A Metamodeling Approach to the Usability, Correctness, and Adaptability of Workflow Software Systems for Healthcare

Author1, Author2, Author3

Institution Name, Country
email1, email2, email3

Abstract.

1 Introduction

Rising costs, ageing populations and increased expectations are making the current healthcare systems in the developed world unsustainable. Information technology has the potential to support healthcare but its application to support the continuum of care has not nearly reached its full potential. Barriers include the proliferation of systemseven within one hospital – which do not support interoperability; the fact that systems must be highly customized to adequately serve local situations, (usually a time consuming, and error prone process); the fact that change in health care is the only constant as adaptations to deal with such things as new medications, changing protocols and management strategies are constantly in need of being made; the fact that software engineering itself for such safety critical systems as healthcare needs new strategies to ensure that systems behave correctly in every possible scenario; and finally, the fact that the healthcare process itself is often a team process involving many players, including family physicians, nurses, therapists of various kinds, specialists, lab technicians, managers of both hospital and community based programs, all with different needs from IT and all with different and often limited ability to handle complex technology. The active participation of the patient (and his family) in the management of his own health is becoming a critical issue as the cost of chronic diseases is quickly out pacing the resources that can be directed to healthcare.

MDSE is an emerging and promising methodology for software development, targeting challenges in software engineering relating to productivity, flexibility and reliability of systems. The construction of various kinds of models (eg. blueprints, mockups etc) is a well-known approach in the more traditional engineering fields; these models are used as artefacts to enable engineers to describe designs and validate whether a proposed design has desired qualitative and quantitative properties. We propose to use MDSE in an analogous manner for the development of reliable and robust workflow software systems supporting the diverse and often safety critical requirements of healthcare process in both primary care and community care environments.

While the use of executable graphical languages and domain specific modelling languages greatly reduces the possibility of coding errors, these techniques do not follow the metamodelling approach and may lead to impaired links between the workflow model and the generated code. Many different MDSE technologies automatically generate code from models [ADD Tools, REFERENCES]: these technologies are particularly suited to specifying the structural aspects of software systems; generally, whereas the actual behavior is programmed manually. Some technologies for behavioural modelling in MDSE exist ([2], [1]), but current approaches are often at a low level of abstraction and lack domain concepts for specifying behavior [9].

A collaborative group of researchers in Norway and Canada have been working on various issues relating to these problems. We [22], [19], [11] proposed a formal approach to workflow modelling based on the Diagram Predicate Framework (DPF) [18], [21] which provides a formalism of (meta) modelling and model transformations based on category theory and graph transformations ([5], [7], [Ehrig etal]) We [24] extended the formal foundation of DPF to define (static) semantics for timed and compensable workflow models and defined the dynamic semantics of models by a transition system where the states are instances and transitions are applications of transformation rules, and showed how to exploit reduction methods and sweepline methods to model check complex workflows [10]. We developed a domain specific language to expedite workflow system development [17], [16] and began the development of a user-friendly interface to allow the health practitioner to determine the correctness of behavioral properties of a healthcare workflow protocol [20].

2 A Metamodeling Approach to healthcare workflows

In [15] we discussed the implementation of a user-friendly tool to aid clinicians and patients in the workflow process implied in a clinical practice guideline with an accompanying module that gives the patient a user friendly interface for self management of lifestyle attributes which caused the patient to be at risk. The patient can input data for such lifestyle attributes such as, exercise, smoking, intake of fruits and vegetables and record such attributes as weight and blood pressure. The web-based tool would allow both the patient and the clinician to view summary data on lifestyle parameters between visits [give some pictures here!!] and provide calendar views of past activities, future appointments, etc.

The PhD thesis of [3] promoted a metamodelling approach to the development of workflow processes in healthcare settings, which incorporated the concept of monitor. Many healthcare processes involve numerous stakeholders with different requirements, and the user becomes a critical part of the healthcare workflow process whether it be the physician, a specialist, a lab tech or the patient, in situations where management of lifestyle parameters is a critical component of the process. We are now researching a metamodelling approach to workflows which incorporates the concepts of stakeholder and monitor and provides user friendly interfaces for a variety of users. Such processes are safety

critical and must be adapted to particular clinical setting and changed/updated frequently due to changing guidelines, medications, and patient preferences; thus they serve as an excellent application domain for the development of evolving, adaptable and correct software systems, in general.

In his thesis, Baarah [3], proposed an application framework for care process monitoring that collects and integrates events from event sources, maintains the individual and aggregate state of the care process and populates a metrics datamart to support performance reporting. He presented a UML-style metamodel for the care process monitoring application that had 3 main components: a process model , a performance model and an enterprise model. The process model defines the care process in terms of states to be monitored, resources and rules that specify the transition from state to state as events are received from the enterprise model. The performance model measures how well the goals for the care process are being achieved in terms of metrics computed from the monitored states, and events for the process. Alerts are defined to flag when targets are not being met. No automated implementation of the metamodel was attempted, correctness of the process was investigated only through the use of test scripts, and user interface issues were not considered.

In the thesis, Baraach basically structured a workflow system into 3 components:

- a workflow component;
- a monitoring component this will typically send alerts etc. when a critical event occurs; and,
- a data source component to persist the data that are relevant in the workflow.

We extend this model to include users and user interactions; allowing us to model user interaction as part of the process. The users may interact with:

- part of the workflows
- part of the database
- the monitoring system, that is, users will typically receive alerts from the monitoring system

All interactions occur at specific times. A workflow consists of a network of tasks and specified users are responsible for performing each task. While performing a task a user provides data to the system, typically, this is filling out a standardized web-based form, or some form which provides automatic integration with the appropriate healthcare database.

3 what is a good name???

RABBI – If we look at the workflow you have in the CBMS-Nova we recently submitted, we can make a simple modification to this where we include a sub workflow where the patient is registering his life style information between visits 1 and 2; the tasks should be annotated with a marking to denote the user D doctor, N nurse, P- patient, T-technician etc. So in this workflow the patient will

be responsible for registering the life style information between visit 1 and 2 and the doctor will be responsible for the rest of the workflow. Both the doctor and the patient should have read access to the lifestyle information. The D should have a user interface a bit similar to the one the patient uses i.e. some kind of form, but it can have many more featureseg sending an e-mail to the lab for a lab test.

Based on this we could maybe introduce a task based access model [as described for example in [13]]

For the paper, I think we could incorporate this simple idea with two users of the system and some alerts that get fired in a specific situation. There could be two types of alerts one, less urgent, for example to the patient which reminds the patient if he forgets to enter data on some day and another, which we could call an alarm , which is more urgent which is to both the patient and doctor for example, if the weight gain over fewer than 5 days exceeds 10 pounds (which is indicating to the doctor that the patient is retaining excessive fluids and the doctor should consult that patient immediately)

We should also write about how the user is interacting with the system, how is the input given and the ways data can be read from the system can talk about developing interfaces that give summary data to the patient and doctor and aggregate information to a manager, The idea to use the calendar to represent the time is nice used to represent activities already accomplished and future appointments.

We need schematic of the metamodel; an actual simple workflow model and the various user interfaces appropriate to manager, doctor and patient.

YNGVE we had discussed 2 users do we want the manager interface too???

4 Data aware workflow modelling

The syntax and semantics of the workflow modelling language which we use in our approach may be found in [19,23]. The modelling language is defined using the Diagram Predicate Framework (DPF) [18] and implemented using the DPF Workbench [12]. In DPF, a modelling language is given by a metamodel and a diagrammatic predicate signature (see Fig. 1). The metamodel defines the types and the signature defines the predicates that are used to formulate constraints. A metamodel in DPF consists of an underlying graph, and a set of constraints. We say that a model conforms to (is an instance of) a metamodel if the model's underlying graph is typed by the metamodel's underlying graph, and if the model satisfies the constraints defined in the metamodel. In DPF, the semantics of a (meta)model is given by the set of its instances. DPF supports a multi-level metamodelling hierarchy, in which a model at any level can be regarded the metamodel for models at the level below it.

In the design of our workflow modelling language we have three modelling levels: M2, M1 and M0 (see Fig. 1). The metamodel of our workflow modelling language (which is at level M2) consists of a node Task and an arrow Flow. Simply put, this means that we can define a set of tasks together with the

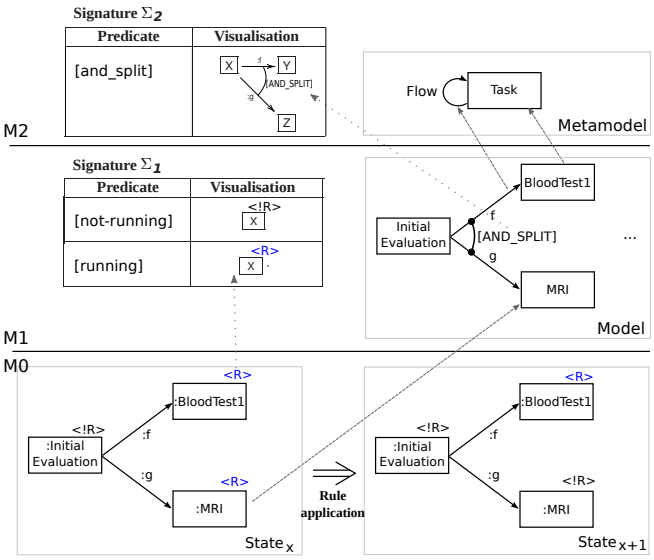

Fig. 1: Workflow modelling hierarchy: the dashed arrows indicate the types of model elements, the dotted arrows indicate the relation between the signatures and the models

flow relations between these tasks. As seen in Fig. 2, our workflow models are diagrammatic models describing in which order specific work tasks should be executed. Each task is represented by a box. If there is an edge $T_1 \to T_2$ starting in task T_1 and ending in task T_2, then task T_1 must be performed before task T_2. Special binary constraints on forks (joins) specify splits (respectively, merges) of workflow branches. In fact, joins and forks could be extended in the standard way to arbitrary triples, quadruples, etc. We have three kinds of splits: `[and_split]`, `[or_split]` or `[xor_split]`, and three kinds of merges: `[and_merge]`, `[xor_merge]` or `[or_merge]`. The meaning of these constraints are as usual: both branches have to be executed in an `[and_split]`; exactly one branch has to be executed in an `[xor_split]` and one or two branches have to be executed in an `[or_split]`. We use $T_\square$ code to specify the details of tasks and flow relations at level M1. Task specification writen in $T_\square$ include data flow and UI definition. Branching conditions for the flow relations associated with `[or_split]` and `[xor_split]` may be specified using $T_\square$ code. Fig. 2 shows some composit tasks such as 'Visit1', 'Visit2', 'CBPM' those are abstractions of subworkflows. The subworkflow for 'Visit1' composit task is shown in Fig. 2. These abstractions are sometime useful in order to reduce the number of repetation.

Given a specific workflow model at level M1 (like the one in Fig. 1) and the predicates `[running]` and `[not-running]` (denoting, respectively, that a task instance is running or not running) collected in a signature Σ_1, we create another modelling language which we use to define *workflow states*, or, equivalently,

instance of a workflow model. The workflow states are located at level M0. We generate states by applying model transformation rules.

5 Proposed User Interface

In this section we present the user interface design through a running example. We modeled a Hypertension management workflow from the guideline of clinical pathways [1]. For workflow modeling we used DERF where one can graphically design a workflow model. Fig. 2 shows the overall model of the Hypertension Management Workflow. Initially patients' blood pressure is measured at the 'Initial BP' Task which may cue the clinical hypertension management procedures if the BP is found to be greater than or equal to 140/90. If the initial BP is found normal (<= 140/90), the workflow terminates. In Fig. 2 patients' Hypertension is managed through investigation and treatment. The clinical procedure (i.e., 'Visit1', 'Visit2', and all other subsequest tasks in In Fig. 2) starts at the doctors' office. Patients with high BP have risk of organ failures and/or other chronic illness. During the first visit at doctors' office ('Visit1') BP is measured twice, an initial assessment is done, and an investigation is started with diagnostic tests. 'Visit2' is scheduled with less than a month time if high BP is found during 'Visit1'; otherwise 'Visit2' is scheduled after one month. During 'Visit2' BP is measured, diagnosis is performed and followup visits are scheduled. The workflow executes 'Ambulatory blood pressure monitoring' ('ABPM') or 'Self home blood pressure monitoring ('SHBPM') if they are available otherwise a Clinical BPM is performed which refers to the 'CBPM' task in the overall workflow model. In the rest of the section we present the new user interface for the execution of the 'Visit1' workflow for a particular workflow instance/case.

Fig. 3 shows 4 windows named 'Workflow Viewer', 'Task Execution Viewer', 'Lookup Viewer', and 'NOVA Browser'. 'Workflow Viewer' shows the list of all available tasks on the right side of the window and whenever a task is executed, it shows the task name on a calendar. A task being executed are shown in the 'Task Execution Viewer'. Inputs required from the end user are shown in branches and the end user is supposed to select a branch and assign a value to it through the 'Lookup Viewer'. The 'Lookup Viewer' helps the end user to input information either by showing relevant values from an ontology or by allowing end user to enter information. Once a task is executed, the information are hierarchically displayed in the 'NOVA Browser'. We introduced 'NOVA Browser' in [14]. Using 'NOVA Browser' user may expand a branch to see detailed information, may use the time travel view to traverse backward/forward and view information. the details Fig. 3 is showing that the user is executing the 'Measure BP' task. Inside the 'Task Execution Viewer' window user provides input to execute the task. This is an alternative way of taking user input instead of using *Forms*. While executing tasks the user has access to historical information, workflow

[1] The Chinook Primary Care Network: `http://www.chinookprimarycarenetwork.ab.ca`

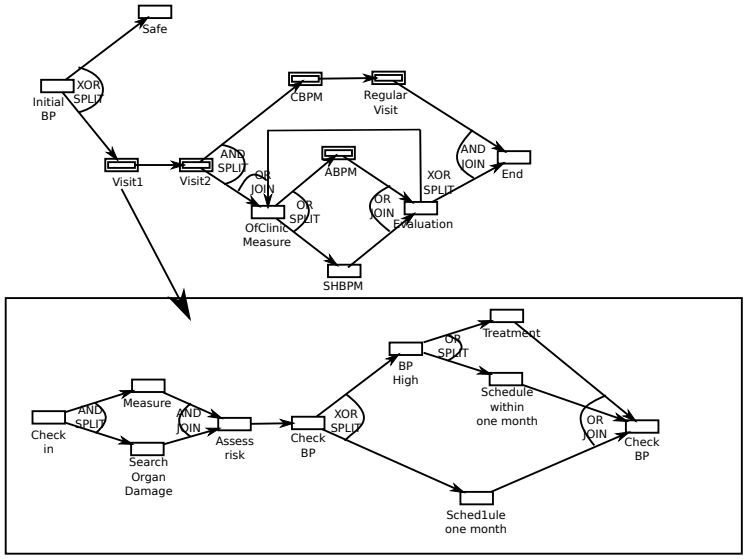

Fig. 2: Hypertension Management Workflow (Overall)

instance; this arrangement provides the user more user-friendly environment to concentrate on care rather than struggling with huge number of *Forms*.

The 'Lookup Viewer' at the bottom left of Fig. 3 provides options allowing the user to select or enter data. This window is connected to an ontology and shows only relavent terminology from an ontology to help user input information while executing a task. Data inserted or selected by the user in the 'Lookup Viewer' is reflected in the 'Task Execution Viewer' window. When the user is finished entering all input for a task, the task is executed and this updates 'NOVA Browser' nodes. Fig. 4 shows how the 'NOVA Browser' looks like after executing the 'Measure BP' task.

Information available in the 'NOVA Browser' can also be used to annotate as a 'cause' for the execution of a task. This is one of the important feature of the proposed user interface design. Most software applications do not capture the cause of executing a task. While executing a task, the user uses his/her domain knowledge, expertise and finds some clue from the available information for a patient's case. This human expertise is crucial for healthcare applications. Recording pre-conditions while executing tasks would help analyzing historical information. Since healthcare professionals have to deal with many cases everyday, it might become very complicated for them to remember exactly under what circumstances they prescribed something or decided to do something for a patient. The way we proposed annotating cause of a task execution should greatly benefit healthcare professionals as they can easily point out and keep records of what motivated/influenced them to make a certain decision. Fig. 5 shows the execution of a task from 'CBPM' workflow named 'Change Thresh-

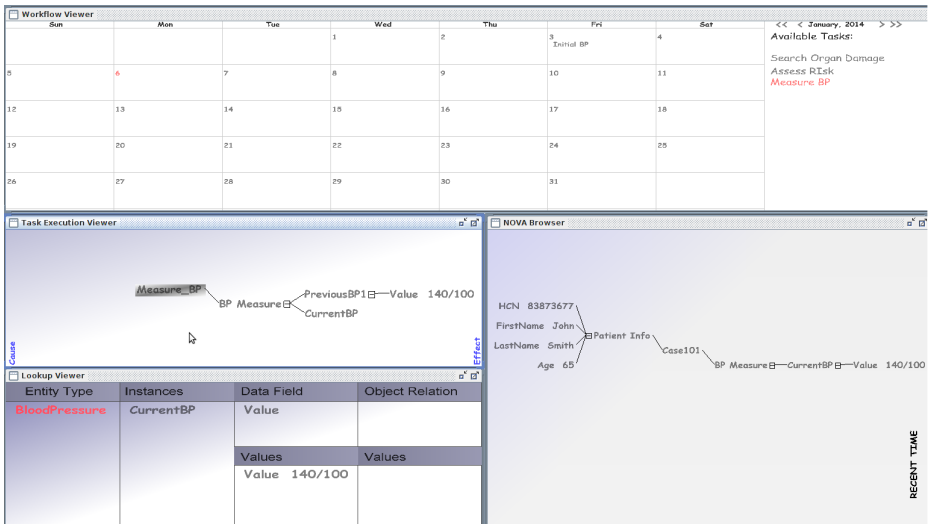

Fig. 3: Hypertension Management Workflow (Visit1)

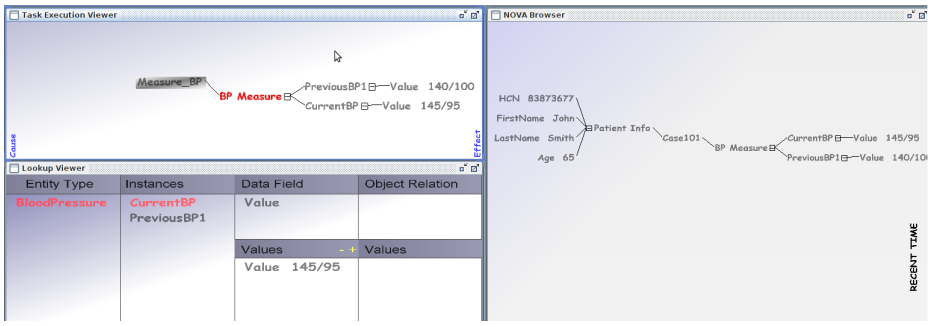

Fig. 4: Execution of Measure BP task

old'; the patient "John Smith" has a chronic kidney disease name 'Albuminuria' which is the potential reason for changing the threshold to 130/80 for subsequent workflow execution.

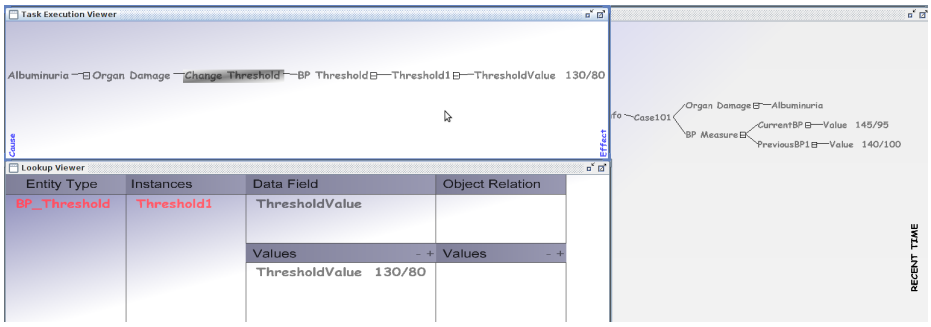

Fig. 5: Annotating cause for 'Change Threshold' Task

The reader may have noticed that the 'Task Execution Viewer' and the 'NOVA Browser' displays data similarly as in mindmap. A mind map is a visual thinking tool that graphically represents ideas and concepts and has been recognized as a brainstorming tool. We designed the windows similarly so that the user may brainstorm with the available information in front of them while executing tasks.

6 Syntax for $T_{\square}$

In [17] [16] we presented $T_{\square}$ for writing task specifications for workflows where action methods written in $T_{\square}$ code were transformed into web-services and view methods were transformed into Android user interface. In this paper we present an enhanced version of $T_{\square}$ in order to support the proposed user interface design. The enhanced version of $T_{\square}$ provides syntax to create and view nodes and branches in the 'Task Execution Viewer'. It also provides syntax to write action statements to manipulate the 'Lookup Viewer' window upon selection of a node from the 'Task Execution Viewer'. In the rest of this section we provide newly added syntax of $T_{\square}$. Note that a developer such as a programmer is supposed to write task specifications with $T_{\square}$. The end user such as a healthcare professional does not have to know the $T_{\square}$ syntax.

A variable in $T_{\square}$ may be bound with ontology classes, individuals, data properties or object properties. To bind a variable with an ontology class (see Line.2), individual (see Line.3), property (see Line.4) we may use following syntax:

```
1 var  vClass, vIndividual, vProp ;
2 {B$ vClass := BPMeasure $B}
3 {B$ vIndividual := BPMeasure("currentBP") $B}
```

```
{B$ vProp := vIndividual.Value $B}
```

Binding a variable with an individual (or property) means, binds it with an existing ontology individual (or property), rather than creating a new individual (or property). In order to create a new individual (or property) the following syntax with tag '{C$' may be used:

```
var vNewIndividual ;
{C$ vNewIndividual := BPMeasure("previousBP") $C}
{C$ Value(vNewIndividual, "120/80") $C}
```

A variable may be displayed in the 'Task Execution Viewer' by the following syntax. This syntax assigns a label "First Name" to the variable 'p' and the label is displayed in the viewer.

```
setInputLabel ( p , "FirstName" )
```

The 'Lookup Viewer' is provided to take input from the user. It has four panels: i) class panel, ii) individual panel, iii) data property panel, and iv) object property panel. These panels are used to visualize existing classes, individuals and their properties. One can select, create, or delete new individuals and/or properties of individuals from these panels. One can also programmatically assign data and/or object property facts to an individual using $T_{\square}$ syntax. To keep the self similar patterns, we introduce the access operator, dot('.'), for $T_{\square}$ variables to access properties. The idea is similar to object oriented programming; only here we will be using it for ontology concepts. One can instantiate a variable with an Ontology class and then access its individuals by '.' operator; and then further access the individual's properties by the '.' operator. An example is provided at the end of this section.

Upon selection of a variable from the 'Task Execution Viewer' by the user, its values are displayed in the 'Lookup Viewer'. Depending on the value of the variable, child nodes are displayed in the 'Task Execution Viewer' in tree structure. If the value assigned to the variable is of type Ontology Class, the selected individuals from the 'Lookup Viewer' of that class are displayed as child nodes in the 'Task Execution Viewer'; if the value assigned to the variable is of type Ontology Individual, the properties of that individual are displayed as child nodes in the viewer. Properties of an individual node are also displayed in the 'Task Execution Viewer' in self similar tree structure format; for data properties, the value of the data properties are displaed as child nodes; and for object properties, the individuals are displayed as child nodes.

In this enhanced version of $T_{\square}$ syntax, we have included some callback methods: $onSelectIndividual$, $onSelectProperty$, $onCreateIndividual$, $onCreateProperty$, $onDeleteIndividual$, $onDeleteProperty$. A variable may be registered with one or more callback methods. Whenever an appropriate event happens, the corresponding callback method registered to the variables are invoked. One can perform some operations inside the callback methods by writing $T_{\square}$ code; if a variable is manipulated from a callback method, its UI is also updated in the "Task Execution Viewer". The following code fragment shows an example:

```
1  var  weight, height, bmi;
2  {R$ Patient(?p), BMI(?p, ?bmi) → select(?p,?bmi)$R}
3  {B$ height := c.Height$B}
4  {B$ Weight := c.Weight$B}
5  setInputLabel ( height , "Height" );
6  setInputLabel ( weight , "Weight" );
7  height.onCreateProperty (heightValue){
8  bmi = (weight/(height * height)) * 703; }
9  weight.onCreateProperty (weightValue){
10 bmi = (weight/(height * height)) * 703; }
```

From Line.2 an ontology query is performed for a 'Patient' individual with its 'BMI' property. The variable *height*, *weight* are bound with the 'height' and 'Weight' property of individual p from Line.3. The callback function associated to the *height* and *weight* variable will be invoked if a property is added from the "Lookup Viewer". The newly added property fact, $heightValue$ and/or $weightValue$, are then used to calculate the *bmi* value. This update will be reflected to the node associated with the variable.

7 Integration with other applications

In this section we present the possibility of integrating DERF workflows with other applications by an example healthcare application. We developed a 'Personal health monitor' smart phone application that patients can use at home. The purpose of the application is to assist patients keeping their health record such as blood pressure record, body mass index, hours of exercise, dietary, etc. and monitor their performance with their lifestyle target that was set by the physician from 'Hypertension management workflow'. It is required that the 'Personal health monitor' application interacts with the 'Hypertension management workflow' presented in section 5. The integration has been accomplished by a task named 'SHBPM' (see Fig. 2) from the 'Hypertension management workflow'. We wrote an *action* method for the task 'SHBPM' and that exposed a web-service. Through the web-service it is possible for the 'Personal health monitor' smart phone application to invoke the task 'SHBMP' and submit the data entered by the patient. Fig. 6 shows two screenshots from the smart phone application that takes blood pressure input from the patient and displays a graph of their recent blood pressure changes. Fig. 7 shows another two screenshots that patient can use to monitor their exercise and eating behavior change. This smart phone appliction can also fetch appointment date from the workflow and reminds patient about the next visit date.

8 Related Work

In [6] the authors introduced the knowledge representation features of a multi-paradigm programming language called Go! that integrates logic, functional, object oriented and imperative programming styles. In that paper the authors

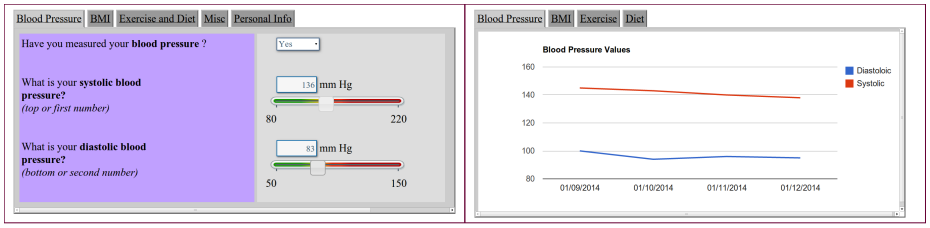

Fig. 6: Personal health monitor (Blood pressure monitoring)

Fig. 7: Personal health monitor (Exercise and diet pressure monitoring)

described the Go! language and its use for ontology oriented programming, comparing its expressiveness with OWL. This is related to our work since the authors also proposed a language for building executable ontologies. However the syntax proposed for $T_{\square}$ is simple and abstract and $T_{\square}$ provides syntax for control flow and UI design.

In [4] Baker et. al., surveyed a large number of existing workflow systems and listed their features considering different problem aspects. But none of them is following ontology based model driven approach as in $T_{\square}$.

Several heath applications have been developed recently based on smart phone systems such as [8] but they are lacking formal foundations such as workflow language, ontology, etc.

9 Conclusion

Incorporating stakeholders and monitors in the MDSE paradigm is highly innovative, however these features are essential if software systems are to automatically perform the kinds of tasks users are increasingly demanding. We believe that the metamodeling together with Model Driven Software Engineering principles in general (1) can be used as the main artefact in the development process of correct and adaptable workflow software systems targeting healthcare applications which incorporate monitors for sensitive data parameters and interfaces for a variety of stakeholders, and, moreover, (2) can be manipulated by computer tools to automatically produce a workflow implementation which

can be deployed in healthcare settings, which can easily be adapted to reflect the numerous customizations and changes required in a healthcare setting due to requirements and limitations of local settings, updates in protocols, etc. The DPF framework is used to implement our ideas. A demo of a small prototype may be found at: `http://www.screencast.com/t/QxG5knxVoO`.

Acknowledgement

We would like to thank Natural Sciences and Engineering Research Council of Canada, Atlantic Canada Opportunities Agency (ACOA) for supporting this research. We are greatful to Miao Huang for his contributions to the Hypertension Workflow Model and Jane Newlands for providing valuable feedbacks to us.

References

1. Fujaba Development Team: The Fujaba Tool Suite, http://www.fujaba.de/
2. Object Management Group: Semantics of a Foundational Subset for Executable UML Models (FUML) (February 2011), http://www.omg/spec/FUML/1.0/
3. Baarah, A.H.: An application framework for monitoring care processes. PhD thesis, University of Ottawa, 2013
4. Barker, A., van Hemert, J.: Scientific workflow: A survey and research directions. In: Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, vol. 4967, pp. 746–753. Springer Berlin / Heidelberg (2008)
5. Barr, M., Wells, C. (eds.): Category Theory for Computing Science, 2Nd Ed. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1995)
6. Clark, K.L., McCabe, F.G.: Ontology oriented programming in Go!. Appl. Intell. 24(3), 189–204 (2006)
7. Diskin, Z., Wolter, U.: A diagrammatic logic for object-oriented visual modeling. Electr. Notes Theor. Comput. Sci. 203(6), 19–41 (2008)
8. Gao, C., Kong, F., Tan, J.: Healthaware: Tackling obesity with health aware smart phone systems. In: Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on. pp. 1549–1554 (2009)
9. Kindler, E.: Model-based software engineering: The challenges of modelling behaviour. In: Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications. pp. 4:1–4:8. BM-FA '10, ACM, New York, NY, USA (2010), `http://doi.acm.org/10.1145/1811147.1811151`
10. Kristensen, L.M., Lamo, Y., MacCaull, W., Rabbi, F., Rutle, A.: Exploiting progress for memory efficient verification of diagrammatic workflows. 25th Nordic Workshop on Programming Theory (2013), Extended Abstract.
11. Lamo, Y., Wang, X., Mantz, F., Bech, Ø., Rutle, A.: Dpf editor: a multi-layer diagrammatic (meta) modelling environment. Proceedings of SPLST (2011)
12. Lamo, Y., Wang, X., Mantz, F., MacCaull, W., Rutle, A.: DPF Workbench: A Diagrammatic Multi-Layer Domain Specific (Meta-)Modelling Environment. In: Roger, L. (ed.) Computer and Information Science, Studies in Computer Intelligence, vol. 429, pp. 37–52. Springer (2012)
13. Leyla, N., MacCaull, W.: A personalized access control framework for workflow-based health care information. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops (2). Lecture Notes in Business Information Processing, vol. 100, pp. 273–284. Springer (2011)

14. MacCaull, W., Rabbi, F.: NOVA Workflow: A Workflow Management Tool Targeting Health Services Delivery. In: FHIES 2011: International Symposium on Foundations of Health Information Engineering and Systems (2011)
15. Rabbi, F., MacCaull, W.: Designing user-friendly uis for the execution of clinical practice guidelines. 27th International Symposium on Computer Based Medical Systems, 2014, Submitted
16. Rabbi, F., MacCaull, W.: Model driven workflow development with $T_{\Box}$;. In: Bajec, M., Eder, J. (eds.) CAiSE Workshops. Lecture Notes in Business Information Processing, vol. 112, pp. 265–279. Springer (2012)
17. Rabbi, F., MacCaull, W.: $T_{\Box}$: A Domain Specific Language for Rapid Workflow Development. In: MODELS 2012. pp. 36–52. Lecture Notes in Computer Science, Springer (2012)
18. Rutle, A.: Diagram Predicate Framework: A Formal Approach to MDE. Ph.D. thesis, Department of Informatics, University of Bergen, Norway (2010)
19. Rutle, A., MacCaull, W., Wang, H., Lamo, Y.: A Metamodelling Approach to Behavioural Modelling. In: Proceedings of BM-FA 2012: 4^{th} Workshop on Behavioural Modelling: Foundations and Applications. pp. 5:1–5:10. ACM (2012)
20. Rutle, A., Rabbi, F., MacCaull, W., Lamo, Y.: A user-friendly tool for model checking healthcare workflows. Procedia Computer Science 21(0), 317 – 326 (2013), the 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013) and the 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH)
21. Rutle, A., Rossini, A., Lamo, Y., Wolter, U.: A formal approach to the specification and transformation of constraints in mde. J. Log. Algebr. Program. 81(4), 422–457 (2012)
22. Rutle, A., Wang, H., MacCaull, W.: A formal diagrammatic approach to compensable workflow modelling. In: Weber, J., Perseil, I. (eds.) FHIES. Lecture Notes in Computer Science, vol. 7789, pp. 194–212. Springer (2012)
23. Rutle, A., Wang, H., MacCaull, W.: A Formal Diagrammatic Approach to Compensable Workflow Modelling. In: Liu, Z., Wassyng, A. (eds.) Foundations of Health Informatics Engineering and Systems, Lecture Notes in Computer Science, vol. 7789, pp. 194–212. Springer Berlin Heidelberg (2013)
24. Wang, H., Rutle, A., MacCaull, W.: A Formal Diagrammatic Approach to Timed Workflow Modelling. In: Proceedings of TASE 2012: 6^{th} International Conference on Theoretical Aspects of Software Engineering. vol. 0, pp. 167–174. IEEE Computer Society (2012)