Abstract. In this work we present GOSpeL, a simple graphical language for modeling guidelines in a flow-chart fashion, and an algorithm capable of translating a GOSpeL model to a formal language based on computational logic and abductive logic programming in particular. The main advantage of this formalism lies in its operational proof-theoretic counterpart, which is able to verify the conformance of a given guideline execution w.r.t. the model, both at run-time or a posteriori. The feasibility of the approach has been tested on fragments of cancer screening protocols.

1 INTRODUCTION

In recent years, the use of guidelines and processes management techniques for describing and automating complex systems has become very popular in many real application areas, such as business process management and medicine. In order to take full advantage of these techniques it is important to provide for them a corresponding formal description.

Business processes and clinical guidelines are usually graphically represented as flow-charts, in order to clearly express the sequence of activities to be performed. The use of a graphical language for guidelines definition is therefore universally considered a necessary step to the aim of simplifying the job of guidelines developers and to explicitly communicate their logic to the staff who will execute it. On the other hand, once a guideline has been specified, it could be very useful to verify that actors executing it are compliant with the behavior rules expressed by the guideline itself.

In this work we present GOSpeL, a preliminary result of our research activity aimed to propose a solution for the specification and conformance verification of guidelines. GOSpeL (Guideline prOcess Specification Language) is a simple graphical language for modeling guidelines in a flow-chart fashion, with a particular regard to clinical ones. Given a GOSpeL model, we have defined an algorithm capable of translating it to the SCIFF language [1] based on computational logic and abductive logic programming in particular.

The SCIFF language was originally developed in the context of SOCS European project [9] for the specification and verification of agents interaction protocols within open and heterogeneous societies. Abduction [6] is a reasoning paradigm (well suitable in the medical field) which consists of formulating hypotheses (called abducibles) to account for observations; in most abductive frameworks, integrity constraints are imposed over possible hypotheses in order to prevent inconsistent explanations. The idea behind the SOCS frame-

1 DEIS – Università di Bologna – Viale Risorgimento, 2 – 40136 Bologna, Italy. {fchesani,aciampolini,pmello,mmontali}@deis.unibo.it
2 Dip. di Ingegneria – Università di Ferrara – Via Saragat, 1 – 44100 Ferrara, Italy. strseg@unife.it
Given the partial or the complete history of a specific execution (i.e., the set of already happened events), the operational counterpart of the abductive proof procedure ([8]) generates expectations about participants behaviour so as to comply with Social Integrity Constraints.

The most distinctive feature of this proof procedure is the ability to check that the generated expectations, considered as a particular class of abducibles, are fulfilled by the actual participants behaviour which cannot be assumed a priori in a society of autonomous interacting entities. If a participant does not behave as expected w.r.t. the model, the proof procedure detects and raises as soon as possible a violation.

2.1 Application to medical guidelines

Our approach is suitable for well focused medical guidelines whose execution should be strongly enforced (like for example screening protocols).

In this context, we are interested in detecting two different types of violation. The first one is raised when a participant does not act as expected by the guideline (i.e., an expectation is not fulfilled by a corresponding happened event); the second one is raised when a participant performs activities not expected by the guideline specification (e.g., an extended abductive proof procedure ([8]) expresses in minutes). Note that the high priority predicate is defined in the SOKB and its evaluation can eventually lead to an abductive reasoning.

When a violation is detected, two possible hypothesis could be given in order to explain such a violation: either the participant exhibited a wrong behavior w.r.t. the model, or it could be the case that the model has not been properly defined or updated and therefore it does not fit well with the real guideline’s application.

Assuming the latter hypotheses, violations are a useful hint in order to understand how the model specification lacks.

3 AN OVERVIEW OF GOSpeL

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.

Like a typical flow-chart language, 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.

GOSpeL allows guideline designers to follow a top-down approach representing guidelines at different abstraction levels. Therefore, simple and complex activities are distinguished. A simple activity represents a single atomic step within a guideline and it is used to model the case where the participants should perform some work. This kind of action cannot be further subdivided in activities of lower abstraction levels. Complex activities (usually called macroblocks) are managed at their abstraction level like atomic ones even if they encapsulate a new sub-process definition. Each (sub)process has one start block and one or more return blocks (an end block is eventually used to abort the entire process).

Thanks to macroblocks, it is possible to split recursively the guideline process into sub-processes, lowering the level of abstraction. In particular, GOSpeL proposes three macro-block types: complex action, iteration and while. The last two are used to explicitly represent cycles (with break-support).

<table>
<thead>
<tr>
<th>activities</th>
<th>gatesways</th>
<th>start blocks</th>
<th>end blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic activity</td>
<td>complex activity</td>
<td>iteration</td>
<td>while</td>
</tr>
<tr>
<td>exclusive choice</td>
<td>deferred choice</td>
<td>parallel fork</td>
<td>parallel join</td>
</tr>
<tr>
<td>start</td>
<td>cyclic start</td>
<td>return</td>
<td>end</td>
</tr>
</tbody>
</table>

Gateways are used for modeling complex guidelines as long as they express workflow’s decision points and activities concurrency. Parallel fork and join blocks are used to represent respectively split and synchronization of multiple threads of control.

Finally, two different decision types are supported. The first one, called exclusive choice, associates to each outgoing relation a logical guard: when the choice is reached, all guards are evaluated and one among the paths which have a true condition is chosen. Guards evaluation is mapped to a CLP constraint evaluation or call of a predicate defined in the SOKB, thus leading to a reasoning mechanism. A deferred choice, instead, proposes different exclusive alternatives, but without specifying any logical condition for choosing among them. Thus, the choice is delayed until one of the possible paths is actually performed by participants.

GOSpeL blocks can be connected using order relations, in order to express how the flow will walk through them, or temporal relations, in order to model time constraints among activities (e.g. deadlines).

Now we have briefly introduced the structure of a GOSpeL diagram but we have not yet described how atomic activities and logical conditions are expressed. In order to specify that an atomic activity represents actually the temperature measurement on a patient performed by a physician, we make use of a domain ontology. This ontology is mainly composed by two taxonomies: the first is developed by knowledge experts to model activities of interest at the desired abstraction level, whereas the second describes domain’s entities, namely actors, objects and terms. Each ontological activity is associated with one or more ontological entities that represent the kind of participants involved in its execution. In the example above, the activity taxonomy contains the concept of measurement associated to the physician, patient and temperature ontological entities.
Each atomic activity block is semantically specified choosing an ontological activity and a block of formal participant of the types requested. A formal participant is defined by a name and a concept inside the entities taxonomy. During the execution, each formal participant will be grounded to a concrete one.

4 TRANSLATION ALGORITHM

In this section we provide a description of the algorithm capable of visiting a GOSpeL diagram and translate it to a set of Social Integrity Constraints.

Let us consider an atomic activity block in a GOSpeL diagram, say, block $A$. For simplicity, we do not care about its completion time and, therefore, $A$ represents directly an observable and relevant event to be generated. The properties of this event, namely its name and its logical variables, are respectively determined by the name of the ontological activity and the set of formal participants associated to $A$ (thus, the examination of a patient $Pat$ performed by a physician $Phy$ becomes the event examine($Phy$, $Pat$)). We will use notation $e_A$ for representing the event to which a generic block $A$ is mapped.

Now we assume that $A$ is correctly executed (i.e., $e_A$ is generated by the "hospital society", becoming a happened event). Following the $IC$ viewpoint, the part of the diagram next to $A$ may be seen as a suggestion of the behaviour which has to be exhibited by participants. Therefore, we are generating an $IC$ whose body contains the happening of $e_A$ and whose head is determined by the consequent diagram part. Leaving $A$ and going forward, for each branch we will encounter, sooner or later, a new activity block, which maps to an expectation about the future participants behaviour. Afterward, we consider these blocks as new start points and proceed recursively.

4.1 A Few Definitions

Now we give some definitions just for the sake of clarity. Our aim is to identify special diagram sub-sets, which play a key role during the translation procedure.

In the previous paragraph we have shown that an atomic activity block is translated to an event: we say therefore that it is an event-block.

Definition 1 (event-block) A GOSpeL element is an event-block if it is mapped directly to an event during the translation procedure.

Beyond atomic activities, start, return and end blocks are event-blocks too: even if they don’t really represent a concrete working step during the process application, they are used as terminal points that identify the start and the conclusion of a (sub)process.

In an operational sense, we have seen that the translation procedure operates starting from an event-block and visiting the diagram until, for each branch, another event-block is found; after having translated this part, the algorithm proceeds recursively starting from the new encountered event-blocks. Therefore, the algorithm visits a GOSpeL diagram partitioning it into special sub-sets (called minimal windows) and translating each sub-set to one Social Integrity Constraint.

In order to define a minimal window, we introduce some other concepts: precursors and successors set, path, window and window’s source and fringe.

Definition 2 (precursors and successors set) Given a block $b$:

- $Suc_b$ is the set of blocks to which $b$ is directly connected through its outgoing relations (successors set);

- $Pre_b$ is the set of blocks to which $b$ is directly connected through its ingoing relations (precursors set).

Definition 3 (path) A path $P(s, d)$ is a sequence of blocks through which block $s$ and block $d$ are connected, following the order relations. Identifying the sequence as $b_0 = s, b_1, \ldots, b_{n-1}, b_n = d$, we have:

$$b_j \in Suc_{b_{j-1}} \land b_j \in Pre_{b_{j+1}}, \forall j = 1, \ldots, n - 1$$

Definition 4 (window) A window $W$ is a set of connected blocks:

$$\forall b_1 \in W \exists b_2 \in W s.t. \exists P(b_1, b_2) \in W \lor \exists P(b_2, b_1) \in W$$

Note that $P(s, d) \in W$ iff all the blocks of the sequence belong to $W$.

Definition 5 (window source and fringe) The source and the fringe of a window $W$ are respectively:

- $S_W = \{b \in W | \exists b' \in W s.t. \exists P(b', b)\}$
- $F_W = \{b \in W | \exists b' \in W s.t. \exists P(b, b')\}$

We can give now a formal definition of a minimal window.

Definition 6 (minimal window) A window $W$ is minimal iff $\forall b \in W$ the following properties hold:

1. if $b \in S_W$ then $b$ is an event-block;
2. else if $b \in F_W$ then $b$ is an event-block;
3. else $b$ is not an event-block (i.e., is a split or merge);
4. if $b$ is a split-block then $Suc_b \in W$;
5. if $b$ is a merge-block then $Pre_b \in W$.

The last two properties ensure that when a split-block (a merge-block respectively) belongs to the minimal window, all the branches which diverge from (converge to, resp.) it are included.

Note that, for a well-formed flow-chart, it is impossible to have a window that contains a split-block followed by a merge one (each path that connects two blocks of this type must include at least an activity-block between them).

4.2 Mapping of a Minimal Window to an IC

Figure 1 shows a minimal window together with its translation. It’s easy to see a tight similarity between the minimal window and the abstract parse tree of the corresponding $IC$.

The translation procedure of a minimal window $W$, named in the following $GENERATE_{IC}$, operates as follows $^{4, 5}$:

1. $\forall b \in S_W$ generates $H(e_b, T_b)$ (if $b$ is a macroblock, its end event is chosen);
2. creates a body composing the happened events in a way that depends on the merge-blocks in $W$;
3. $\forall b \in F_W$ generates $E(e_b, T_b)$ (if $b$ is a macroblock, its start point is chosen);
4. creates a head composing the expectations in a way that depends on the split-blocks in $W$.

$^4$ remember that $S_W$ and $F_W$ contain only event-blocks (Property 1 and 2 of Definition 6).

$^5$ For the sake of clarity, we make the assumption that each ontological activity is associated at most to one activity block. The general algorithm does not require to state this assumption.
windows and mapping each window to a an IC:
1: ics = ∅, visited = ∅, fringe ← Start
2:  while fringe ≠ ∅ do
3:     cur ← REMOVE.ONE(fringe)
4:     W ← CONSTRUCT_MINIMAL_WINDOW(cur)
5:     ics ← ics ∪ GENERATE.IC(W)
6:     visited ← visited ∪ SvW
7:     fringe ← [fringe ∪ Fw] − visited
8:  end while

The fringe set, which initially contains only Start, represents dynamically the frontier of the part already covered. At each iteration step, one element is extracted from fringe, say, cur. At line 4, the minimal window W s.t. cur ∈ SvW is found. Operationally, W is constructed starting from cur and visiting the diagram partially forward and partially backward (when a merge block is encountered, Property 5 of Definition 6 says that all its previous branches should be included). The mapping of cur is then handled by the GENERATE.IC procedure, which has been described in the previous paragraph. Finally, the visited and fringe sets are updated to avoid repetition: remember indeed that in GOSePL different alternatives may converge to a single path. Figure 2 shows how a fragment of simple diagram is partitioned into minimal windows.

5 A SIMPLE EXAMPLE

Let us consider the fragment of a simplified clinical guideline on cervical cancer screening [3] shown in Figure 2.

In this example, a lab Lab analyses a pap-Test IDsample of patient Pat and send a report PTres, containing a set of signs on the sample, to the screening physician Phy. Phy evaluates PTres and classifies IDsample as positive (cancer evidence found) or negative (normal). If positive, the protocol prescribes that Pat should be invited, in parallel, for the cancer treatment and for a psychological consult.\(^6\) Note that the treatment invitation should be sent to the patient within a deadline of six days. In case of a negative evaluation a letter should be sent to Pat reporting that the pap-test is normal. The positive and negative flows converge in a single one which proposes as activity the scheduling of the next pap-test.

5.1 Translation of a GOSeL Fragment

Let us suppose that, initially, fringe = \{A\}. At the first iteration step the algorithm extracts A and, launching a visit from it, individuates W1, which has \(S_{W_1} = \{A\}\) and \(F_{W_1} = \{B, C, D\}\). Therefore, the following IC is produced:

\[
\begin{align*}
H(\text{analysePapTest} & (\text{Lab}, \text{Pat}, \text{IDsample}, \text{Phy}, \text{PTres}), T_{ana}) \\
& \rightarrow \text{positive}(PTres) \\
& \land E(\text{treatmentInvitation}(\text{Phy}, \text{Pat}, \text{IDsample}), T_{tinv}) \\
& \land T_{tinv} > T_{ana} \land T_{ana} + 6 \\
& \land E(\text{psiInvitation}(\text{Pat}, \text{PTres}), T_{psi}) \land T_{psi} > T_{ana} \\
& \land \text{not}(\text{positive}(PTres)) \\
& \land E(\text{sendNegLetter} (\text{Phy}, \text{Pat}, \text{IDsample}, \text{PTres}), T_{sen}) \\
& \land T_{sen} > T_{ana}
\end{align*}
\]

(2)

Note that the temporal constraint between A and C is inserted as a CLP constraint over \(T_{tinv}\) and \(T_{ana}\). Other time constraints are automatically generated due to the partial order imposed by order

\(^6\) For simplicity, we omit the part of the diagram in which the patient confirms or refuses these proposals.
relations. The exclusive choice condition is mapped to the evaluation of the predicate positive1, contained in the SOKB. A pap-test is positive if almost one cervical cancer type between the classes HSIL, LSIL and ASCUS [3] can be detected. Since each cancer type is characterized by a specific set of laboratory results, the predicate positive1 verifies if almost one of three possible cancer types has more than an half of its supporting signs in PTres. This is a trivial description used only in order to exploit the reasoning capabilities of the SOKB.

Now the algorithm proceeds updating the fringe set, which becomes fringe = {B, C, D}. Supposing B is extracted, the algorithm finds window W2, whose translation is straightforward.

After having translated W2 the fringe contains C, D and E. If either block C or D are extracted, due to the presence of a parallel join the algorithm finds a window which has SW2 = {C, D} and FW2 = {E}, and generates the following IC:

$$
H((\text{treatmentInvitation}(\text{Phy, Pat, IDsample}), \text{Ttre}), \text{psy}) \\
\lor H((\text{psyInvitation}(\text{psy, Pat}), \text{Tpsy}), \text{psy}) \\
\lor H((\text{screeningSchedule}(\text{Phy, Pat, InvDate}), \text{Tscr}), \text{psy}) \\
\lor T_{\text{psy}} = T_{\text{scr}} > T_{\text{sce}} > T_{\text{psy}}
$$

(3)

### 5.2 Conformance Verification Test

Since now the graphical description is translated to the SCiFF language, the SCiFF proof procedure can be used in order to verify the conformance of a guideline execution w.r.t. the model, both at runtime or a posteriori. Let us consider for example a simple execution of the above guideline’s fragment, namely a set of happened events⁷:

1. $H((\text{analysePapTest}(\text{Lab, Pat, IDsample, Phy, PTres}), T_{\text{ana}}), 5)$
2. $H((\text{psyInvitation}(\text{psy, Pat}), 7)$
3. $H((\text{treatmentInvitation}(\text{phy, pat, 123}, 20)$
4. $H((\text{screeningSchedule}(\text{phy, pat, 15apr2007}), 30)$

When the pap-test analysis is passed to the proof procedure, the first IC triggers and, supposing that the predicate positive1([res1, ..., resn]) succeeds, we have two pending expectations: $E((\text{treatmentInvitation}(\text{phy, pat, 123}), T_{\text{tre}}) \land T_{\text{tre}} \in [6, 11])$ and $E((\text{psyInvitation}(\text{psy, pat}), T_{\text{psy}}) \land T_{\text{psy}} > 5$.

Now we have that the second happened event fulfills the second expectation, grounding $T_{\text{psy}}$ to 20, whereas the treatment invitation event matches with the first one. Unfortunately, the match implies that $T_{\text{tre}}$ unifies with 30, which does not satisfy the deadline and causes therefore a violation to be raised.

### 6 RELATED WORKS

In literature, several graphical notations have been proposed to represent clinical guidelines but, for the sake of space, we limit ourselves with only two examples of medical guidelines support systems: GLARE [10] and PROforma [4].

GLARE [10] is a system for acquiring, representing and executing clinical guidelines. The GLARE representation language consists of different types of actions: plans (i.e., composite actions) and atomic actions (i.e., queries and decisions). All actions are linked by control relations, defining their order of execution. The system provides consistency checks, advanced temporal reasoning techniques and what-if functionalities.

PROforma [4] is a formal language capable to represent a clinical guideline in terms of a network of tasks and data items. There are four types of tasks: plans (i.e., composite actions), decisions, actions (typically clinical procedures) and inquiries (requests for further information or data).

Comparing GLARE and PROforma with our approach we notice that GLARE and PROforma are powerful tools for supporting both representation and execution of medical guidelines while, at the moment, GOSpeL is not integrated with an execution engine. However, we address the conformance verification issue that, as much as we are concerned, GLARE and PROforma do not tackle (GLARE deals only with time constraints conformance). For this reason our approach can be considered complementary w.r.t the ones proposed by GLARE and PROforma.

Moreover, while GLARE and PROforma are mainly focused on the medical domain, GOSpeL can be used to model other domains. This can be done representing actors and entities in a novel ontology. Finally, since GOSpeL has an explicit model of both actors and activities, it is suitable for representing both administrative and diagnostic aspects of guidelines, where usually many different participants are involved. This is the case of screening protocols.

The conformance verification issue has been explored also in the field of business process management. For example, in [7] the au-

---

⁷ In a typical scenario, these events may be passed to the proof as soon as they happen or extracted from the log of a hospital information system.
thors show how to perform conformance verification a posteriori. Their framework, however, deals only with simplified models since it is impossible to express conditions on the content of activities, deadlines and predicates, which can be instead expressed by using GOSpeL.

7 CONCLUSIONS

In this work we have described GOSpeL, a simple flow-chart language for modeling guidelines, showing how it can be automatically translated to the SCIFF logic-based language. The SCIFF representation, together with the events which occur during the guideline execution, are used by an abductive proof procedure to perform the conformance verification both at run-time or a posteriori.

We are testing the feasibility and the potentials of our approach in the context of a national project, using GOSpeL for describing and verifying cancer screening protocols. In this way, we are going to extract event logs from the screening information system and check if they are compliant with national and regional recommendations.

Another ongoing work is about the proof of “high level” properties on the formalized guideline specification. To the aim of doing that, we are using an extension of the SCIFF proof procedure (named g-SCIFF), developed within the SOCS European project and already tested on security protocols [2]. For instance, given the IC representation of the above guideline fragment, we can ask to g-SCIFF if a history exists s.t. a treatment invitation is sent to the patient. If this is the case, g-SCIFF will produce a successful proof, generating the corresponding history.

In the future, an interesting issue would be to apply the translation algorithm proposed in this paper to other guidelines support systems (such as GLARE and PROforma), in order to integrate their many features with the conformance verification capability.

ACKNOWLEDGEMENTS

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


