Arriclides: An Architecture Integrating Clinical Decision Support Models

Kris Verlaenen, Wouter Joosen, Pierre Verbaeten
Distrinet, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
{kris.verlaenen, wouter.joosen, pierre.verbaeten}@cs.kuleuven.be

Abstract

In this paper, we present an open architecture that enables the coexistence of and the collaboration between different and heterogeneous clinical decision support models. Clinical decision support can be (and has been) built starting from complementary knowledge representation models, for instance rule-based representations or workflow models to mention just a few. Each model can contribute to the quality and effectiveness of an overall clinical decision support system. Our architecture, Arriclides, enables the integration of a set of representative and quite different models. An evaluation of our prototype implementation shows that such an integration has been achieved while preserving performance and scalability.

1 Introduction

Clinical decision support systems can aid care providers in optimally diagnosing and treating patients. Guidelines that support care providers in performing their tasks must be defined in a computer-interpretable way to provide patient-specific advice during the clinical encounter. Different types of clinical knowledge models have been developed in the past for expressing clinical knowledge.

This paper presents an open architecture for clinical decision support that enables the coexistence and collaboration between different and heterogeneous clinical decision support models. This allows clinical experts to define clinical knowledge using the most suitable model. Specialized models also allow the use of constructs that are closely related to a certain type of knowledge or decision support: this makes decision support systems easier to use and understand.

The Arriclides architecture does not only allow the combination of different clinical knowledge models; it also offers generic support for a lot of non-functional requirements such as scalability, performance and integration. As a result, new knowledge models can be included more easily into the architecture because implementers can focus on the core syntax and semantics of their model. The result is a decision support system that is extensible, allowing clinical institutions to gradually include new knowledge models without having to migrate existing knowledge or to redo the integration.

This paper is structured as follows. After introducing clinical decision support in section 2, section 3 describes the different clinical knowledge models that are supported by the Arriclides architecture. The high-level architecture itself is described in section 4. The architecture of the execution environment and the rule-based processing unit are described in more detail in section 5 and 6 respectively. A prototype implementation is presented and evaluated in section 7. Related work is discussed in section 8; we state our conclusions in section 9.

2 Clinical decision support

Clinical practice guidelines (CPGs) are a powerful method for improving the standardization and the quality of medical care [5]. The purpose of such guidelines is to support care providers in diagnosing or treating problems. CPGs describe the interaction between a patient, care providers and the environment in light of specific clinical circumstances. CPGs are tools for encouraging best practices in clinical care; they contribute to improving safety, quality and cost effectiveness. Professional organizations, government agencies and healthcare institutions have published a plethora of clinical guidelines.

Care providers rarely have the time to keep up with all state-of-the-art guidelines. Systems aiding physicians in following particular guidelines are called clinical decision support systems (CDSSs). Studies have shown that guidelines best affect clinician behavior if they deliver patient-specific advice during the clinical encounter [13]. For guidelines to be delivered at the point of care through CDSSs, they must be represented in a computer-interpretable format to enables automatic inference based on available patient data. However, this is far from the only requirement for an effective CDSS.

We have defined a set of core requirements that must be met by CDSSs. (A similar set of requirements for a sharable guideline representation format has been suggested by Boxwala *et al.* [1]). We briefly explain the rationale for these requirements, as they have been the main drivers while creating our architecture.

- Knowledge must be represented in a human-understandable but computer-interpretable manner.
- A clinical decision support architecture should support the entire life cycle of guidelines [2], including steps like authoring, dissemination, application and evaluation.
- A clinical decision support architecture should support different types of knowledge: clinical knowledge can be divided into different types (a) based on the stage of the care process it is related to (e.g. diagnostic, treatment, prognosis), and (b) based on the medical domain it is used in. These are but two examples.
- A clinical decision support architecture should support different types of decision support. A decision support system can be reactive, proactive or retrospective and can manifest itself in a number of ways, e.g. as documentation, alerts and reminders, diagnostic assistance, therapy critiquing and planning, prescribing assistance, reasoning, etc.
- The clinical knowledge models must offer expressive power to allow the definition of all kinds of knowledge in an unambiguous way. This is the most optimal way to tackle complexity, for example by using constructs that are closely related to problem domain.
- The clinical decision support architecture should be manageable, meaning that the configuration and administration should be similar for all different types of knowledge and support.
- The clinical decision support system must be integrated with existing data repositories, to be able to automatically provide patient-specific advice.
- The clinical decision support architecture should be extensible as research about clinical knowledge models is still continuing and new standards for representing patient data or for interacting with existing clinical services might still arise. The inclusion of new process models and/or standards must be straightforward.
- The clinical decision support architecture should support traceability: care providers should always be capable of requesting why certain recommendations were made by the decision support system. They always remain in charge, because they can reject or override recommendations. All information about these recommendations should be stored so that this information can be used for evaluation.
- Clinical knowledge should be shareable between different organizations. Local adaptation of this knowledge to compensate for variations in practice settings, availability of equipment and medication, local policies, etc. should be supported as well.
- A clinical decision support architecture should enable portability in different clinical settings, on different platforms using different implementations, etc.

Notice that these core requirements can be classified into a group that is focused on the knowledge models themselves, and a group that addresses the key properties of the overall architecture.

Besides these core requirements, a clinical decision support system has to address additional requirements that are necessary for successful deployment and exploitation of the decision support applications, such as scalability, performance, security, reliability, etc.

The next section focuses on the knowledge models that are used to encode and represent clinical knowledge. This will on the one hand further exemplify the problem domain; on the other hand this will also illustrate some of the above mentioned core requirements, especially those that characterize the knowledge models that must be expressive, diverse and manageable.

3 Clinical knowledge models

In the last decade, several groups have defined models for representing computer-interpretable guidelines (CIGs) (Arden Syntax, Asbru, EON, GLIF, GUIDE, PRESTIGE, PRODIGY, PRO*forma*, SAGE, etc.) [14, 15]. A different approach, based on their specific interest and expertise, has resulted in different clinical knowledge models (also called process models). Since much effort goes into creating guidelines in a computer-interpretable format, it is desirable that different medical institutions and software systems can share them, explaining the need for *standardized* clinical knowledge models. Several comparison studies [14, 15] have identified similarities and differences between these different clinical knowledge models. Attempts to create one standardized representation format have not succeeded so far.

While one model might be the most appropriate candidate for modeling a certain type of clinical knowledge in a specific setting, using it to encode a totally different type of

decision support might be difficult without making the resulting knowledge too complex and hard to understand. For example, because the decision process when diagnosing patients is often less predictable than when treating patients that have already been diagnosed with a particular disease, different building blocks should be offered to clinical experts encoding this knowledge in a computer-interpretable form. By using constructs that are closely related to the problem domain, the useability of a clinical model for encoding a certain type of clinical knowledge can be greatly increased. Domain experts can more easily understand a certain knowledge model if the building blocks that should be used are closely related to the problem at hand. This usually leads to higher productivity, quality and maintainability.

Therefore, we believe that having a limited set of heterogenous knowledge models, each used for modeling a certain type of clinical knowledge or decision support, might be the best way to keep the complexity of the clinical knowledge under control. Each process model could be seen as a domain-specific language targeted to a particular type of decision support. By allowing the coexistence or even interaction between different process models, clinical experts can choose the most suitable model when encoding a certain type of knowledge in a specific setting.

In our current prototype implementation we support four process models: clinical workflow, clinical pathways, validation rules and PRO*forma*. We selected these because we believe they can be seen as representatives for some important types of clinical decision support. Clinical workflow can be used to control the execution of low-level clinical protocols. Clinical pathways allow the description of the possibly long-term treatment of diseases, across the boundaries of a single clinical institution. Validation rules allow the use of rule-based, declarative knowledge. And finally, by also including an existing process model like PRO*forma*, we prove that clinical institutions who have already invested in a particular knowledge model can integrate their existing knowledge while at the same time opening up their clinical decision support system to other knowledge models.

Although we believe that these four models already represent a large part of the different clinical knowledge models currently in use, we do not claim that we support all types of clinical knowledge and decision support. We believe that other models might be required in order to support other types like diagnostic assistance and reasoning. That is why extensibility of the clinical knowledge models supported by our architecture is of the utmost importance.

3.1 Clinical workflow

Care protocols can be seen as a sequence of nursing tasks that are related to each other and where the choice of which task to perform next might be influenced by the results of previous tasks. These kind of protocols can best be designed using clinical workflow. Workflow systems are already being used extensively in a lot of different sectors. Workflow processes are modeled using flow charts where each task is represented as a node. Nodes can be linked and there is support for parallelism, synchronization, choice, cycling, etc.

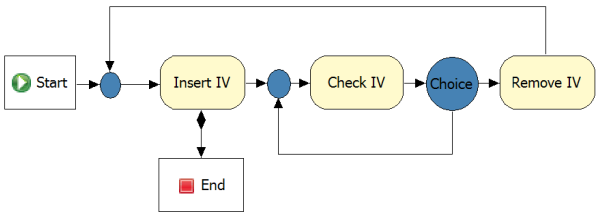

Figure 1. A simple clinical workflow example: IV Protocol

Our clinical workflow model is an extension of a generic workflow model, improved for use in a clinical context. For example, there is support for patient states, deviation from the proposed flow, etc. We support a large part of the workflow patterns as defined by van der Aalst [19], and our workflow model can be extended to support new types of nodes.

Figure 1 is an example of a protocol to prevent intravenous phlebitis, which is inflammation of a vein due to the presence of an intravenous catheter (IV). After inserting the IV, the patient is checked regularly for signs of inflammation, so that the IV can be removed and reinserted if necessary.

3.2 Clinical pathways

Clinical pathways (CPs) are multidisciplinary plans of best clinical practice for specified groups of patients with a particular diagnosis that aid the coordination and delivery of high quality care. These clinical pathways may extend to other healthcare facilities and may even span a lifetime of care in chronic diseases. A patient-centric, process-oriented approach to healthcare is expected not only to improve the quality and safety of care, but will also enhance the cost efficiency because a better planning will eliminate unnecessary wait periods, sub-optimal resource utilization and avoidable duplication of tests.

Most clinical pathways defined nowadays are in a textual form. The format varies from a pure textual description of the best practices, to time-task charts where the columns represent the different time periods and the cells contain the tasks that should be executed at those points in time, grouped into different categories (e.g. nursing, kinesitherapy, diet, etc.).

Our clinical pathway model allows the specification

	Day 1		Day 2
Nursing			
	IV Protocol		IV Protocol
	Measure blood pressure		Measure blood pressure
	Measure temperature		Measure temperature
Examinations			
	Blood examination		Hemoculture
	Hemoculture	if T >= 38.5°C	Ionogram
	RX Thorax	if age > 50years and abdomen	Left colonoscopy
Medication			
	Tarivid IV		Tarivid IV

Figure 2. Part of a clinical pathway for the treatment of gastroenteritis

If
the patient's gender is : Female
and the patient is pregnant
and ordering : X-ray scan

Then
Alert the care provider :
Pregnant females should not receive an X-ray scan.

Figure 3. A sample validation rule for X-ray scans

of clinical pathways using a computer-interpretable time-task matrix. Clinical pathways are divided into different episodes, and contain all the tasks that should be performed during that episode. Goals and patient outcomes can be linked to these episodes. The model also offers advanced support for planning (using time constraints), financial management and variance analysis.

Figure 2 shows a part of the clinical pathway for the treatment of gastroenteritis. The two columns represent the first two days of the treatment. All tasks that must be performed on these days are shown in the appropriate category. A condition is associated with a task if that task should only be recommended under certain circumstances (e.g. a blood culture on day one is only recommended if the temperature of the patient is larger than or equal to 38.5°C).

3.3 Validation rules

Care providers have to fill in a lot of forms during the treatment of patients, and since every mistake in this context can be life-threatening, all input should be validated. We present a rule-based model for validation, where clinical experts can declaratively define validation rules, using a natural language syntax. Whenever abnormalities are detected, different types of alerts and reminders could be used to inform the appropriate care provider(s). We support both specific and generic validation rules, where the specific rules are used for validating the input of one specific form only and generic can be used for validating data coming from all kinds of sources.

Figure 3 shows an example of a generic validation rule that is used to prevent X-ray scans on pregnant females to avoid exposing the baby to radiation. The care provider is suggested to use an ultrasound or MRI instead. This validation rule consists of three conditions and one action. Note that the doctor can ignore this warning (but could be forced to clarify his decision) if he believes the X-ray is necessary anyway.

3.4 *PRO*forma

To prove that our architecture supports the inclusion of existing clinical knowledge models, we added support for PRO*forma* [6], a clinical knowledge model developed at Cancer Research UK. PRO*forma* uses constraint satisfaction graphs for specifying clinical guidelines. This is similar to the flow charts used in clinical workflow.

It is important to know that it must be relatively easy to combine these models such that clinical experts can choose the most appropriate model for representing their clinical knowledge. For example, the clinical pathway model can include validation rules for validating user input and clinical workflow for sequencing certain tasks within one episode.

Although there exist many approaches for representing clinical knowledge, the design and implementation of good and useful decision support systems that will last and evolve are still active areas of research [16]. In the next sections, we present Arriclides, an open architecture supporting the integration of different knowledge models into on clinical decision support system.

4 High-level architecture

The Arriclides architecture offers support for the full life cycle of clinical knowledge models, from authoring and dissemination to local adaptation, usage and evaluation. We provide an architecture that allows the specification, distribution, execution and analysis of knowledge. This enables the use of clinical knowledge models in providing clinical decision support to care providers.

This high-level architecture is similar to some existing propositions in the domain of clinical decision support and could be seen as a reference architecture. The four most important components in the high-level architecture are shown in Figure 4:

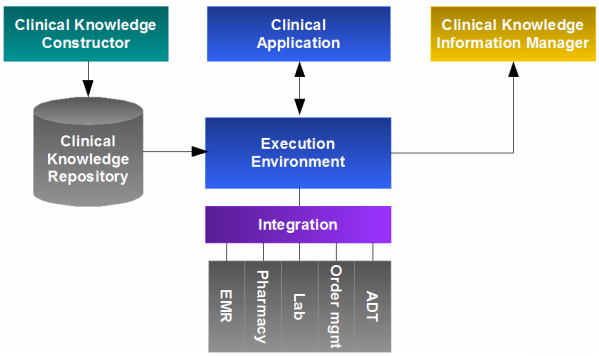

Figure 4. High-level architecture

- **Clinical Knowledge Constructor**: allows clinical experts to create new clinical knowledge using any of the supported knowledge models. It supports validation, simulation, etc.

- **Clinical Knowledge Repository**: All clinical knowledge is stored in a repository. It supports distribution, versioning, local adaptation, etc.

- **Execution Environment**: Decision support must be integrated into the existing runtime environment used by care providers when treating patients. This existing infrastructure could already contain components like an order entry management system, an electronic medical record (EMR), a patient admissions/discharge/transfer (ADT) system, etc. The decision support execution environment must be integrated ensuring it is invoked whenever decision support is necessary and it should have access to existing services and data repositories. The graphical user interface (GUI) of the clinical application that is used by the care providers typically needs to be extended as well.

- **Clinical Knowledge Information Manager**: Information collected during the execution of decision support processes can be analyzed (typically off-line) and possibly used for improving existing processes.

For the remainder of this paper, we focus on the architecture of the execution environment, since it is the most challenging, innovative and complex part of the Arriclides architecture.

5 Execution Environment

Patient-specific decision support can be achieved by creating a decision support engine that is capable of executing a model for a specific patient (also called enactment of the model). Most existing clinical knowledge models have created their own execution engine [17, 21]. But this has lead to different engines capable of executing only one specific model, and interaction between these engines is difficult if not impossible. Moreover, these engines are all faced with similar problems such as integration, persistence, scalability, etc.

We present an execution architecture that allows the combination of or even interaction between different process models. Support for different knowledge models can easily be plugged in, making it an open and extensible architecture. This means that a clinical institution that wants to add decision support to its environment is not forced to select only one clinical knowledge model, but can choose a set of decision support models that best suit their needs, preventing vendor lock-in.

The description of some clinical knowledge in a certain knowledge model is called a *process*. For example, clinical protocols are expressed as workflow processes, and clinical pathway processes are created for specifying the best practice for patients with a particular diagnosis. A care provider that wants to apply this knowledge to a specific patient should request the execution of such a process. At that point, a *process instance* is created, containing all patient-specific runtime information about the state of the patient in that process.

When executing a process instance, the decision support system can invoke external services to perform a specific task. Typical examples are a recommendation to a care provider to perform some clinical task, a notification, querying for certain patient data, scheduling an appointment, etc. Some work could have to be performed by a human actor, while other might be executed automatically. A *work item instance* is an abstract representation of such a unit of work containing all information needed for executing the task. This allows us to reuse the same processes in different clinical settings. When integrating the decision support architecture into an existing runtime environment, implementers must map these work items to invocations of existing services.

Figure 5 shows the main components in the execution environment architecture:

- The **Process Manager** handles the communication between a decision support requester and the decision support engine. It offers an API for starting processes for a specific patient and manipulating existing process instances.

- The decision support system should be notified of events that might influence the execution of (especially long-running) process instances. The **Event Manager** offers an API for signaling such events and makes sure that all relevant process instances are notified.

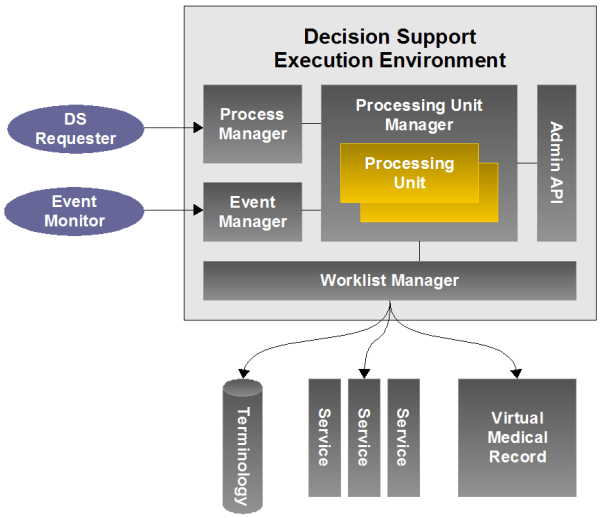

Figure 5. Architecture of the decision support execution environment

- The decision support system is responsible for selecting tasks (work items) to be executed, and when to execute these tasks, but it does not know how to execute them. Whenever the decision support system requests the execution of some work item, the **Worklist Manager** makes sure the work is executed. It is responsible for finding the actor or external service responsible for executing the work, delegating the work, and notifying the decision support system when the work has been completed or aborted.

- The **Processing Unit Manager** is the component that is responsible for actually applying the decision support knowledge encoded in processes. It is notified of all changes in the environment and decides when to execute process instances that might be influenced by those changes (e.g. instantly or in batch at night). For each of the supported clinical process models, a **processing unit** must be plugged in, capable of interpreting processes of the specific type. Whenever a process instance must be reevaluated, the processing unit must deduce the new state of the process instance based on the previous state and on other external data, using the clinical knowledge encoded in the process. At the same time it can generate work item instances that must be performed. (These are delegated to the worklist manager.)

Processing units contain the actual logic that is needed for executing a clinical knowledge model. In order to add support for a particular knowledge model, a processing unit must be implemented capable of interpreting that model. The processing unit manager offers a *pluggable* architecture where new processing units can be registered. The processing unit is responsible for applying processes of a certain knowledge model to patients, by modifying the state of the process instance representing the state of the patient in a process, and by requesting the execution of work item instances if certain external services need to be invoked. The architecture makes sure that the appropriate processing unit is activated whenever necessary, that this unit has access to the most up to date information and that the work items it generates are executed by the appropriate services.

To allow inter-operability between different models and sharing of clinical knowledge across institutions, all patient data referenced in clinical processes must be defined using a standardized patient data model that is coupled to standard terminologies. We use the concept of a (simplified) virtual medical record (VMR) [11] as a virtual repository containing all data necessary for decision support. When integrating the decision support architecture into an existing runtime environment, implementers must map these concepts to their local electronic medical records (EMR) representation and repositories.

Our architecture offers generic support for a lot of requirements, so that implementers of a clinical knowledge model can focus on defining the operational semantics of their model and should not be concerned with the implementation details of requirements. We will not discuss all supporting components in full detail. We limit this description to a summary of the architectural tactics that have been selected and implemented to meet these requirements:

- Scalability: The number of patients treated by one clinical institution can become very large and the amount of clinical knowledge is ever growing. By distributing all these patients and knowledge over different decision support systems, the clinical decision support architecture remains scalable with respect to the number or patients, the amount of patient data, the amount of clinical knowledge, the number of care providers requesting decision support, etc.

- Performance: To make sure that care providers do not have to spend time waiting for decision support responses, the architecture embeds information caching, parallel execution, asynchronous (non-blocking) communication and reuse of previous results.

- Monitoring: The Admin API helps the administrator in maintaining the decision support system. This includes configuration, inspecting or modifying the state of existing process instances and work item instances, retrieving performance measurements, etc. The API offered to administrators is similar for all process models.

- Security: the architecture allows the application of generic security solutions for problems like authentication of care providers requesting decision support, authorization so that only care providers who have the right to execute certain processes can do so, confidentiality and integrity to protect the privacy of the patient.
- Integration: by integrating the architecture with existing data repositories and services, patient-specific advice can be provided at the point of care. Integration is based on the use of a standardized patient data model and terminology. In order to have fully automated decision support, the decision support system must also be notified of relevant events in the surrounding systems. The effort of integrating the architecture in an existing runtime environment is reused across all clinical knowledge models.
- Persistence: the current state of process instances is stored persistently, so that the runtime state of a certain patient in a process does not remain in memory if this information is currently no longer needed, and to allow recovery in case of failure. The architecture automatically retrieves the last known state before evaluating a process instance and persists the changes afterwards. There is support for transactions so that a process instance always remains in a consistent state.

A new process model can easily be included in our decision support architecture by plugging in a new processing unit that is capable of interpreting the model. Because the architecture already solves a lot of non-functional requirement, implementers of processing units should only be concerned with defining the operational semantics of their knowledge model. Different techniques could be used for implementing such processing units, for example:

- The operational semantics of the process model could be encoded using a standard (functional) programming language.
- Implementers could use other technologies to aid in the execution of their model. Typical examples are mathematical models like graph theory and petri nets, generic reasoning engines, etc.
- Light-weight versions of execution engines for existing knowledge models could be integrated and plugged into our architecture.

In the next section, we present our approach to provide developers of process models with an elegant implementation strategy to plug in support for their knowledge model. In this approach, rules are used to describe the operational semantics of a specific model, and a rule engine is used inside the processing unit to execute these rules (operational semantics).

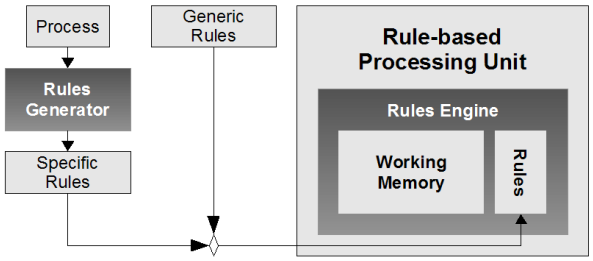

Figure 6. Rule-based processing unit

6 Rule-based processing unit

Most execution engines for existing clinical knowledge models hard code the semantics of the process model into the engine code. This is not a very natural way of defining the operational semantics of a process model, and it limits the extensibility and adaptability of the processing unit and the process model itself.

We have created a *rule-based processing unit* that acts as a template that implementers can use to plug in support for their process model. This processing unit requires implementers to specify the operational semantics of their model using declarative rules. For each of the building blocks offered by the model, one or more rules should specify how that building block should be treated when executing processes of that type. For example, when looking at the clinical pathway model, rules are used to define what should happen when episodes are started, when tasks are executed, when patient goals are reached, etc. This is a more natural way of implementing support for a processing unit since the logic for executing a process is separated from other implementation issues.

A rule engine is embedded inside the rule-based processing unit that will use these rules when executing processes of that type. This means that implementers that want to plug in support for their clinical knowledge model can do so by encoding the semantics of their model using declarative rules and by registering a rule-based processing unit that will use these rules for executing their processes.

Figure 6 shows the architecture of the rule-based processing unit. It contains a rule engine, consisting of a working memory that holds the current state of the process instance and the patient and rules that describe how the process should be executed. We use two types of rules: *generic and specific rules*. Generic rules are valid for all processes of a certain clinical knowledge model and describe how the building blocks of that process model should be treated during execution. For example, the parallel split used in the workflow model to create concurrent branches is associated with a generic rule that states that all outgoing branches of the parallel split should be triggered when this node is

reached during execution.

Specific rules are used for evaluating expressions inside processes. Clinical models often use an expression language to reason about data or to express logical expressions. For example, the clinical workflow model allows the use of exclusive choice nodes when the care provider is confronted with a set of alternatives and one and exactly one of these alternatives should be selected. An expression could be linked to each of the branches that specifies when that specific branch should be selected. This expression should be evaluated at runtime. GELLO [18] is an example of such an expression language that allows the construction of standardized, platform-independent expressions. The rule engine can be used to evaluate these expressions by creating rule templates in which the expressions are inserted. When a rule constructed using these templates is evaluated, the expression inside the rule is automatically evaluated as well. We call these rules specific rules because they are used to evaluate expressions of one specific process. A *Rules Generator* is used to create a specific rule for each expression that is encountered in a process. For example, one specific rule is generated for each outgoing branch of an exclusive choice, triggering that specific outgoing branch if the constraint linked to that branch is satisfied.

The set of rules that is created by combining the generic and the specific rules of a process then contains all knowledge that is needed to apply that process to patients. Based on the current state of a patient in a process instance, (which is inserted in the working memory of the rule engine,) the rules are used to modify the state of the process instance (if appropriate) and request the execution of work item instances (if certain external services need to be invoked). These external services could be used to create alerts, recommendations, etc. By creating these alerts and recommendations, the clinical decision support can obviously aid care providers in coordinating and delivering high quality care.

7 Prototype and evaluation

We have created a prototype that implements the Arriclides architecture as defined in sections 4, 5 and 6. Our prototype allows clinical experts to define clinical knowledge using the knowledge constructor that is then stored in the repository and can later be used in the execution environment to generate patient-specific advice. Although the knowledge that is required for analyzing decision support processes is already stored at runtime, we have not yet developed a clinical knowledge information manager capable of analyzing this information.

To further demonstrate the capabilities of our prototype, we first present two scenarios that are supported by the prototype: Clinical experts could define validation rules that are used to prevent medication errors by checking for unwanted medication interactions. Similarly, validation rules could be used in combination with input forms to check the validity of the data entered by care providers. Notifications can be used to inform the care providers in case of any problems.

A second scenario describes the typical use of clinical pathways and workflow: If a patient is diagnosed with a particular disease, and a clinical pathway has been defined for that disease, the patient can be assigned to that pathway. The clinical decision support system then creates recommendations that present the most optimal treatment based on available patient data. These recommendations could also include workflows to coordinate long-running nursing protocols. Whenever new data about the patient is available, the status of the clinical pathway is updated and the decision support system checks whether new actions should be recommended. This process continues until the treatment ends or the clinical pathway is no longer applicable.

In the next paragraphs, we discuss the implementation of the knowledge constructor and the execution environment in more detail and describe some measurements that were performed to determine the performance characteristics of our execution environment.

7.1 The knowledge constructor and repository

The knowledge constructor has been based on the Eclipse Rich Client Platform [4], because it provides a pluggable architecture and offers support for creating different types of advanced (graphical) editors. We extended the Eclipse user interface with specialized views and editors for creating clinical knowledge for each of the supported clinical knowledge models.

The constructor can be divided into different parts: A first set of editors is used to define and/or import terminology used in the hospital as well as the structure of the patient data. The patient data model is based on the HL7 Reference Information Model (RIM) [9]. We have extended this model with the concept of a *Recommendation* to represent suggestions made by the decision support system. Recommendations can be accepted or rejected by care providers.

Clinical experts can then start defining clinical knowledge using any of the provided knowledge models. The flow charts for expressing clinical workflow or PRO*forma* processes are based on the Graphical Editing Framework (GEF) [7] and include a drag-and-drop editor, validation and a simulator. Validation rules can be created with a user-friendly editor using a natural language format based on the ILOG JRules [12] graphical editor. The clinical pathways are constructed as time-task-matrices based on advanced Java Swing tables.

The clinical knowledge created using the constructor is

stored as XML files in the file system. Knowledge can be uploaded to a distributed clinical knowledge repository that has been implemented using Java Enterprise Edition (Java EE) on a JBoss application server and a MySQL database.

7.2 The execution environment

Two versions of the execution environment as presented previously in Figure 5 have been implemented:

- A fully-functional, distributed implementation allows decision support at server side. It supports the execution of all the four knowledge models that have been driving the Arriclides architecture. The execution environment is integrated with services that allow the creation of patient-specific recommendations and sending notifications to care providers. All patient data needed during decision support evaluation can be accessed from a VMR. The distributed implementation is based on Java EE and is deployed on a JBoss application server. For persistence it uses Hibernate backed up by a MySQL database.
- The light-weight implementation allows the use of reactive decision support at client side: it can be used to provide input validation and alert generation. It has access to data at client side and can generate notifications to the user. It uses Java Standard Edition (Java SE) and must integrated with the client application.

Rule-based processing units have been created to support the execution of clinical workflow, clinical pathways, validation rules and PRO*forma*. Two different rule engines were used during the implementation: Drools v2.4 [3], an open-source rule engine in Java and ILOG JRules v4.6 [12], a commercial Java rule engine.

7.3 Performance of the execution environment

In many occasions, decision support results should be available as soon as possible. For example, the detection of medication interactions should be signalled immediately to prevent medication errors. The valuable time of care providers should also not be wasted by requiring them to actively wait for the decision support system.

Therefor, we have measured the response time of the execution environment, as shown in Table 1. For this purpose, we used the IV Protocol as presented in section 3.1 as an example. We have determined the response time by measuring how long it takes the workflow processing unit to determine whether the IV should be reinserted or not after providing the results of the IV check. This response time can be divided into three distinct timings:

Table 1. Performance measurements

	JRules	**Drools**
Time to load process	515 ms	1891 ms
Time to load data	12 ms	31 ms
Time to execute process	15 ms	16 ms

- The time consumed for loading the process. This includes retrieving the process from the repository and allowing the processing unit to parse it.
- The time consumed for loading the data needed during the execution of the process instance. This data includes the previous state of the process instance and relevant patient data.
- The time consumed for executing the process. The processing unit deduces the next state of the process instance and generates work item instances representing the invocation of external services.

We have measured the performance using both the JRules and Drools rule engine. Results show that loading the data and executing the process only takes a few tens of milliseconds, while the time for loading the process is more around the order of one second. But since process definitions usually are not changed that often, they can easily be cached so the loading should only occur once, e.g. at the start-up of the application. Therefor, the results of the decision support system can be presented almost instantaneously.

8 Related work

A lot of research has already been performed in the area of clinical knowledge models. We briefly compare our approach with the main alternative in the domain of integrating clinical decision support systems: creating a uniform meta-model for all relevant clinical knowledge models. We also sketch how our work compares to decision support systems for business process management, that also tackle the problem of integrating an execution engine in an existing runtime environment. Finally we refer to relevant standardization efforts in the area of workflow management.

As introduced and summarized in section 3, a lot of different clinical knowledge models have already been defined by different research groups. Comparison studies [14, 15] have revealed that some of these models offer quite similar building blocks. To allow the sharing of clinical knowledge encoded in different formats across clinical institutions, D. Wang has created GESDOR [20], an execution engine supporting the execution of two different clinical knowledge models, namely GLIF and a variant of PRO*forma*. He does

so by mapping both models to a generalized guideline ontology. This strategy assumes that a generalized model can be defined to support a diversity of useful (and often quite different) knowledge models. Defining such a generalized ontology was possible for Wang because the two models that have been addressed share a similar structure. We believe that it is practically impossible to create and maintain a generic ontology capable of expressing all different (yet relevant) types of knowledge and decision support. Notice that one has to manage the complexity of the generic ontology and the mapping of the different knowledge models to this generic ontology. In fact, our architecture eliminates the need for such a generic ontology.

In the area of business process management (BPM), process engines have been used extensively to orchestrate the interaction between different business actors. JBoss jBPM - a platform capable of executing workflow processes - offers support for different workflow languages, like WS-BPEL [10], jPDL, and Pageflow. They use the term *Graph Oriented Programming* [8] to identify all languages based on the execution of a graph. Our architecture extends this approach by not only supporting graph-based workflow languages but many other types of knowledge models.

Within the context of standardization of interfaces, the Workflow Management Coalition (WfMC) [22] has defined a reference model and standardized interfaces for workflow management systems. If different workflow products by different vendors support these interfaces, these products could easily be integrated and/or replaced, preventing vendor lock-in. Our decision support architecture defines similar interfaces in order to create a standardized API for using clinical decision support, regardless of which process model is used to encode the clinical knowledge.

9 Conclusion

In this paper, we have discussed Arriclides, an open architecture that integrates a variety of clinical decision support models. The overall architecture is similar to many propositions in the domain, and it could act as a reference architecture for clinical decision support systems. The execution environment that lies at the heart of Arriclides is an essential differentiator of our work, as it supports heterogeneity. We have implemented a prototype of Arriclides and initial evaluations show acceptable performance. Scalability can be achieved while preserving diversity in the underpinning clinical decision support models.

Acknowledgment

This paper is based on research performed in the context of the *"Healthcare networks of the 21st century: a Clinical Pathway based approach"* project from the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References

[1] A. Boxwala, M. Peleg, R. Greenes, E. Shortliffe, and V. Patel. Functional requirements for a representation for sharable guidelines, 2000.
[2] G. Browman. The clinical practice guideline life cycle.
[3] Drools, http://www.jboss.com/products/rules.
[4] Eclipse rich client platform, http://wiki.eclipse.org/index.php/rich_client_platform.
[5] M. Field and K. Lohr. *Guidelines for clinical practice: from development to use*. National Academy Press, 1992.
[6] J. Fox and A. Rahmanzadeh. Disseminating medical knowledge: the proforma approach. *Artificial Intelligence in Medicine*, 14:157–181, 1998.
[7] Graphical editing framework, http://www.eclipse.org/gef.
[8] Graph oriented programming, jbpm userguide v3.
[9] Hl7 reference information model, http://www.hl7.org.
[10] IBM. Bpel4ws v1.1 specification.
[11] P. Johnson, S. Tu, and M. Musen. A virtual medical record for guideline-based decision support. *AMIA*, 2001.
[12] Ilog jrules, http://www.ilog.com/products/jrules.
[13] J. Overhage and W. Tierney. A randomized trial of corollary orders to prevent errors of omission. *J Am Med Inform Assoc*, 4(5):364–375, 1997.
[14] M. Peleg, S. Tu, and J. Bury. Comparing models of data and knowledge for guideline-based decision support: A case-study approach, 2002. Report SMI-2002-0923.
[15] M. Peleg, S. Tu, J. Bury, P. Ciccarese, J. Fox, and R. Greenes. Comparing computer-interpretable guideline models: a case-study approach. *J Am Med Inform Assoc*, 10:52–68, 2003.
[16] M. Peleg, S. Tu, R. Haux, and C. Kulikowski. *2006 IMIA Yearbook of Medical Informatics: Decision Support, Knowledge Representation and Management in Medicine*. Schattauer, 2006.
[17] P. Ram, D. Berg, and S. Tu. Executing clinical practice guidelines using the sage execution engine. *Medinfo*, 2004.
[18] M. Sordo, A. Boxwala, O. Ogunyemi, and R. Greenes. Description and status update on gello: a proposed standardized object-oriented expression language for clinical decision support. *Medinfo*, 11:164–8, 2004.
[19] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.
[20] D. Wang, M. Peleg, D. Bu, M. Cantor, G. Landesberg, E. Lunenfeld, S. Tu, G. Kaiser, G. Hripcsak, V. Patel, and E. Shortliffe. Gesdor: A generic execution model for sharing of computer-interpretable clinical practice guidelines. *Proc AMIA Symp*, 2003.
[21] D. Wang, M. Peleg, S. Tu, A. Boxwala, O. Ogunyemi, Q. Zeng, R. Greenes, V. Patel, and E. Shortliffe. Design and implementation of the glif3 guideline execution engine. *J Biomed Inform*, 37(5):305–18, 2004.
[22] The workflow management coalition, http://www.wfmc.org.