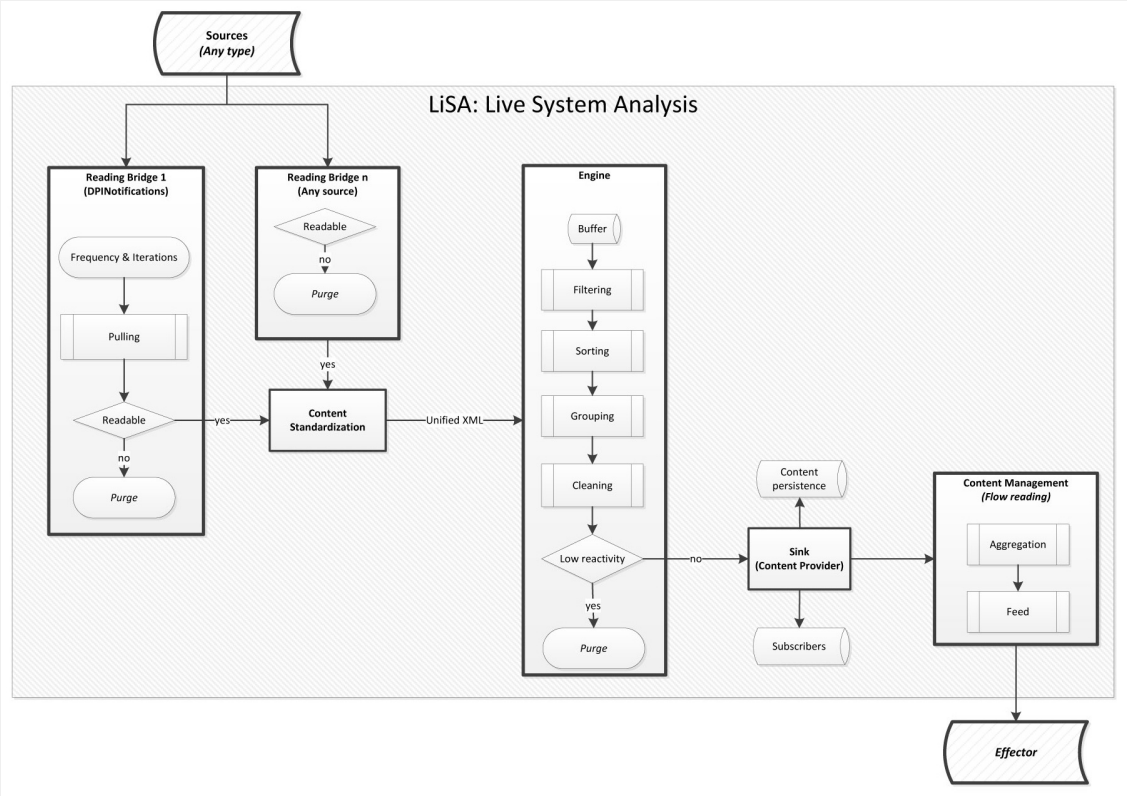

Figure 3

Conceptual model.

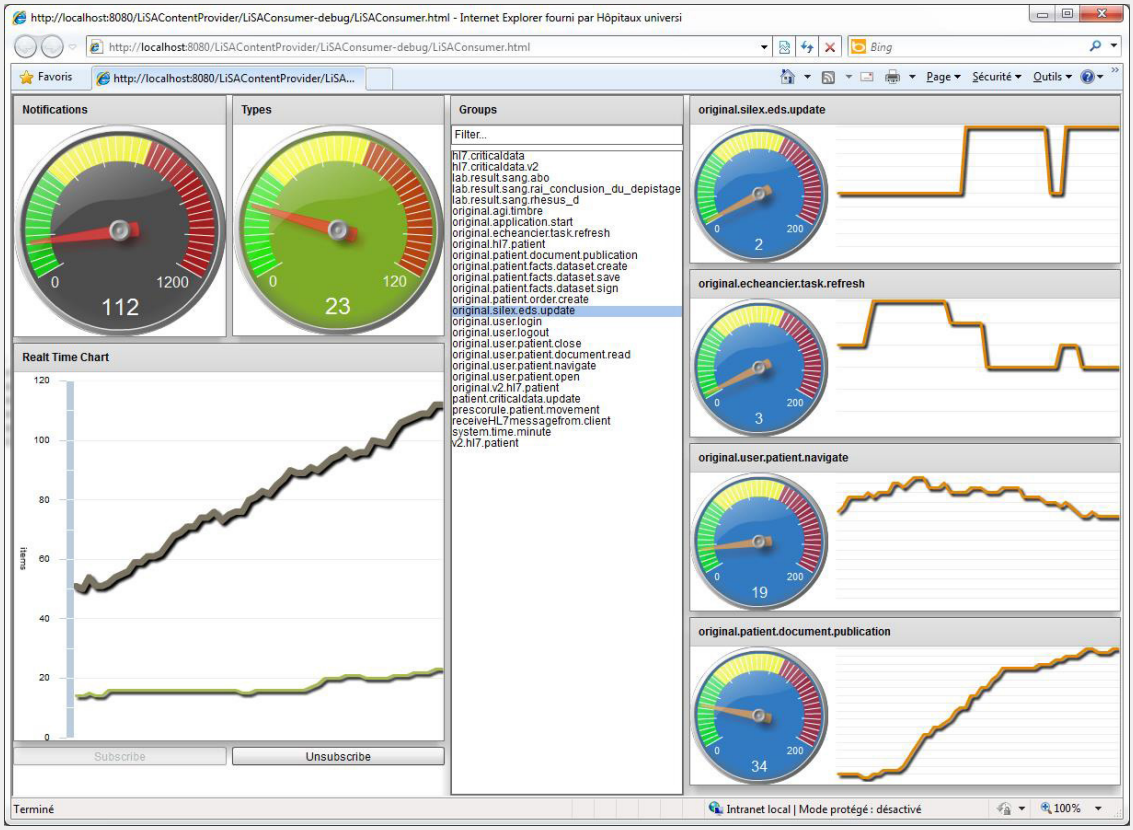

Figure 4

Web interface prototype.