Clinical Decision Support System for Point of Care Use: Ontology Driven Design and Software Implementation

K. Farion
Departments of Pediatrics and Emergency Medicine, University of Ottawa
Children's Hospital Of Eastern Ontario
Ottawa, Canada

W. Michalowski, Sz. Wilk*, D. O'Sullivan
MET Research Group, Telfer School of Management, University of Ottawa
Ottawa, Canada

S. Rubin
Department of Surgery, University of Ottawa
Children's Hospital Of Eastern Ontario
Ottawa, Canada

D. Weiss
Institute of Computing Science, Poznan University of Technology
Poznan, Poland

* Corresponding author:
Szymon Wilk
Telfer School of Management
University of Ottawa
55 Laurier Ave East
Ottawa, ON K1N 6N5
tel.: +1 613-562-5800 x. 4196
fax: +1 613-562-5164
e-mail: wilk@telfer.uottawa.ca

Summary

Objectives

The objective of this research was to design a clinical decision support system (CDSS) that supports heterogeneous clinical decision problems and runs on multiple computing platforms. Meeting this objective required a novel design to create an extendable and easy to maintain clinical CDSS for point of care support. The proposed solution was evaluated in a proof of concept implementation.

Methods

Based on our earlier research with the design of a mobile CDSS for emergency triage we used ontology driven design to represent essential components of a CDSS. Models of clinical decision problems were derived from the ontology and they were processed into executable applications during run time. This allowed scaling applications' functionality to the capabilities of computing platforms. A prototype of the system was implemented using the extended client-sever architecture and Web services to distribute the functions of the system and to make it operational in limited connectivity conditions.

Results

The proposed design provided a common framework that facilitated development of diversified clinical applications running seamlessly on a variety of computing platforms. It was prototyped for two clinical decision problems and settings (triage of acute pain in the emergency department and postoperative management of radical prostatectomy on the hospital ward) and implemented on two computing platforms – desktop and handheld computers.

Conclusions

The requirement of the CDSS heterogeneity was satisfied with ontology driven design. Processing of application models described with the help of ontological models allowed having a complex system running on multiple computing platforms with different capabilities. Finally, separation of model and runtime components contributed to improved maintainability – changes in runtime components did not require changing the application models.

Keywords

decision support systems, clinical; point of care systems; software design; ontology driven design

1. Introduction

Research described in this paper is concerned with supporting decision making by physicians at the point of care in an acute and emergency care settings. It started several years ago with development of a clinical decision support system (CDSS) to support emergency triage of patients presenting to the emergency department (ED) with acute pain conditions. This system called MET1 (Mobile Emergency Triage) [1] was designed to help with management of pediatric patients using information about their history, physical examination and a limited number of laboratory tests. MET1 included two clinical applications (supporting triage of pediatric abdominal pain [2] and pediatric scrotal pain [3]) and it ran exclusively on handheld computers. The MET1 abdominal pain application (MET-AP) was prospectively evaluated in the ED at the Children's Hospital of Eastern Ontario in 2003-2004 in order to compare its triage accuracy with the accuracy of emergency physicians. The detailed results of this evaluation are reported in [4].

MET1 was designed to support a set of homogeneous decision problems in a single setting (ED) and to operate on a single computing platform (a handheld computer). This limited design was typical of early generation mobile CDSSs [5], and it was sufficient for supporting the basic functionality. However, to allow wider implementation and adoption, CDSSs need to support physicians evaluating heterogeneous decision problems in different settings, and to run seamlessly on various platforms automatically scaling to their capabilities [6]. In response to these demands for versatile and flexible CDSSs we propose a new CDSS design (referred to as MET2) that represents the next generation of CDSSs.

The requirements for CDSS expanded functionality come from research (see selected papers in [7]) and were also expressed by physicians who participated in the MET1-AP trial. While decision support at the point of care using handheld computers was a helpful and important aspect of the functionality, ED physicians favored a system running on a wider range of devices, thereby enabling a wider range of functionality. For example, while categorical data was easily entered through drop-down menus on the portable device, the ability to enter comments was very limited and would be facilitated by a device with a keyboard and a larger display. Ammenwerth et al. [8] describe the contradictions between requests for small and transportable devices versus those with larger displays. Furthermore, the tasks being performed as well as personal preference significantly influence adoption of different devices. In response, they propose a "multi-device architecture" for electronic information processing and communication in the clinical setting. A similar claim is put forward by activity-based computing – a new paradigm for pervasive clinical computing that considers user activities (tasks) as first class objects in a computing environment [9]. According to this paradigm users should be able to resume their activities on arbitrary computing devices (available at the point of care or suited for the task).

This paper is organized as follows. In the next section we discuss related research on using ontology in CDSS design. In Section 3 we briefly describe the design of the MET1 system and in Section 4 we outline how we responded to a challenge of creating the new generation of CDSSs with the design for MET2. In Section 5 we elaborate on the technical aspects of the MET2 design and describe its implementation. This description does not cover the issues of integration with existing hospital systems, as they were reported earlier [1]. Finally, we conclude with a discussion in Section 6.

2. Literature Review

Ontology represents both the explicit and implied concepts used within a particular discipline, and the relationships between these concepts [10,11]. Ontological engineering, dealing with developing and using ontology [12], has become an important research focus in information science.

In the field of medicine ontology has been extensively exploited in the form of controlled terminologies and classifications for knowledge representation, understanding and exchange. This line of research is exemplified by attempts to define standardized classifications and nomenclatures such as SNOMED-CT® [13] or OpenGALEN [14].

In recent years, use of ontology as a mechanism for representing knowledge in CDSSs has gained momentum [15] and has become more common in supporting and solving decision problems [16,17]. This has coincided with the evolution of CDSS architectures [18]. The first CDSSs were standalone systems that were running separately from other hospital systems (e.g., AAPHelp for diagnosing abdominal pain [19]). They evolved into integrated system where decision support was embedded into hospital information systems (e.g., HELP offering support in such clinical areas as laboratory, nurse charting, radiology or pharmacy [20]). Then, the integrated systems evolved into separated systems with shareable information and decision support content (e.g., SEBASTIAN [21] with multiple XML-based modules containing clinical knowledge in machine-executable format).

Ontology can be used to construct knowledge bases with instances of defined concepts (these instances represent facts about specific problems), thus, it has been employed to represent information and knowledge in the systems with shareable information. Subsequently to this line of research, medical informatics has also explored ontology driven design, where ontology and derived knowledge bases are separated from domain independent processing algorithms (often referred to as solving algorithms or solvers) [22,23]. The idea of separating ontology and derived knowledge bases from solvers enables reusability and also improves the robustness of the system design. One solver may be used with different ontology and knowledge bases, and vice versa. Moreover, new solvers may be added without having to alter the ontology or the knowledge base, and changes in the ontology and the knowledge base do not require modifying solvers.

The idea of separated ontology and solvers was used in the EON system. EON provided a set of middleware components to automate various aspects of protocol directed therapy (checking whether a patient was eligible for a particular therapy and planning the therapy) [24]. It was developed for clinical trial protocols for the treatment of cancer and HIV infections, and later it was extended to cover the management of chronic diseases and other types of guidelines. Now EON is used in the ATHENA CDSS for hypertension management [25]. EON included the generic guideline ontology that defined concepts related to clinical protocols and the medical specialty ontology that defined concepts corresponding to findings and interventions for a particular area of medicine. Adding support for new problems required extending the medical specialty ontology and linking it to the guideline ontology. Moreover, EON contained two generic solvers – for checking protocol eligibility and for planning therapy.

A similar ontology driven design was applied in BioSTORM – a system for syndromic surveillance (monitoring of prediagnositc data for early detection of disease outbreaks) [26]. The system uses the data-source ontology and the problem-solving ontology. The data-source ontology defines characteristics, types and relationships of monitored data coming from various

sources (e.g., 911 emergency data). The problem-solving ontology organizes and characterizes solvers available in the system in terms of data and knowledge a given solving method uses. BioSTORM also includes the controller component that identifies and deploys solvers to analyze incoming data streams.

The ontology driven design has not been used only in CDSSs but also it has been applied in bioinformatics to design grid systems [27]. For example, PROTEUS is a grid-based problem solving environment for composing and running applications aimed at analyzing sequences of proteins [28]. It allows for multiple applications represented as distributed workflows of software components. The system includes the domain ontology and the application ontology. The domain ontology classifies and describes concepts in the domain of bioinformatics as well as available resources (e.g., solvers and external databases). The application ontology classifies and describes available application workflows (composed from concepts from the domain ontology). Workflows from the application ontology are executed by the execution manager.

3. Design of the MET1 System

MET1 followed the requirements to support a set of homogeneous decision problems and to operate on a single computing platform. Its logical design involved ontology and derived knowledge bases that were separated from a solver. The MET1 ontology included the data ontology and the support ontology, with the data ontology specifying concepts related to structure of information to be processed, while the support ontology defined concepts related to decision models (all models used by MET1 were rule-based).

Both ontologies were used to derive data models and support models respectively. The data model encompassed a knowledge base with instances of concepts from the data ontology that described the structure of clinical information considered for a specific problem (i.e., a specific acute pain condition). Following the idea of automatic generation of user interfaces for knowledge acquisition tools [29], the data ontology and derived data models were annotated with additional information allowing the construction of customized user interfaces for collecting and presenting clinical data [1]. The support model encompassed a knowledge base composed of the decision rules representing knowledge on how to solve a specific decision problem (e.g., to make triage disposition for a patient with an acute scrotal pain).

The data and support models had to be created for every decision problem handled by the system and each pair formed an application model for this specific problem. Application models were then transformed on request into executable applications. The general design of the MET1 system is presented in Figure 1. The two major architectural components of this design were the application repository and the executor. The application repository managed and stored the available application models. The executor created applications according to their application models and executed them. MET1 included also an interface repository with components for building a user interface and one solver that was used with all decision models.

Typically, upon the ED physician's request, the executor retrieved an appropriate application model from the application repository and created the user interface according to the data ontology using components from the interface repository. Then, it presented the interface to the ED physician for recording and viewing clinical data and for calling the triage support function. When this function was invoked, the executor linked the solver with the support model, solved it for collected data, and presented the results. After the physician had finished working with the

application, the executor purged the application model and was ready to respond to the next request.

4. Design of the MET2 System

Successful adoption of CDSSs in clinical practice depends on their broadly understood versatility. Specifically, a new generation CDSS should be able to support heterogeneous decision problems (in particular those that require heterogeneous decision models and solvers) at different settings and to execute seamlessly on multiple computing platforms [6]. The design of earlier CDSSs was too limiting to satisfy these requirements – Table 1 lists the major shortcomings. These shortcomings prompted us to propose the novel design of MET2. Despite expanding the system functionality beyond emergency triage we decided to stay with MET acronym as a label identifying our research.

The first step in addressing the design shortcomings of earlier CDSSs was to revisit the ontology. Figure 2 presents the ontology used in the MET2 design. To improve readability only the most important classes and "is-a" and "association" relationships are presented (e.g., it shows that a rule solver *is a* solver or that data entry forms are *associated* with user interface). It expands the MET1 support ontology and introduces two new components – the interface ontology and the configuration ontology. These two new ontologies are used to derive interface and configuration models that enhance the application model, so in MET2 it encompasses data, support, interface and configuration models. Moreover, we allow application models to include several support and interface models that are suited to capabilities (e.g., memory, computational power, display size, interaction modalities) of specific platforms, thus dealing with possible platform variability.

The support ontology has been extended to handle decision problems requiring different types of decision models and different solvers. We have introduced concepts representing different types of decision models, different types of solvers, and the associations between them. When deriving specific support models from the support ontology, these associations allow coupling decision models with solvers, so the executor knows which solver to invoke when running an application.

An application model may include several platform-specific support models, thus, a MET2 application running on a powerful platform can use a complex support model (where a complex decision model is coupled with a complex solver), while the same application executed on a computationally weak platform may switch to a simplified support model. A drawback associated with such scaling is that these two models may provide potentially contradicting outcomes for the same patient. This issue may be addressed by tuning simplified support models so their solution strategy is more conservative than that of complex models, following the percept that if in doubt, the system should suggest a more conservative course of action.

The interface ontology introduced in MET2 defines concepts representing various components of the user interface (e.g., forms and attribute editors). It is used to derive interface models included in specific application models. Explicit representation of user interface components allows us to define sophisticated user interfaces and address one of the shortcomings associated with the earlier CDSS design where the user interface was described by simply annotating data models. With several platform-specific interface models included in an application model, the application executed on a handheld computer may split collected and displayed information across multiple screens and use handwriting recognition for data entry, while the same application running on a

desktop computer displays all information on a single screen and allows entering data with a keyboard and mouse.

The configuration ontology is introduced in MET2 to handle increased complexity of application models that may include multiple platform-specific support and interface models. This ontology is used to derive configuration models that link support and interface models with target computing platforms – such links are called “profiles” (one configuration model may specify multiple profiles for different platforms).

The MET1 data ontology allowed defining data models for different decision problems and it could be reused in MET2 design without extensive changes (it only required stripping annotations on the user interface as they were made redundant by introducing the interface ontology). However, in the MET2 design, we introduced the Entity-Attribute-Value (EAV) approach to structure clinical information as it allows for more flexible and effective handling of heterogeneous data [30]. In this approach the concept representing a patient-physician encounter becomes a central entity that is characterized by a set of clinical attributes, and its instances are described by values of the attributes. The attributes specify the historical information, physical examination findings and test results that should be collected during an encounter and considered when making a decision about a patient. A data model derived from the data ontology contains definitions of these attributes for a specific decision problem (e.g., triage of a scrotal pain).

The data ontology is relatively simple and represents data collected during an encounter, because such information (the most recent medical history and a current patient state) is required to provide early decision support at the point of care. The data ontology is also used to facilitate integration with other hospital information systems because data interoperability can be ensured at this level. If past patient data recorded in the electronic patient record (EHR) is required, definitions of clinical attributes can be expanded with information about how their values should be retrieved from the EHR. With this information explicitly available, MET2 will not only be able to analyze values of attributes entered by the physician, but also to use data already stored in the EHR.

The general system design of MET2 is presented in Figure 3. In order to address the CDSS design shortcomings associated with the “single solver, single platform” approach, we have introduced two new architectural components: the adapter and the solver repository that replaces a single solver. The adapter is responsible for adapting a multi-platform application model to a specific platform and the solver repository stores all required solvers. The MET2 design assumes that the interface and the solver repositories store solvers and interface components for multiple computing platforms. In order to simplify the description we will refer to solvers and interface components as runtime components, and to the interface and solver repositories as runtime repositories.

Upon the ED physician’s request, the executor manages the creation of a specific application in MET2. The executor is aware of the computing platform on which it runs, and this information is appended to the request sent to the application repository. The retrieved multi-platform application model is passed to the adapter, which adapts it to the requested platform by selecting the interface and the support model referenced in the corresponding platform profile. The platform-specific model is transferred back to the executor that creates the application using platform-specific runtime components retrieved from the runtime repositories and executes it. When the physician finishes using the application, the executor purges the application model and

the retrieved runtime components to release computing resources. Such a request-execute-purge cycle allows running multiple complex applications on computationally weak platforms.

Separation of runtime components from application models and the request-execute-purge cycle significantly improves extensibility of the MET2 system and the reusability of its components. New application models are added to the system by storing them in the application repository. If the required runtime components are not already available in the runtime repositories, such components have to be added, however no other changes to system components are necessary. Also, changes in specific application models require either no changes to the system, or they are limited to the runtime repositories. Finally, multiple application models may easily share runtime components and thus any update of shared components is immediately registered by all applications.

5. Implementation of the MET2 System

5.1 High-level Implementation

In implementing the MET2 design we followed the client-server paradigm. The simplest approach would be to install the executor on the MET2 client and the remaining architectural components on the MET2 server. This would require a permanent connection between the client and the server. However, interruptions to wireless connectivity are typical in most hospital settings for various reasons. Moreover, maintaining permanent connection on a mobile device may be power consuming, thus severely limiting the usefulness of the system by forcing the physician to charge the device frequently. Thus, we assumed that the system should be able to function off-line with occasional connections between the client and the server and implemented it using the extended client-server architecture [31]. When duplicating the server functionality on the client side, we assumed that the client had to store a subset of available application models and runtime components so it is able to respond to physicians' requests for applications without connecting to the server. This required hosting some of the architectural components on the client and on the server. Specifically we duplicated the application repository and the runtime repositories (the interface repository and the solver repository).

The high-level implementation of MET2 design is presented in Figure 4. For clarity the duplicated components located on the server side are labeled as *central* and those located on the client side are labeled as *local*. This implementation introduces two new architectural components that were not present in the general design (see Fig. 3) – the retriever and the packager. The retriever, upon request from the executor, retrieves the required application model from the central repository and passes it to the adapter. The retriever is necessary on the server side to allow the executor to reach the remote central repository. On the client side, the executor retrieves application models directly from the local repository. The packager facilitates the process of transferring required runtime components from the server to the client. It captures a platform-specific application model processed by the adapter and examines the interface and the support models to identify which runtime components are required for executing the application. Then, it retrieves the components from the central runtime repositories, packages them with the application model and passes the whole "package" to the executor. The executor stores the application model in the local application repository and the runtime components in the local runtime repositories (as all application models stored locally on the client side are single-platform, there is no need for the local adapter). After storing the contents of a package, the

executor creates and runs the application according to its model. When the application is no longer necessary, the executor may purge the received application model and the runtime components or cache them for future use.

Following our earlier experience [1], we decided to introduce the patient repository that stores information about currently processed patients (it stores instances of classes defined in the data ontology) and acts as a buffer between the MET2 system and hospital systems [1]. We use the central patient repository located on the MET2 server with all currently processed patients, and the local patient repository with patients processed locally on the MET2 client. When connection between the client and the server is available, the central and the local patient repositories are synchronized.

5.2. Low-level Implementation

The low-level implementation of the MET2 system design is presented in Figure 5. The figure also identifies specific technologies and tools used to develop a working prototype of the system. We used Java as an implementation language because it is available for mobile and desktop computing platforms. Moreover, a system written in Java allows transferring runtime components over a network – this was critical for sending runtime components from the MET2 server to the MET2 clients. In the implemented version of the MET2, technology constraints forced us to look for some non-standard solutions described below. To simplify the implementation we used the same solutions on both mobile and desktop platforms.

The MET2 server was implemented on a Java application server to provide Web service communication between the server and clients. The central application repository was realized as a repository managed by Protégé [32]. Protégé handles multiple ontological languages (e.g., frames or OWL Web Ontology Language [33]) and offers dedicated editor for easy creation and modification of ontology and derived models. It also provides an advanced programming interface in Java, so repositories can be programmatically accessed and modified. We decided to use the simpler frame-based representation, because the extensive capabilities of OWL (e.g., reasoning about ontology) were not required for this implementation.

The Protégé repository is also used to store patient data – this simplifies implementation as we use a single storage mechanism. Although this solution offers limited efficiency (e.g., Protégé repository lacks indexing of its content), it is sufficient for the central patient repository if the number of currently managed patients is limited.

Protégé does not offer any mechanism to synchronize the content of its repositories, therefore we had to develop a synchronizer to keep patient data consistent and to solve potential conflicts with update timestamps. The synchronizer is implemented as the patient manager Web service. The same solution is used for the adapter and the packager. They are combined into the application manager Web service. Both Web services are accessed via SOAP (a standard protocol for invoking Web services) [34].

Central and local runtime repositories are realized as collections of Java archives (JAR files) with compiled Java code. The local patient and application repositories are implemented as XML repositories. To manage these repositories we developed our own programming interface that mimics and abstracts the one offered by Protégé. This was not the most efficient solution because better capabilities are provided by XML-based databases. Unfortunately, at the time of MET2 development, XML-based databases were not available for mobile platforms.

5.3. Executor

The executor, implemented as a Java program, allows physician to work with a specific clinical application. It is the only component of the MET2 system that has to be pre-installed on the client side – the other ones are downloaded by the executor from the server. To limit communication between the client and the server, the executor first checks if the application model is stored locally and requests it from the server using the application manager Web service if it is not available. In response it receives a package with an application model and the necessary runtime components, and then stores the package in the local repositories. During execution the executor checks what Java classes are referenced in the interface and support models, retrieves them from the local runtime repositories and uses them accordingly.

The executor has indirect and direct modes of selecting clinical applications. In the indirect mode the executor presents the physician with a list of patients stored in the local patient repository. When the physician selects a patient, the executor identifies the presenting complaint for this patient and requests a corresponding application model. In the direct mode the physician first selects applications from the list of all applications available on the server. This allows the physician to preload the client with a set of applications (for example the most frequently required) and to limit subsequent communication between the client and the server to patient data synchronization using the patient manager Web service.

5.4. Application Models

Currently, the MET2 system contains models for three clinical applications – triage of pediatric abdominal pain (MET2-AP) [2], triage of pediatric scrotal pain (MET2-SP) [3] and postoperative management of radical prostatectomy (MET2-RP) [35]. The first two models are revised and extended versions of the respective MET1 applications (they involve the expanded ontology and include multiple interface models). Inclusion of the last application model was made possible because of the new MET2 design.

Construction of the application models started with corresponding data models. With the help of clinical experts and medical literature we defined clinical attributes that should have been considered in each decision problem. Then we used retrospective chart data to build decision models considering those that are frequently used in clinical decision making [36]. These models were coupled with corresponding solvers to form support models. The choice of decision models and solvers was verified in a series of computational experiments [37,38]. We also checked the performance of the selected support models on both computing platforms and concluded that scaling was not required, thus the same models were used on desktop and mobile computers.

Finally, we created interface models following user-centered and task-centered design principles [39,40]. For each application model we prepared two interface models – one for a handheld computer and one for a desktop computer. We also created platform-specific interface components that were required to construct these interfaces. Sample user interface screens created from the interface models for the MET2-RP application are presented in Fig. 6 – when executed on a desktop computer, interface displays all data fields on a single screen for easy viewing and navigation (Fig. 6a), while on a handheld computer these fields are split into several tabs to fit a small screen (Fig. 6b).

6. Discussion

According to [41] it is important that the CDSS design shifts from a narrowly focused "single solver, single platform" paradigm to a much broader one. Many CDSSs (e.g., [19,42-44]) were designed as stand-alone applications suitable for a given clinical condition only. Such design does not meet requirements of system's versatility and capability to be executed on multiple computing platforms. With the MET2 design we meet these requirements by providing a unifying environment that can handle multiple clinical applications executed on multiple computing platform.

The new design of MET2 uses the ontology and derived models to represent key components of a CDSS. However, unlike the majority of other design frameworks, it extends the ontology to allow construction of multiple models of specific clinical applications not only in terms of data and support functionality, but also in terms of their user interface and computing platform configuration. The MET2 design separates application models from runtime components and introduces the request-execute-purge cycle of an application. Such a solution ensures extensibility of the MET2 system and reusability of runtime components. The system may be expanded by adding new application models and existing models may be updated without the need to change runtime components. Similarly, runtime components may be updated without changing application models. Finally, runtime components may be shared by multiple application models.

We tested the MET2 system design by implementing the system according to the extended client–server paradigm and by creating models of three applications supporting different clinical decision problems. The application models scale to the capabilities of specific computing platforms (we needed to scale the user interface model only). As a part of future research we plan to work on intelligent scaling of application models to the capabilities of specific platforms. Rather than providing several platform-specific models, an application model would include a single generic (platform-independent) support and interface model that would be processed by an intelligent adapter and transformed accordingly. This approach has been already applied with limited success in creating model-based user interfaces [45]. We will also work on extending the capabilities of a patient data repository by moving to effective database solutions (e.g., XML-based database) and leaving ontology editors (e.g., Protégé) for ontological engineering only.

Following results of the clinical trial of the MET1 system that revealed the differences between decision making process of physicians and medical residents [46], we plan to make the MET2 a user-aware system. This new feature should allow the system to act differently in relation to user's skills level (physician, resident, medical student). We discovered that expert physicians tend to use more information than residents [46], which suggests the support model (the decision model in particular) for an expert may require more inputs than the support model for a novice. Moreover, novice physicians have problems with accurate collection of some clinical findings (especially those related to physical examination of the patient). If these findings are to be used as input information, the user interface should provide additional support through explanations or feedback to facilitate the data gathering task.

Acknowledgments

This research was supported by grants from NSERC-CIHR Collaborative Health Research Program.

The authors would like to thank anonymous reviewers for helpful comments and suggestions.

References

1. Michalowski W, Slowinski R, Wilk S, Farion K, Pike J, Rubin S. Design and development of a mobile system for supporting emergency triage. Methods Inf Med 2005; 44 (1): 14-24.
2. Michalowski W, Slowinski R, Wilk S. MET system: a new approach to m-health in emergency triage. J Inf Technol Healthc 2004; 2 (4): 237-249.
3. Michalowski W, Wilk S, Farion K, Pike J, Rubin S, Slowinski R. Development of a decision algorithm to support emergency triage of scrotal pain and its implementation in the MET system. INFOR 2005; 43 (4): 287-301.
4. Farion K, Michalowski W, Rubin S, Wilk S, Correl R, Gaboury I. Prospective evaluation of the MET-AP system providing triage plans for acute pediatric abdominal pain. Int J Med Inf 2008; 77 (3): 208-218.
5. Fischer S, Stewart TE, Mehta S, Wax R, Lapinsky SE. Handheld computing in medicine. J Am Med Inform Assoc 2003; 10 (2): 139-149.
6. Ball MJ, Silva JS, Bierstock S, Douglas JV, Norcio AF, Chakraborty J, et al. Failure to provide clinicians useful IT systems: opportunities to leapfrog current technologies. Methods Inf Med 2008; 47: 4-7.
7. Berner ES, editor. Clinical Decision Support Systems. Theory and Practice. 2nd ed. New York: Springer Science+Business Media; 2007.
8. Ammenwerth E, Buchauer A, Bludau B, Haux R. Mobile information and communication tools in the hospital. Int J Med Inf 2000; 57 (1): 21-40.
9. Bardram JE, Christensen HB. Pervasive computing support for hospitals: an overview of the activity-based computing project. IEEE Pervas Comput 2007; 6 (1): 44-51.
10. Rogers JE. Quality assurance of medical ontologies. Methods Inf Med 2006; 45 (3): 267-274.
11. Gruber TR. A translation approach to portable ontology specifications. Knowl Acquis 1993; 5 (2): 199-220.
12. Kishore R, Zhang H, Ramesh R. A helix-spindle model for ontological engineering. Commun ACM 2004; 47 (2): 69-75.
13. SNOMED Clinical Terms® User Guide. July 2008 International Release: The International Health Terminology Standards Development Organisation; 2008.
14. Rogers JE, Roberts A, Solomon WD, van der Haring E, Wroe CJ, Zanstra PE, et al. GALEN ten years on: tasks and supporting tools. In: Patel V, Rogers R, Haux R, editors. Medinfo 2001: Proceedings of the 10th World Congress on Medical Informatics. Amsterdam: IOS Press; 2001. p. 256-260.
15. Noy NF, Rubin DL, Musen MA. Making biomedical ontologies and ontology repositories work. IEEE Intell Syst 2004; 19 (6): 78-81.
16. Tu S, Eriksson H, Gennari JH, Shahar Y, Musen MA. Ontology-based configuration of problem-solving methods and generation of knowledge-acquisition tools: application of PROTÉGÉ-II to protocol-based decision support. Artif Intell Med 1995; 7 (3): 257-289.
17. Curbézy M, Musen MA. Ontologies in support of problem solving. In: Staab S, Studer R, editors. Handbook on Ontologies. Berlin, Heidelberg: Springer-Verlag; 2004. p. 321-342.
18. Wright A, Sittig DF. A four-phase model of the evolution of clinical decision support architectures. Int J Med Inf 2008; 77: 641-649.
19. de Dombal FT, Leaper DJ, Staniland JR, McCann AP, Horrocks JC. Computer-aided diagnosis of acute abdominal pain. Br Med J 1972; 2 (5804): 9-13.

20. Gardner RM, Pryor TA, Warner HR. The HELP hospital information system: update 1998. Int J Med Inf 1999; 54 (3): 169-182.
21. Kawamoto K, Lobach DF. Design, implementation, use and preliminary evaluation of SEBASTIAN, a standards-based Web service for clinical decision support. In: Proceedings of the AMIA 2005 Annual Symposium; 2005. p. 380-384.
22. Musen MA, Schreiber AT. Architectures for intelligent systems based on reusable components. Artif Intell Med 1995; 7 (3): 189-199.
23. Musen MA. Scalable software architectures for decision support. Methods Inf Med 1999; 38 (4-5): 229-238.
24. Musen MA. Domain ontologies in software engineering: Use of Protégé with the EON architecture. Methods Inf Med 1998; 37 (4-5): 540-550.
25. Martins SB, Lai S, Tu S, Shankar R, Hastings SN, Hoffman BB, et al. Offline Testing of the ATHENA Hypertension Decision Support System Knowledge Base to Improve the Accuracy of Recommendations. In: Proceedings of the AMIA 2006 Annual Symposium; 2006. p. 539-543.
26. Curbézy M, O'Connor M, Buckeridge DL, Pincus Z, Musen MA. Ontology-centered syndromic surveillance for bioterrorism. IEEE Intell Syst 2005; 20 (5): 26-35.
27. Joutchkov A, Tverdokhlebov N, Yanovsky A, Golitsin S, Arnautov S, Strizh I. Libraries of strategies and ontology-driven subject area models as "corner stones" in Grid development. Methods Inf Med 2005; 44 (s): 249-252.
28. Cannataro M, Cuda G, Veltri P. Modeling and designing a proteomics application on PROTEUS. Methods Inf Med 2005; 44 (2): 221-226.
29. Eriksson H, Puerta AR, Musen MA. Generation of knowledge-acquisition tools from domain ontologies. Int J Hum Comput Stud 1994; 41 (3): 425-453.
30. Nadkarni PM, Marenco L, Chen R, Skoufos E, Shepherd G, Miller PL. Organization of heterogeneous scientific data using the EAV/CR representation. J Am Med Inform Assoc 1999; 6 (6): 478-493.
31. Jing J, Helal AS, Elmagarmid A. Client-server computing in mobile environments. ACM Comput Surv 1999; 31 (2): 117-157.
32. Gennari JH, Musen MA, Fergerson RW, Grosso WE, Curbézy M, Eriksson H, et al. The evolution of Protégé: an environment for knowledge-based systems development. Int J Hum Comput Stud 2003; 58 (1): 89-123.
33. OWL Web Ontology Language Guide. W3C; cited August 29, 2008. Available from: http://www.w3.org/TR/owl-guide/.
34. Simple Object Access Protocol. W3C; cited August 29, 2008. Available from: http://www.w3.org/TR/soap/.
35. Michalowski W, Wilk S, Thijssen A, Li M. Using a Bayesian belief network model to categorize length of stay for radical prostatectomy patients. Health Care Manag Sci 2006; 9 (4): 341-348.
36. Hanson CW, 3rd, Marshall BE. Artificial intelligence applications in the intensive care unit. Crit Care Med 2001; 29 (2): 427-435.
37. Blaszczynski J, Farion K, Michalowski W, Wilk S, Rubin S, Weiss D. Mining clinical data: selecting decision support algorithm for the MET-AP system. In: Miksch S, Hunter J, Keravnou ET, editors. Artificial Intelligence in Medicine. 10th Conference on Artificial Intelligence in Medicine, AIME 2005, Aberdeen, UK, July 23-27, 2005, Proceedings. Berlin, Heidelberg: Springer-Verlag; 2005. p. 429-433.

38. Wilk S, Slowinski R, Michalowski W, Greco S. Supporting triage of children with abdominal pain in the emergency room. Eur J Oper Res 2005; 160 (3): 696-709.
39. Wilk S, Michalowski W, Farion K, Kersten M. Interaction design for mobile clinical decision support systems: the MET system solutions. Found Comput Decis Sci 2007; 32 (1): 47-61.
40. Michalowski W, Kersten M, Wilk S, Slowinski R. Designing man-machine interactions for mobile clinical systems: MET triage support using Palm handhelds. Eur J Oper Res 2007; 177 (3): 1409-1417.
41. Carter JH. Design and implementation issues. In: Berner ES, editor. Clinical Decision Support Systems. Theory and Practice. 2nd ed. New York: Springer Science+Business Media; 2007. p. 64-98.
42. Sadeghi S, Barzi A, Sadeghi N, King B. A Bayesian model for triage decision support. Int J Med Inf 2006; 75 (5): 403-411.
43. Samore MH, Bateman K, Alder SC, Hannah E, Donnelly S, Stoddard GJ, et al. Clinical decision support and appropriateness of antimicrobial prescribing: a randomized trial. JAMA 2005; 294 (18): 2305-2314.
44. Berner ES, Houston TK, Ray MN, Allison JJ, Heudebert GR, Chatham WW, et al. Improving ambulatory prescribing safety with a handheld decision support system: a randomized controlled trial. J Am Med Inform Assoc 2006; 13 (2): 171-179.
45. Eisenstein J, Vanderdonckt J, Puerta AR. Applying model-based techniques to the development of UIs for mobile computers. In: IUI 2001: Proceedings of the 6th International Conference on Intelligent User Interfaces, Santa Fe, New Mexico, USA, January 14-17, 2001. New York: ACM; 2001. p. 69-76.
46. Hine MJ, Farion K, W. Michalowski, Wilk S. Decision making by emergency room physicians and residents: implications for the design of clinical decision support systems. Int J Healthc Inf Syst Inform 2008 (forthcoming).

Fig. 1. General design of the MET1 system

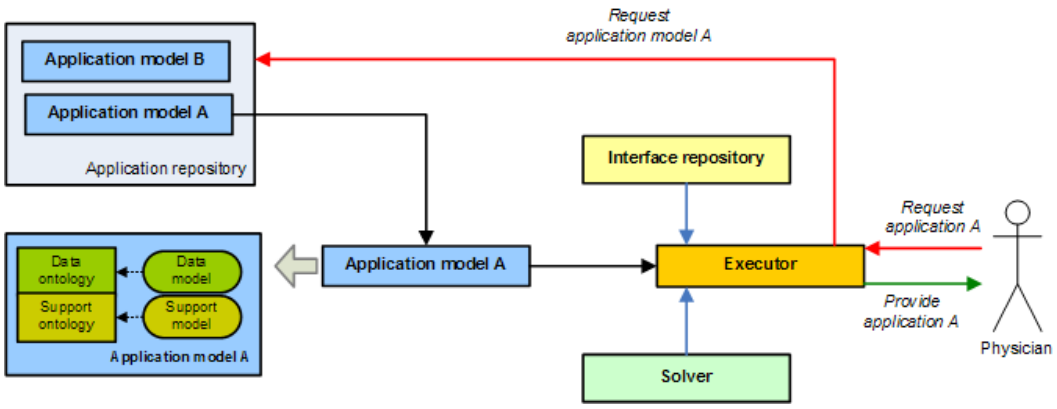


Fig. 2. MET2 ontology

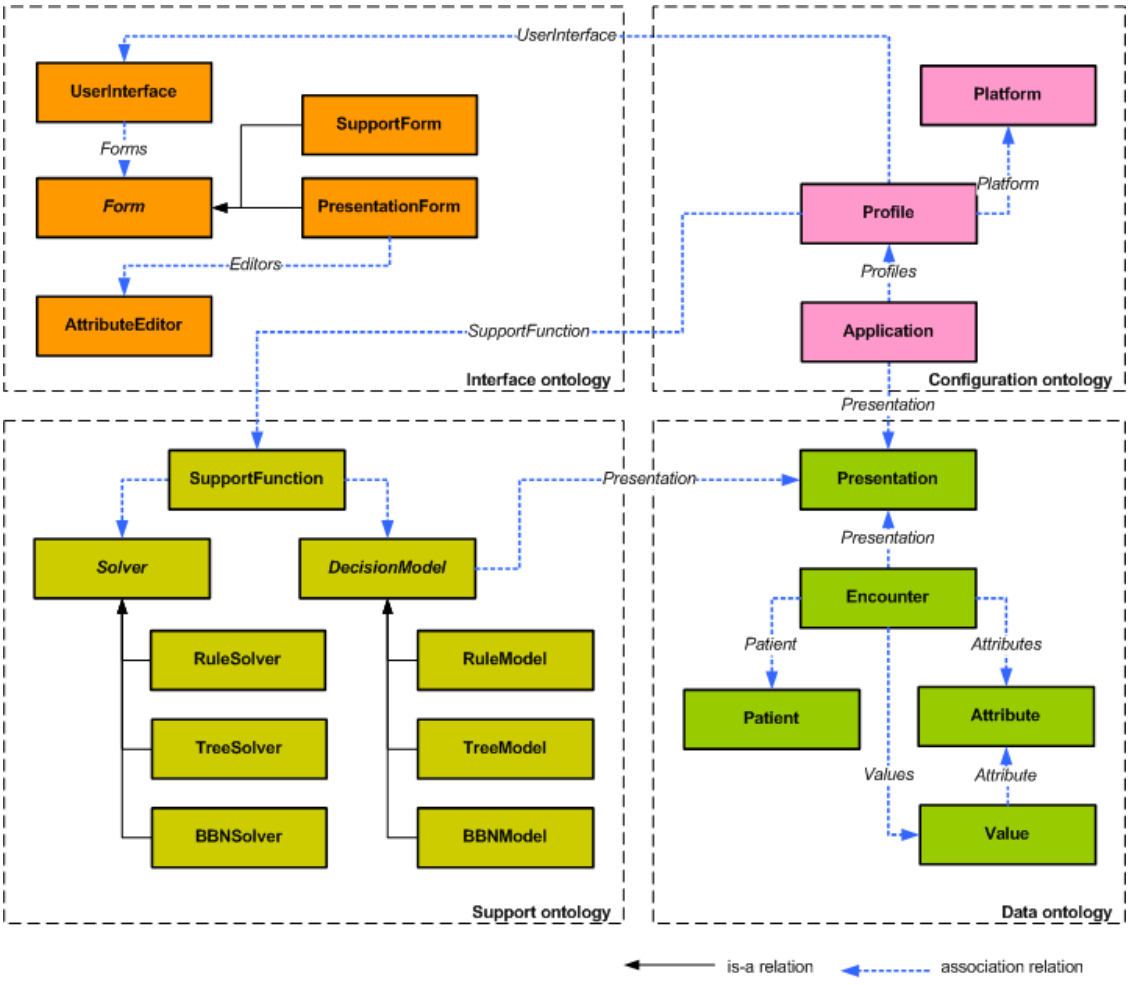


Fig. 3. General design of the MET2 system

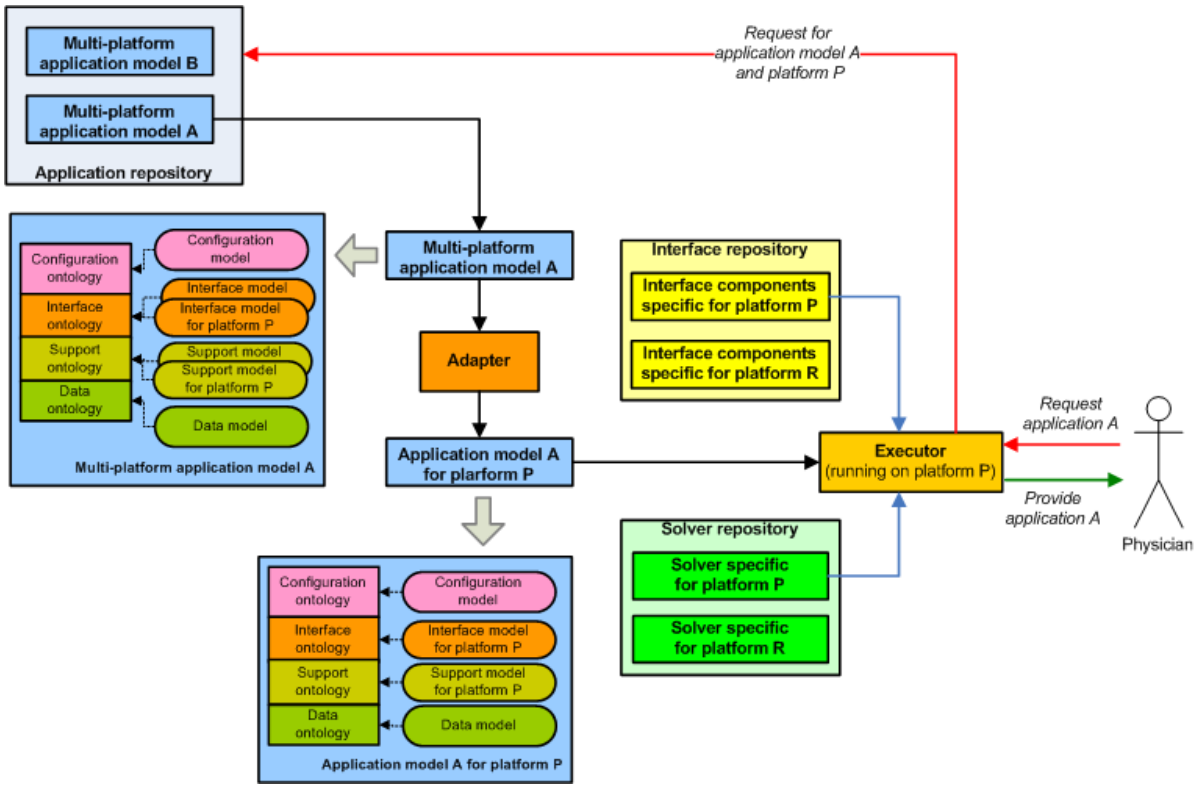


Fig. 4. High-level implementation of MET2 system design

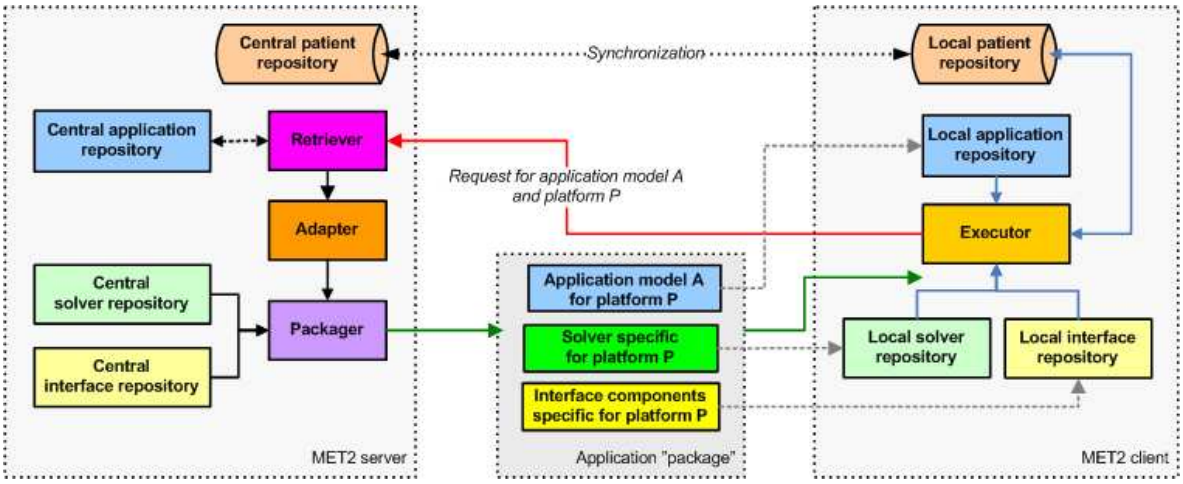


Fig. 5. Low-level implementation of MET2 system design

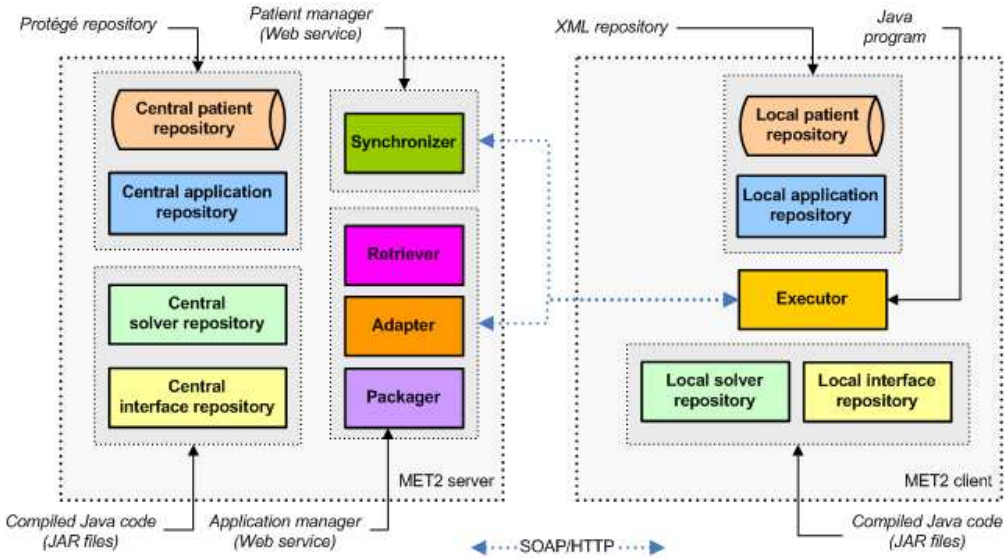
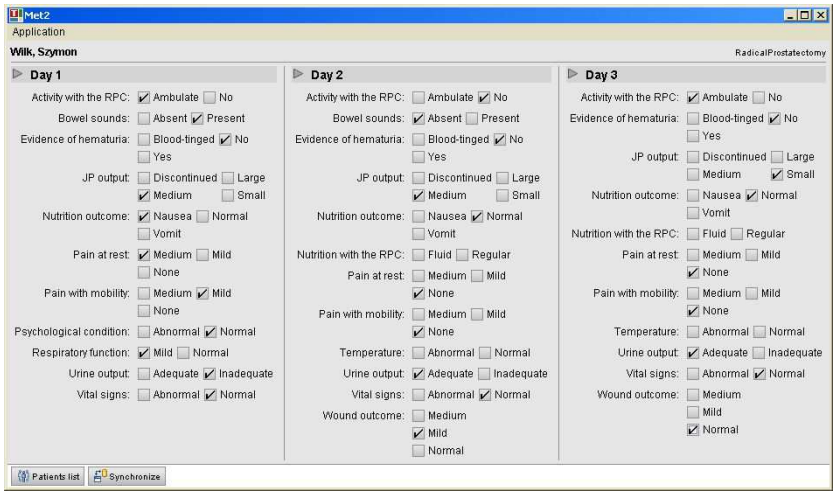


Fig. 6. MET2-RP user interface

a) for a desktop computer



b) for a handheld computer

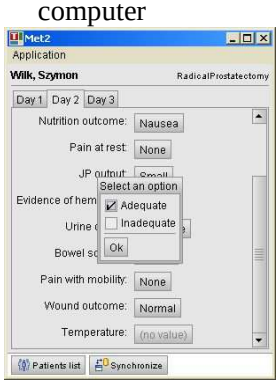


Table 1. Main shortcomings in earlier generation of CDSSs as exemplified by MET1 system

Capability	Level	CDSS shortcoming	MET1 shortcoming
Support for heterogeneous decision problems	Design	▪ Available with similar models	▪ One model type
	Architecture	▪ Limited set of similar solvers	▪ Single solver
Execution on multiple computing platforms	Design	▪ No ability to create sophisticated customized interfaces ▪ No ability to customize support to different computing platforms ▪ No ability to re-use interface components	▪ Interface embedded in a data model ▪ "Hard-wired" interface and support specifications
	Architecture	▪ No ability to run multiple support and platform configurations	▪ Single platform