A Framework for Defining and Verifying Clinical Guidelines: A Case Study on Cancer Screening

Federico Chesani[1], Pietro De Matteis[2], Paola Mello[1], Marco Montali[1], and Sergio Storari[3]

[1] DEIS, University of Bologna, Bologna, Italy
{fchesani, pmello, mmontali}@deis.unibo.it
[2] NOEMALIFE, Bologna, Italy
pgdematteis@dianoema.it
[3] ENDIF, University of Ferrara, Ferrara, Italy
strsrg@unife.it

Abstract. Medical guidelines are clinical behaviour recommendations used to help and support physicians in the definition of the most appropriate diagnosis and/or therapy within determinate clinical circumstances. Due to the intrinsic complexity of such guidelines, their application is not a trivial task; hence it is important to verify if health-care workers behave in a conform manner w.r.t. the intended model, and to evaluate how much their behaviour differs.

In this paper we present the GPROVE framework that we are developing within a regional project to describe medical guidelines in a visual way and to automatically perform the conformance verification.

1 Introduction

Medical guidelines are clinical behaviours recommendations used to help and support physicians in the definition of the most appropriate diagnosis and/or therapy within determinate clinical circumstances. These guidelines, usually represented as flow-charts, are intrinsically complex; moreover, their application in real cases becomes even more problematic. Hence, it is very important to verify if health-care workers applying them are conform with the intended models, and to evaluate how much their applications differ.

The sanitary organization of the Emilia Romagna region (Italy) uses these guidelines to describe and to manage cancer screening programs, in particular screening about cervical, breast and colorectal cancers. These guidelines are used to organize the screening, support test execution, analyze the conformance of the health-care workers behavior and evaluate how well the screening model "fits" the real screening application. Due to the management complexity of these screening programs, the Emilia Romagna sanitary organization is interested in software systems capable to support all these tasks.

In this paper we present the Guideline PRocess cOnformance VErification framework (GPROVE) that we are developing within the SPRING PRRITT regional project. Several software tools already address the guideline representation and execution issues but, to the best of our knowledge, no one deeply explores the aspect of conformance verification. Indeed, the SPRING PRRITT project aims to develop a system for supporting health-care workers involved in the cancer screening programs, covering both management and conformance verification issues. To this purpose, the GPROVE framework proposes a new visual representation language and a way to integrate it within a formal framework, based on computational logic, originally developed in the context of SOCS European project [1].

2 Formalization of the Colorectal Cancer Screening Guideline

As a case study for exploiting GPROVE potentialities we choose the colorectal cancer screening guideline proposed by the sanitary organization of the Emilia Romagna region [2]. Colorectal cancer is a disease in which malignant (cancer) cells form in the tissues of the colon or the rectum. Conformance verification for this screening is useful to compute statistics about the screening process used by the board of the sanitary organization, to monitor the progress of the screening program and to eventually propose changes.

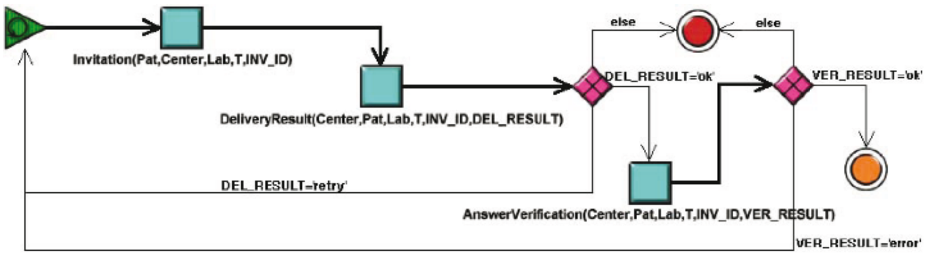
Invitation(Pat,Center,Lab,T,INV_ID)
DeliveryResult(Center,Pat,Lab,T,INV_ID,DEL_RESULT)
else
else
DEL_RESULT='ok'
VER_RESULT='ok'
DEL_RESULT='retry'
AnswerVerification(Center,Pat,Lab,T,INV_ID,VER_RESULT)
VER_RESULT='error'

Fig. 1. A fragment of the screening guideline in GOSPEL: the invitation phase

The guideline describes the screening program as composed by six phases: (1) Screening Planning, (2) Invitation Management, (3) First Level Test with Fecal Occult Blood Test, (4) Second Level Test with Colonoscopy, (5) Second Level Test with Opaque Clisma and (6) Therapeutic Treatment. Thanks to the GOSpeL language we have developed a prototype representation of this guideline (the methodology and the language used are described in Section 3). Figure 1 shows, for example, the GOSpeL flow chart diagram of phase (1).

3 Architecture and Modules of the GPROVE Framework

GPROVE offers several functionalities that allow an easy graphical guideline representation and the automatic translation of such language to a formal one. Thanks to this formal representation of the guideline knowledge, it is possible to perform several analysis on the behavior of the subjects involved in the process described by the guideline. The framework architecture, shown in Figure 2, is composed by four modules: the GOSpeL Editor, the Translation Module, the Verification Module and the Event Mapping Module.

The GOSpeL Language and Editor. GOSpeL is a graphical language, inspired by flow charts, for the specification and representation of all the activities that belong to a process and their flow inside it. The GOSpeL representation of a guideline consists of two different parts: a *flow chart*, which models the process evolution, and an *ontology*, which describes at a fixed level of abstraction the application domain and gives a semantics to the diagram.

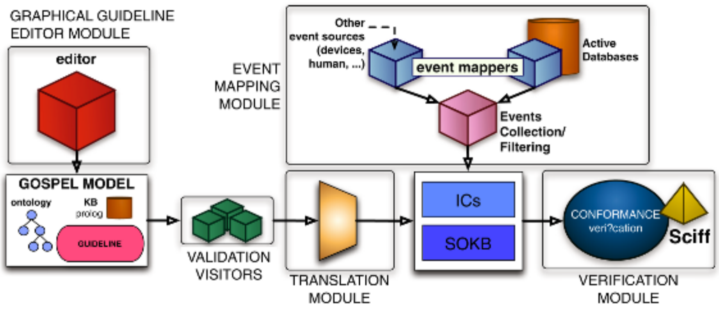

Fig. 2. Architecture and modules of the GPROVE framework

Like many other flow-chart languages, GOSpeL describes the guideline evolution using blocks and relations between blocks. These blocks are grouped into four families (as shown in Table 1): *activities*, blocks which represent guideline activities at the desired abstraction level; *gateways*, blocks that are used to manage the convergence (merge-block) and the divergence (split-block) of control flows; *start* blocks, start points of (sub)processes; *end* blocks, end points of (sub)processes. Due to the lack of space, we omit the detailed description of the GOSpeL language: the interested reader can refer to [3].

Table 1. GOSpeL elements

activities				**gateways**				**start blocks**		**end blocks**	
atomic activity	complex activity	iteration	while	exclusive choice	deferred choice	parallel fork	parallel join	start	cyclic start	return	end
🟦	▶	🟡	🟠	◆	◆	🟩	🟩	▶	▶	🟠	🔴

In the example of the GOSpeL diagram used to represent the invitation management phase, shown in Figure 1, the *DeliveryResult(...)* action block expresses that after the invitation, the screening center waits for the invitation delivery result (*DEL_RESULT*) and if the delivery fails (e.g. due to a wrong address) the invitation is repeated while if the patient is emigrated or died the screening process is aborted. In order to know which path should be followed, variable *DEL_RESULT* is used. For example, the invitation repetition case is represented by an arrow which connects the deferred choice to the cyclic start and is labeled by the logical condition *DEL_RESULT = 'retry'*.

In the editing phase, a semantics is given to the model by means of a domain ontology. This ontology is mainly composed by two taxonomies: one is developed by knowledge experts to model activities of interest, whereas a second taxonomy describes domain's entities, namely *actors*, *objects* and *terms*. Each ontological activity is associated with one or more ontological entities that are involved in its execution. Each atomic activity block in the GOSpeL language is semantically specified by assigning it an ontological activity and a set of formal participant of the requested type.

Participants are introduced within *macroblocks* giving them a unique name; *macroblocks* define also precise visibility rules of participants. When specifying an activity, visible participants can be associated to the selected ontological activity. During the execution, each participant will be grounded to a concrete instance. E.g., the activity block *DeliveryResult* in Figure 1 is associated with the correspondent *DeliveryResult* ontological concept. Its formal participants, like *Pat* and *Center*, are associated with ontological entities too (e.g., *Patient* and *Place* respectively). Creation and management of ontologies is performed through the Protege [4] tool.

The Formal Language and the Translation Module. The formal language used in the framework is a simplified version of the one proposed by Alberti et al. in the SOCS European project for the specification and verification of interaction protocols (see [5] for a complete description). The ongoing social participants behaviour is represented by a set of (ground) facts called *events* and denotated by the functor $\mathbf{H}$(that stands for "happened"). For example, $\mathbf{H}$*(invitation(strsrg, center1, lab1, '10-02-06', 300), 7)* represents the fact that *center1* has sent the invitation with ID *300* to *strsrg* at time *7*.

Even if the participants behavior is unpredictable, once interaction protocols are defined we are able to determine what are the possible expectations about future events. Events expected to happen are indicated by the functor $\mathbf{E}$. Expectations have the same format as events, but they will, typically, contain variables, to indicate that expected events are not completely specified. CLP constraints [6] can be imposed on variables to restrict their domain. For instance, ***E****(DeliveryResult(Center, Pat, Lab, T, INV_ID, DEL_RESULT),* $T_e) \wedge T_e \leq 15$ represents the expectation for *Center* to verify the delivery result of the invitation of *Pat* at a certain time T_e, whose value must be lower than *15* (i.e., a deadline has been imposed on T_e).

The way expectations are generated, given the happened events and the current expectations, is specified by means of *Social Integrity Constraints* (IC_S). An *IC* has the form of *body* $\rightarrow$ *head*, expressing that when *body* becomes true then it is supposed that the events specified in *head* will happen. In this way, we are able to define protocols as sets of forward rules, relating events to expectations.

During the translation phase, atomic activity blocks are treated as observable and relevant events, whereas merge and split blocks determine how events are composed within an integrity constraint. Start and end blocks map to special events used to identify the initial and the final step of a (sub)process.

The translation algorithm operates in two steps: in the first one, it partitions the diagram in special sub-sets called *minimal windows*; in the second one, it translates each minimal window to an IC. Interested readers can refer to [3] for a detailed description of the translation algorithm. Equation 1 has been generated automatically by the translation module, applied on the guideline of Figure 1. It states that, when a screening center *Center* verifies the invitation delivery status *DEL_RESULT*, then one among three next events are expected, depending on the value of *DEL_RESULT*.

$$
\begin{aligned}
&\mathbf{H}(DeliveryResult(Center, Pat, Lab, T, INV\_ID, DEL\_RESULT), T_{res})\\
\rightarrow&\mathbf{E}(AnswerVerification(Center, Pat, Lab, T, INV\_ID, VER\_RESULT), T_{ver})\\
&\wedge DEL\_RESULT = \text{`}ok\text{`} \wedge T_{ver} > T_{res}\\
\vee&\mathbf{E}(start(invitation), T_s) \wedge DEL\_RESULT = \text{`}retry\text{`} \wedge T_s > T_{res}\\
\vee&\mathbf{E}(abort(), T_a) \wedge DEL\_RESULT \neq \text{`}ok\text{`} \wedge DEL\_RESULT \neq \text{`}retry\text{`} \wedge T_a > T_{res}
\end{aligned}
\tag{1}
$$

The Verification Module. The conformance verification is performed by means of the operational counterpart of IC_S, i.e. the SCIFF abductive proof procedure [7]. Given the partial or the complete history of a specific execution (i.e., the set of already happened events recorded in a event log), SCIFF generates expectations about participants behaviour so as to comply with IC_S. A distinctive feature of SCIFF is its ability to check that the generated expectations are *fulfilled* by the actual participants behaviour (i.e., that events expected to happen have actually happened). If a participant does not behave as expected w.r.t. the model, the proof procedure detects such discrepancy and raises a violation.

Two kinds of guideline conformance violations are detected by the Verification Module: the first violation type happens when SCIFF does not find in the log an event that was expected (**E**(event) without the relative **H**(event)); the second violation type is detected when it finds an event that was not expected (**H**(event) without the relative **E**(event)): if an event is not explicitly expected, then it is automatically considered as prohibited.

At the moment, we are interested in verifying "a posteriori" if the event log of a guideline process case is conformant w.r.t. the model. However, since SCIFF is able to perform both off-line and run-time verification, we are also investigating the possibility to check during the guideline application if participants behave in a conform manner.

Event Mapping Module. The proposed event mapping module is capable to map ontological activities to a concrete DBMS and interact with it in order to extract the corresponding events and create an event log. Events are actively sent to the event mapping module by the DBMS (i.e. by triggers) or can be acquired via query statements.

4 Conclusions and Future Works

In this paper we have described a new framework, named GPROVE, for specifying medical guidelines and for verifying the conformance of the guideline execution w.r.t the specification. This framework is composed by four modules, that allow the user to graphically specify the guidelines (by means of the GOSpeL language and editor), to translate the specification into a formal language, to capture the events associated with the guideline execution and to verify the conformance of such executions w.r.t. the guideline. GPROVE is currently developed in the context of an Italian regional project on cancer screening management. The feasibility of the approach has been exploited on the real scenario of the colorectal cancer screening.

The literature proposes many systems related to the GPROVE framework. GLARE [8] and PROforma [9], to cite some, are powerful tools for supporting both representation and execution of medical guidelines. GPROVE, at the moment, is not integrated with an execution engine. However, it addresses the conformance verification issue that, as far as we know, neither GLARE nor PROforma tackle (indeed, GLARE deals only with time constraint conformance). Hence GPROVE can be considered a complementary approach w.r.t the ones proposed by GLARE and PROforma. Moreover, GPROVE can be used to model processes in different application domains, by means of different ontologies.

The conformance verification issue is addressed in [10], where the authors define the conformance verification of a process model on a event log as an analysis that involves two aspects: underfitting and overfitting. In our framework, the SCIFF proof procedure identifies both of them even if with a limited set of conformance statistics that will be improved soon.

In the future we plan to complete the development of the GPROVE modules and to extend the expressive power of GOSpeL, maintaining at the same time a valid mapping to the formalism. We will also deal with conformance verification at execution time.

Acknowledgments. This work has been partially supported by NOEMALIFE under the regional PRRITT project "Screening PRotocol INtelligent Government (SPRING)" and by the PRIN 2005 project "Specification and verification of agent interaction protocols".

References

1. Societies Of ComputeeS (SOCS), IST-2001-32530, http://lia.deis.unibo.it/research/socs/ (2006)
2. Emila Romagna, Colorectal cancer screening in the Emilia Romagna region of Italy, http://www.saluter.it/colon/documentazione.htm (2006)
3. Chesani, F., Ciampolini, A., Mello, P., Montali, M., Storari, S.: Testing guidelines conformance by translating a graphical language to computational logic. In: To appear in the proceedings of the ECAI 2006 workshop on AI techniques in healthcare: evidence-based guidelines and protocols. (2006)
4. Protege, http://stanford.protege.org (2006)
5. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interactions using social integrity constraints. Electronic Notes in Theoretical Computer Science **85**(2) (2003)
6. Jaffar, J., Maher, M.: Constraint logic programming: a survey. J. of Logic Programming **19-20** (1994) 503–582
7. The SCIFF Abductive Proof Procedure, http://lia.deis.unibo.it/research/sciff/ (2006)
8. Terenziani, P., Montani, S., Bottrighi, A., Torchio, M., Molino, G., Correndo, G.: The glare approach to clinical guidelines: main features. Stud. Health Tech. Inf. **101** (2004) 162–166
9. Fox, J., Johns, N., Rahmanzadeh, A.: Disseminating medical knowledge-the proforma approach. Artificial Intelligence in Medicine **14** (1998) 157–181
10. van der Aalst, W.M.P.: Business alignment: Using process mining as a tool for delta analysis and conformance testing. To appear in Requirements Engineering Journal (2006)