Managing High Disease Risk Factors : a use case in the KMR-II Healthcare Infrastructure

Davide Sottara[1], Emory Fry[2], Esteban Aliverti[3], Mauricio Salatino[3], and John Harby[4], Stefan Killen[4], Tuyet Nguyen[4], Mike Wright[4]

[1] Università di Bologna (davide.sottara2@unibo.it)
[2] Naval Health Research Center (eafry@gmx.com)
[3] PlugTree
[4] Soadex Inc.

Abstract. The goal of the KMR-II project is the creation of a unified architecture for a knowledge intensive patient healthcare management. Using a combination of knowledge integration techniques, it provides a framework for the interaction between patients, providers and managers. At the same time, is provides intelligent clinical decision support and automates management procedures. In this paper, we introduce some of the architectural aspects and show how it allows to manage a simple use case, where a predictive model is used to notify both a patient and their provider that there is a high probability that the patient will develop some disease in the future.

1 Introduction

Supply and demand forecasting in the healthcare industry is predicated on the dual requirement of an accurate analysis of care resources within a community and equally accurate demand forecasting. Unfortunately, current predictive models for projecting population-based demand and the resource required to provide quality care are simply inadequate. It is not unusual for organizations to estimate demand using simplistic models for generalized populations regardless of the clinical characteristics of the patients being served, or to adequately analyze the capacity of availability of resources.

The Distributed Decision Support and Knowledge Management Repository (KMR-II) system provides healthcare planners with the analytic tools they require to develop and deploy sensitive predictive models tailored to the specific characteristics of the target population. It allows organizations to (i) develop disease specific models for accurate forecasting of demand (anticipated number of patients with a specific disease), and (ii) plan for the optimal utilization of existing resources to care for them. It is focused on the integration of commodity rule and workflow management systems for we believe this approach represents the best opportunity to deliver "knowledge services" that can be layered on both military and civilian health information networks.

The second mission of the KMR infrastructure is to change behavior - its inference engines decide what to do, its workflow capabilities automate those tasks, and its notification capabilities communicate with intended recipients. Ultimately, all these capabilities need to be exposed to the end user if behavior is to change. KMR provides managed Presentation Services to assist developers in deploying tightly integrated graphical user interfaces that not only ensure the results of Clinical Decision Support (CDS) and Predictive Analytics can alert and inform the intended user, but can also materially effect behavior by writing orders, booking appointments, changing device settings, etc ...

To this end, we are developing a knowledge intensive infrastructure, capable of leveraging different forms of knowledge: from a semantic description of the application domain, to predictive, diagnostic and planning models for decision support, to complex event processing for real-time operation management, to business rules and workflows for policy enforcement. We have adopted, but not limited ourselves to, a patient-centric architecture, in which a context ("virtual medical record" or VMR) is built for each patient, where all their historical data are stored. Each context is actually the working memory of a reasoning engine, capable of applying various inference strategies to the the clinical data contained therein. The engine, then, is provided at runtime with the appropriate knowledge, according to the specific evaluations which must be performed on the patient.

To give an overview of the architecture and its design principles, we will show a simple use case built on top of our general-purpose infrastructure. In this scenario, a healthcare organization recommends the usage of a set of predictive models, in order to evaluate the risk level a person may have, to develop one or more diseases. A provider is given the capability to apply one or more of those models to their patients. The models would leverage the clinical, historical data present in a patient's medical record - or allow the provider to fill in any missing piece of information - to estimate the probability (with an associated confidence interval) that the patient might suffer from a certain disease in the future. The result of this evaluation, usually based on a comparison of the patient's characteristics with the features of a reference population of individuals, can then be used to apply diagnostic and preventive actions[5]. In particular, should a model detect that a risk threshold has been exceeded for a disease, the system would make sure that both the patient and their provider are notified of this event, delivering a content-rich, informative alert message, using one or more predefined channels (e-mail, SMS, voice call, ...) and ensuring that an acknowledgment is returned.

The knowledge-based part of the architecture has been implemented using the open source Knowledge Integration Platform Drools. In addition to its friendly licensing model, it is a suite of tools which allows to model, combine and execute different forms of knowledge in the same environment, facilitating the development of complex applications such as the one we are describing. Drools comes in the form of a core rule engine, supporting a rich, semi-declarative language, and is extended with a set of additional modules which greatly enhance its inferential capabilities. In section 2, then, we will describe how the tool has been used to

[5] also recommendable by the system, although the topic is not covered in this paper

develop the core of our application, while section 3 will provide more details on the use case we are presenting.

2 A Knowledge Intensive Architecture

Nowadays, large-scale applications are implemented either using Service-Oriented Architectures (SOA), Event-Driven Architectures, or a combination thereof. The two have been proven to be complementary ([4]), since the distinction mostly depends on how the components interact (synchronously vs asynchronously) rather than on structural differences. In our setting, we need both event-driven and request-driven services. A patient record would be continuously updated with information coming from various and heterogeneous sources, possibly triggering the parallel execution of different policies and processes. At the same time, various individuals in the healthcare management organization could request access or even perform operations on the patient's VMR to manage their status.

Given the nature of the data in the records, the knowledge-intensive nature the operations to be performed, the necessity to adapt the policies as the patient's conditions change in time and ultimately the need to provide a highly dynamic set of functionalities, we chose not to implement the core functionalities strictly as services. While still using more traditional (web) service interfaces for peripheral components such as the persistence, security and GUI sub-systems, we have decided to found the core of the architecture on the concept of rich, intelligent agents rather than services. Even from a modelling perspective, an "agent" better represents the idea of a dedicated patient manager: it is the agent, then, who is in charge of processing the incoming events and providing the required services.

Drools Agents. A Drools Agent is itself a knowledge-based application: its internal logic is programmed using rules rather than traditional imperative code. Its external interface is based on the concept of *communication performatives* and has been designed according to the principles of the FIPA[6] standard. This standard regulates the interactions between intelligent agents, defining formats for messages and their content, in addition to formalizing protocols for agent-to-agent communications. To integrate agents in a non agent-based SOA, the agents expose a single endpoint as a (web) service, named `tell`. This service allows to deliver messages to the agent, either from another agent or a more traditional service. When a message is received, it is immediately inserted into a knowledge session, where a set of rules will interpret and process it. The agent currently understands most of the FIPA performatives and is designed to rely on the concept of *expectation* [1] to make sure that the appropriate protocols are respected when exchanging messages (e.g. either refusals or responses are received in a timely fashion for a given request).

[6] http://www.fipa.org/

Internally, the agent leverages the functionalities of `drools-grid` to deploy and maintain separate knowledge sub-sessions (i.e. runtime instances of the reasoning engine) on different nodes of a network. Each session is usually dedicated to a single patient. The rules in the main agent session, then, preprocess the messages and implement a dynamic content-based routing service [3], to ensure that messages regarding a specific patient are ultimately processed in the appropriate sub-session. The master session, in fact, converts each message into a set of working memory commands (assert, retract, query, . . .). These commands are executed in the appropriate sub-session: when the results are available, they are delivered back to the main session where the appropriate response message is generated. In addition to the obvious load balancing benefits, this architecture allows to separate the messaging and protocol management rules (applied in the main session) from the clinical knowledge, since the sub-sessions are not aware of the agent-oriented nature of the environment sustaining them. An overview of the agent structure is summarized in figure 1.

While our architecture will support most FIPA performatives, the most relevant are `Inform`, `Request` and `Query`.

– `Inform` is used mainly for event notifications: it allows the asynchronous delivery of both data and events to any patient session. The events will be able to trigger consequences, but the source of the event will generally not be notified.
– `Query` is used to access the patient records and extract any information present therein in the form of facts
– `Request` is used to request the execution of specific actions on a patient session. A request is implemented using a fact insertion (to trigger the execution) followed by a query (to get the results).

The use of the `Request` performative, in combination with a rule-based management of the request message content allows to provide dynamic, declarative services. A `Request` contains an `Action` with a name and a list of arguments. The management rules are based on patterns matching the `Actions` and, as their consequences, trigger the actual executions aimed at satisfying the request, either directly or through chaining. These rules, then, are equivalent to the implementation of a service. With respect to more traditional services, interface contracts and registration and discovery functionalities are not (yet) supported, but the declarative nature of the implementation has many advantages in terms of deployment, maintenance and lifecycle management in general. Rules, in fact, are generally managed using a (Business) Rule Management System (`drools-guvnor`, in our case) which takes care of issues such as authoring, publication and versioning.

Rich Knowledge Sessions. To actually implement the patient specific services, the individual sessions can leverage the full power of the reasoning engine. The extension of production rules with event processing (`drools-fusion`) and workflow management (`drools-jbpm`) capabilities are well known and will not be

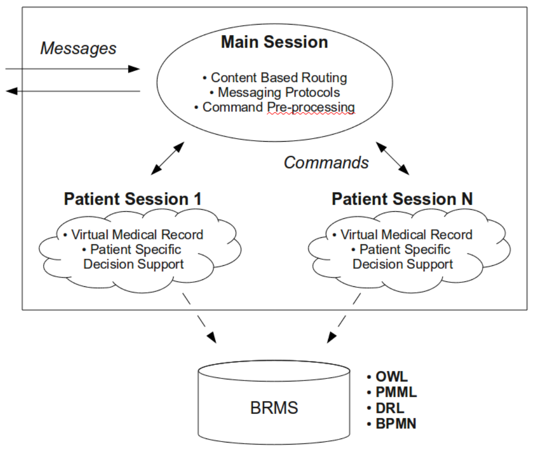

Fig. 1. Clinical Decision Support Agent Architecture

further discussed. To manage our complex use cases, however, additional extensions are being implemented as a part of the KMR-II project. The enhanced modules[7] include:

– `drools-semantics` adds semantic capabilities to the engine. In addition to providing a basic form of semantic reasoning [2], it currently implements some techniques to extract a data model from an ontology [5]. This data model can then be used to ground rules in the concepts defined in the ontology itself. The model is based on java interfaces: in order to bind to the interfaces either object instances or - in alternative - triple-described individuals, a technique known as *traiting* is used, similarly to what has been recently proposed in [8]. A high-level KMR-II ontology is currently being defined, as a part of the project, to describe a medical domain. It will be integrated with other medical "ontologies" to include standard vocabularies.
– `drools-pmml` allows to deploy PMML-encoded predictive models into a knowledge session, where they can be evaluated reactively. PMML, a standard for predictive model exchange, has been designed to declaratively describe the models' structure and parameters, and allows to exchange models between different engines. Models, then, can be trained using tools such as Knime or Weka and imported into Drools for runtime execution. Interestingly, a Drools-based compiler is used to translate a model into a serie of rules which emulate the behaviour of the predictive models themselves, allowing a seamless, homogeneous integration of the models into the rule session [7].
– `drools-chance` extends the traditional inference mechanisms to support uncertain and vague reasoning in a native way [6]. It will be used by the semantic and predictive reasoners, since their outputs are not boolean in general.

[7] https://github.com/droolsjbpm/drools-chance

– `drools-informer` is another rule-based module, derived from a refactoring of the open source tool previously known as "Tohu"[8]. It allows to automatically create and manage *Questions*, special objects which can be bound, at the same time, to a widget in a GUI interface and to a target object's field. Sets of `Questions`, called `Questionnaires`, generate dynamic forms for the dynamic update of facts inside the working memory, effectively making a knowledge session interactive.

3 Use Case - Risk Factor Prediction and Management

To demonstrate the flexibility of the infrastructure, we propose the following simple scenario. We assume that a soldier, just returned from deployment in a war zone, visits their provider for a check-up. Given the patient history, the provider has chosen to evaluate a risk assessment model to estimate the probability that the soldier will suffer from post-traumatic stress disorder (PTSD). The system loads the appropriate predictive model and notifies the provider that some relevant model inputs are missing. Those inputs, however, can be easily acquired by interviewing the patient and provided to the system filling a questionnaire. Once the questionnaire is complete, the model will be evaluated: depending on the actual answers, the estimated risk value might be above the threshold set by the provider. If this is the case, the system will generate two alert messages: one for the provider and one for the patient. The messages are accessible through a "universal inbox", a web-based frontend which displays messages in a fashion inspired by traditional email clients. The body of the messages, however, contains dynamic content, including an interactive form that is used by the recipient to acknowledge the message after it has been read. If the message is not acknowledged within a given deadline, a second message - in the form of an SMS - is delivered as a reminder.

Use Case Implementation. This use case leverages most of the components in the architecture. The presentation services, interacting with the GUI, use three of the decision support agent "services": `getRiskModelDetails`, `getForm` and `setForm`. The first is used to request the deployment (if not already available) of a specific PMML model in a patient's session and it subsequent evaluation, returning the estimated percentage and any correlated information (e.g. the confidence interval)[9]. The second is used to get any interaction questionnaire metadata (generating it if necessary), which allows the GUI to render a web form to collect the answers from the user. In combination with `getRiskModelDetails`, it returns the questionnaire associated to a predictive model instance. The third, instead, delivers the answers to specific questions and returns the result of any validation check, if present.

When the risk threshold is exceeded, a message generation and management process is started internally. The agent has been configured to generate the ap-

[8] http://www.jboss.org/tohu

[9] In the application demo, we use a few mock models

propriate alert messages and deliver them to the appropriate recipients using the appropriate channels. It collects data from the current context (patient, provider, disease, risk level, . . .) and uses them to instantiate one or more message templates. (The actual delivery of the messages is not done directly, but delegated to another dedicated agent, which is out of scope for the purpose of this paper). Each message will also contain the reference to an interaction form, prepared to collect the acknowledgments from the recipient. When the inbox client renders the message content, in fact, the GUI will again invoke the `getForm` and `setForm` services to let the user and the agent interact.

4 Conclusions

This paper shows some aspects of the current state in the development of the KMR-II clinical decision support system infrastructure. The system allows to integrate different types of knowledge (semantic, predictive, operational, . . .) regarding a patient and the best way to provide healthcare for them. The system then applies this knowledge to the data definining a patient's clinical history, automating some management actions and acting as a (clinical) decision support system for the patient's provider and their organization.

The knowledge-based core of the application is deployed in a broader service and event oriented architecture, integrating various data sources and services. The application, moreover, provides a unified web-based interface for both patients and providers, allowing them to interact with the system and between each other in a seamless way.

Acknowledgments

The KMR project is supported by Award Number: W81XWH-11-2-0062, administered through the U.S. Army Medical Research Acquisition Activity, 820 Chandler Street, Fort Detrick, MD 21702-5014.

The opinions of the authors do not necessarily state or reflect those of their respective employers, including the Department of Navy, the Department of Defense, and the United States Government, and shall not be used for advertising or product endorsement purposes.

The whole platform will be released under a BSD-like open source license in early 2012.

References

1. Bragaglia, S., Chesani, F., Fry, E., , Mello, P., Montali, M., Sottara, D.: Event condition expectation (ece-) rules for monitoring observable systems (2011)
2. Bragaglia, S., Chesani, F., Mello, P., Sottara, D.: A Rule-Based Implementation of Fuzzy Tableau Reasoning. In: RuleML. pp. 35–49 (2010)

3. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
4. Luckham, D.: SOA, EDA, BPM and CEP are all Complementary
5. Meditskos, G., Bassiliades, N.: Clips-owl: A framework for providing object-oriented extensional ontology queries in a production rule engine. Data Knowl. Eng. 70(7), 661–681 (2011)
6. Sottara, D., Mello, P., Proctor, M.: Adding uncertainty to a rete-OO inference engine. Rule Representation, Interchange and Reasoning on the Web pp. 104–118 (2008)
7. Sottara, D., Mello, P., Sartori, C., Fry, E.: Enhancing a production rule engine with predictive models using pmml. In: Proceedings of the 2011 workshop on Predictive markup language modeling. pp. 39–47. PMML '11 (2011)
8. Stevenson, G., Dobson, S.: Sapphire: Generating java runtime artefacts from owl ontologies. In: CAiSE Workshops. pp. 425–436 (2011)