An Ontology for Human-Machine Computation Workflow Specification

Nuno Luz[1], Carlos Pereira[2], Nuno Silva[1], Paulo Novais[3], António Teixeira[2], Miguel Oliveira e Silva[2]

[1] GECAD (Knowledge Engineering and Decision Support Group), Polytechnic of Porto
{nmalu, nps}@isep.ipp.pt
[2] DETI/IEETA (Department of Electronics Telecommunications and Informatics/Institute of Electronics and Telematics Engineering of Aveiro), University of Aveiro
{cepereira, ajst, mos}@ua.pt
[3] CCTC (Computer Science and Technology Center), University of Minho
pjon@di.uminho.pt

Abstract. Lately, a focus has been given to the re-usability of workflow definitions and to flexible and re-usable workflow components, culminating with approaches that harness the benefits of the enriched semantics provided by ontologies. Following this trend and the needs of multiple application domains, such as micro-task crowdsourcing and ambient assisted living, of incorporating cooperation between the efforts of human and machine entities, this paper proposes an ontology and process for the definition, instantiation and execution of semantically enriched workflows.

Keywords: Workflow, Task, Ontology, Domain Knowledge

1 Introduction

Extensive work exists regarding workflow specification languages and formalisms [1–5], which focus on the workflow definition and instantiation. Workflows are typically used to represent business processes with languages such as YAWL [2], and commercial languages such as XPDL (XML Process Definition Language) and BPEL (Business Process Execution Language), which lack semantics and formal definitions [6]. The standardization efforts that led to the emergence of these languages date back to 1993 with the emergence of the WfMC (Workflow Management Coalition), a coalition of several companies with the purpose of standardizing workflow model specification (or definitions) [5].

Lately, a focus has been given to the re-usability of workflow definitions and to giving some degree of adaptation and flexibility to workflow components, culminating with approaches that harness the benefits of the enriched semantics provided by ontologies [3, 5, 7]. Besides allowing re-usability, ontologies are conceptual models, closer to the human conceptual level, which provide structure and semantics understandable to machines [8, 9]. In this sense, ontologies are ideal for agile model development focuses on domain knowledge [10].

Particularly, in [5], OWL (Web Ontology Language) ontologies are used to capture the semantics of the workflow domain in order to provide inference and guide the workflow execution engine into the following steps of the workflow.

OWL-S (Semantic Markup for Web Services) [7] is a format that introduces the semantics of OWL to web service specification and composition (service workflows). Although it is not a format for workflow definition, services are represented as processes or workflows of atomic operations with input and output parameters. Thus, the specification of services shares many similarities with workflow definition specification languages.

Current workflow definition approaches lack or do not consider the semantics of the atomic operations performed throughout the workflow. They are usually limited to the specification of input and output parameters along with some identification of the type of operation. OWL-S, however, includes domain ontologies (with the inherent expressivity of Description Logic languages) in web service process definitions. Besides providing benefits in domain workflow extensibility, such semantics would aid in the interoperability of workflow engines and execution agents.

In this paper, a workflow specification ontology tailored for workflows of human-machine computations is proposed. The approach absorbs ideas from other workflow definition languages and retains the benefits of OWL-S.

Workflow definitions according to this approach inherit the benefits of Description Logic ontologies such as re-usability and extensibility. A focus on domain knowledge is given through domain concepts, which describe a specific task or work domain. These concepts can be re-used by multiple workflow definitions.

The ultimate purpose of this architecture is to allow not only the specification of workflow definitions, but also to include all knowledge and semantics of atomic tasks (not only the input and output, but also the full description of the operation itself) in the definition through Description Logics [11]. Furthermore, the reasoning and conceptualization capabilities given by ontologies are exploited in order to establish a human-machine environment for solving workflows of tasks.

Typical applications of this work include micro-task crowdsourcing [12], which benefits from the structure and semantics given by ontologies, and Ambient Assisted Living (AAL) approaches [13], which benefit from the inherent scalability and re-usability of the proposed solution.

This paper is organized as follows. Section 2 presents the proposed CompFlow process, which is followed by its formal definition on section 3. Section 4 provides a use case of CompFlow in the AAL domain. Section 5 concludes this work with some remarks on future work.

2 The CompFlow Process

The CompFlow is a process for (1) workflow definition, (2) instantiation and (3) execution (see fig. 1). The workflow definition phase (1) results in a workflow-definition ontology that extends the CompFlow upper ontology. The workflow-definition ontology represents a workflow-definition for a specific domain, which can be instantiated multiple times in the workflow instantiation phase (2). These instantiations are finally executed during the workflow execution phase (3). In this paper, a focus is given to the workflow definition phase. Phases 2 and 3 are considered in the context of an execution engine and are, thus, left outside the scope of this paper.

In some situations, an abstract workflow-definition may result from the workflow definition phase (1). These definitions capture only the domain knowledge and cannot be instantiated. They can, however, be extended by concrete workflow-definitions.

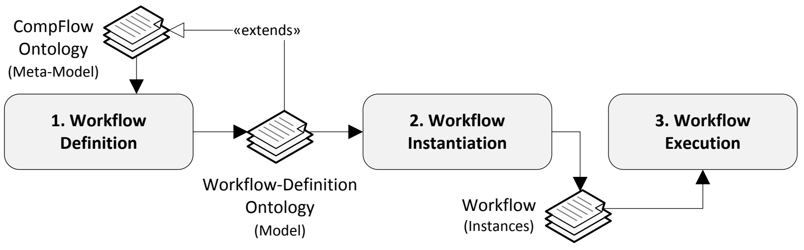
CompFlow Ontology (Meta-Model)
«extends»
1. Workflow Definition
Workflow-Definition Ontology (Model)
2. Workflow Instantiation
Workflow (Instances)
3. Workflow Execution

Fig. 1. The CompFlow workflow definition, instantiation and execution process.

The workflow meta-model (see fig. 2) is fixed and defines the constructs required for workflow-definitions at the model level. In practice, the meta-model is represented through the CompFlow upper ontology.

A workflow is considered to be a set of tasks ordered according to a set of procedural rules in order to deliver a specific result in a specific domain of application or knowledge. The model or specification of a workflow is a workflow-definition, which contains elements called activity-definitions. Activity-definitions have an associated priority value that can be used to establish an execution order. There are four types of activity-definitions: task-definitions, event-definitions, gateway-definitions and workflow-definitions. Analogously, instantiations of a workflow-definition, for different units of work, are called workflows.

Task-definitions model tasks (the full atomic operations and their semantics) in a specific domain. Tasks (instances of a task-definition), inherently belonging to a workflow, perform atomic operations over data, which may have an associated (physical) effect on the state of the world.

Each task is performed by workers which can represent machines and/or humans. Workers have access to a task through a specific interface. Interface-definitions establish the different types of interfaces through which a task can be delivered to a worker. For instance, tasks can be delivered to a worker through a visual interface, sound interface, or simply through a web interface (the common case for crowdsourcing applications). The inclusion of an interface as an element within the ontology allows the customization of standard interfaces according to application scenarios. This customization enables the creation of mixed or multimodal interfaces, capable of merging and coordinating multiple interfaces, commonly used on user-centric environments.

Event-definitions specify events that may either (i) trigger the continuation of an existing workflow or (ii) trigger a new instantiation and execution of a workflow-definition. Events are received and handled through event interfaces that follow a publish-subscribe pattern [14].

Gateway-definitions establish flow control blocks in the workflow-definition. While input gateway-definitions establish different behaviors on the input of the gateway, output gateway-definitions establish different behaviors on the output of the gateway. For instance, regarding input gateways, if a merge input gateway is found during execution, the engine must simply wait for the first input in order to continue. Instead, if a sync input gateway is present, the engine must wait for all inputs to arrive. Regarding output gateways, if a decision output gateway is found, the gateway condition must be evaluated in order to decide which paths must be followed next. If a parallel output gateway is found, all paths are followed concurrently.

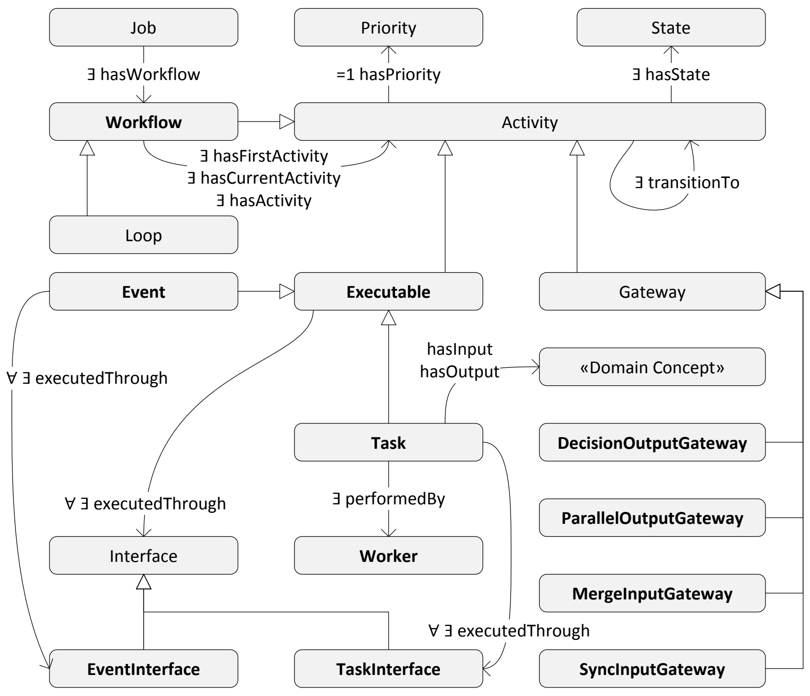

Fig. 2. The CompFlow upper ontology (meta-model).

3 CompFlow Formal Definition

A CompFlow structure is a singleton $CompFlow = (Z)$, where Z is the set of entities pertaining to $CompFlow$. A $CompFlow$ represents a self-contained unit of structured information. Elements in a $CompFlow$ are called workflow definition entities.

3.1 Workflow-Definition

A CompFlow extension, which represents a workflow-definition, is a 11-tuple $WFD(CompFlow) := (A, W, F, P, AR, ER, TR, C, CIN, COUT, \Omega)$ where:

- $A \sqsubseteq Z$ is a set of activity-definitions, where:
  - $E \sqsubseteq A$ is the set of executable-definitions;
    - $T \sqsubseteq E$ is the set of task-definitions;
    - $EV \sqsubseteq E$ is the set of event-definitions;
  - $G \sqsubseteq A$ is the set of gateway-definitions;
    - $DOG \sqsubseteq G$ is the set of gateway-definitions corresponding to DecisionOutputGateways;
    - $POG \sqsubseteq G$ is the set of gateway-definitions corresponding to ParallelOutput-Gateways;
    - $MIG \sqsubseteq G$ is the set of gateway-definitions corresponding to MergeInput-Gateways;
    - $SIG \sqsubseteq G$ is the set of gateway-definitions corresponding to SyncInputGateways;
  - $SA \sqsubseteq A$ is the set of activity-definitions that start the workflow-definition;
    - $ST \sqsubseteq T \sqcap SA$ is the set of task-definitions that start the workflow-definition;
    - $SG \sqsubseteq G \sqcap SA$ is the set of gateway-definitions that start the workflow-definition;
    - $SEV \sqsubseteq EV \sqcap SA$ is the set of event-definitions that start the workflow-definition;
- $W \sqsubseteq Z \wedge W \equiv MW \sqcup HW$ is a set of worker-definitions (or roles), where:
  - MW is the set of machine worker-definitions;
  - HW is the set of human worker-definitions;
- $F \sqsubseteq Z \wedge F \equiv EF \sqcup TF$ is a set of interface-definitions, where:
  - EF is the set of event interface-definitions;
  - TF is the set of task interface-definitions;
- $P \sqsubseteq (Z, \leq)$ is a totally ordered set of priority values;
- $AR : A \rightarrow 2^A \times P$ defines, for an activity-definition, (i) the following activity-definition (transition restriction) and (ii) a priority value, where:
  - $APR : A \rightarrow P$ is an injective function that defines the priority value for each activity-definition;
  - $AAR \sqsubseteq A \times A$ is relationship that defines a transition restriction between two activity-definitions;
- $ER : E \rightarrow F$ is a function that defines the interface-definition for each executable-definition;
- $TR : T \rightarrow W$ is a function that defines the worker-definition (or role) for each task-definition;
- $C \sqsubseteq Z$ is the set of domain concepts that define the input and output of task-definitions;
- $CIN \sqsubseteq T \times C$ is a relation that defines the input concept of the task-definition;
- $COUT \sqsubseteq T \times C$ is a relation that defines the output concept of the task-definition;

- $\Omega^{\alpha} \sqsubseteq \alpha \times \alpha$ is a subsumption relationship between elements of the same set $\alpha \sqsubseteq Z$, which maps to the sub-class-of relationship in Description Logics.

A workflow-definition consists in building a domain-specific workflow-definition ontology (model) that extends the CompFlow upper ontology (meta-model).

An activity-definition a is called abstract iff it doesn't belong to any workflow-definition, i.e., $a \notin SA$ and $\forall x \in A \Rightarrow a \notin AAR^{+}(a,x) \wedge a \notin AAR^{+}(x,a)$. Otherwise, it is called concrete.

For concrete workflow-definitions the following rules must be satisfied:

- A priority value must be specified for each activity-definition: $\forall a : a \in A \Rightarrow \exists p : p \in P \wedge APR(a) = p$;
- An input and output domain concept must be specified for each task-definition: $\forall t : t \in T \Rightarrow \exists c1, c2 : c1 \in C \wedge c2 \in C \wedge (t, c1) \in CIN \wedge (t, c2) \in COUT$;
- A worker-definition must be specified for each task-definition: $\forall t : t \in T \Rightarrow \exists w : w \in W \wedge TR(t) = w$;
- An interface-definition must be specified for every executable-definition: $\forall e : e \in E \Rightarrow \exists f : f \in F \wedge ER(e) = f$.

It is assumed that activity-definitions, interface-definitions and worker-definitions are concept descriptors according to Description Logic languages. This allows the specification of domain-specific abstract definitions, from which new subsumed definitions (through the Ω^{α} relationship) can be created in order to build new workflow-definition. The CI and CO relations can also be subsumed in order to represent specific domain relations between domain concepts. For instance, text constitutes the input and output of a translation task. However the *input* is referred to, specifically, as the *originalText*, and the *output* as the *translatedText*.

In this paper, Description Logic knowledge bases and ontologies are considered. A Description Logic knowledge base contains a TBox (terminological box) and an ABox (assertion box) [15], where the TBox contains all the concepts and relationships that define a specific domain (workflow-definition), and the ABox contains the instances or individuals defined according to the elements in the TBox (workflow instantiation).

3.2 Workflow-Definition Instantiation

An instantiation of a workflow consists in creating and preparing an instance of the workflow definition for future execution.

A CompFlow workflow-definition instantiation is a 10-tuple $WFDI(CompFlow) := (I, instA, instW, instF, S, \Sigma, execThrough, perfBy, instC, \Phi)$ where:

- $I \sqsubseteq Z$ is a set of instances;
- $instA : A \rightarrow 2^{I}$ is a function that relates an activity-definition (task-definition, event-definition or gateway-definition) with a set of instances. Consequently, the set of all activity instantiations AI is defined as $AI = \bigcup_{\forall x \in A} instA(x)$. Instantiations of the sub-sets of A are defined as:

- $instE : E \rightarrow 2^I$ is a function that relates an executable-definition with a set of instances. Consequently, the set of all executable instantiations EI is defined as $EI = \bigcup_{\forall x \in E} instE(x)$;
  - $instT : T \rightarrow 2^I$ is a function that relates a task-definition with a set of instances (tasks). Consequently, the set of all tasks TI is defined as $TI = \bigcup_{\forall x \in T} instT(x)$;
  - $instEV : EV \rightarrow 2^I$ is a function that relates an event-definition with a set of instances (events);
- $instG : G \rightarrow 2^I$ is a function that relates a gateway-definition with a set of instances (gateways);
- $instSA : SA \rightarrow 2^I$ is a function that relates an activity-definition that starts the workflow, with a set of activities that start the workflow;

- $instW : W \rightarrow 2^I$ is a function that relates a worker-definition with a set of instances (workers). Consequently, the set of all workers WI is defined as $WI = \bigcup_{\forall x \in W} instW(x)$;
- $instF : F \rightarrow 2^I$ is a function that relates an interface-definition with a set of instances (interfaces). Consequently, the set of all interfaces FI is defined as $FI = \bigcup_{\forall x \in F} instF(x)$;
- $S \sqsubseteq Z \wedge S \equiv \{notStarted, inProgress, withError, paused, canceled, finished\}$ is the set of possible states that an activity can have;
- $\Sigma : AI \rightarrow 2^{AI} \times P \times S$ is a function that defines for every activity (i) the following activities, (ii) its priority and (iii) its current state, where:
  - $transitionTo : AI \times AI$ is a relation that defines the following activities for every activity;
  - $hasPriority : AI \rightarrow P$ defines the priority value for every activity;
  - $hasState : AI \rightarrow S$ defines the current state for every activity;
- $execThrough : EI \rightarrow 2^{FI}$ is a function that defines, for every executable, the set of interfaces through which it was dispatched;
- $perfBy : TI \rightarrow 2^{WI}$ is a function that defines, for every task, the set of workers that participated in its execution;
- $instC : C \rightarrow 2^I$ is a function that relates a domain concepts with a set of instances. Consequently, the set of all domain instances CI is defined as $CI = \bigcup_{\forall x \in C} instCI(x)$;
- $\Phi : TI \rightarrow 2^{CI} \times 2^{CI}$ is a function that defines for every task (i) the input instances and (ii) the output instances, where:
  - $hasInput : TI \times CI$ is a relation that defines the input instances for every task;
  - $hasOutput : TI \times CI$ is a relation that defines the output instances for every task.

3.3 Interpretation

An interpretation of a CompFlow workflow-definition is a structure $\mathfrak{I} = (\Delta^{\mathfrak{I}}, A^{\mathfrak{I}}, W^{\mathfrak{I}}, F^{\mathfrak{I}}, P^{\mathfrak{I}}, S^{\mathfrak{I}}, C^{\mathfrak{I}}, I^{\mathfrak{I}})$, where:

- $\Delta^{\mathfrak{I}}$ is the domain set assumed to contain a single workflow;

- $A^{\mathfrak{I}} : A \rightarrow 2^{\Delta^{\mathfrak{I}}}$ is an activity-definition interpretation function that maps each activity-definition (task-definition, event-definition or gateway-definition) to a sub-set of the domain set. Accordingly, the following functions exist:
  - $E^{\mathfrak{I}} : E \rightarrow 2^{\mathrm{A}^{\mathfrak{I}}}$ is an executable-definition interpretation function that maps each executable-definition to a sub-set of $A^{\mathfrak{I}}$;
    - $T^{\mathfrak{I}} : T \rightarrow 2^{\mathrm{E}^{\mathfrak{I}}}$ is a task-definition interpretation function that maps each task-definition to a sub-set of $E^{\mathfrak{I}}$;
    - $EV^{\mathfrak{I}} : EV \rightarrow 2^{\mathrm{E}^{\mathfrak{I}}}$ is an event-definition interpretation function that maps each event-definition to a sub-set of $E^{\mathfrak{I}}$;
  - $G^{\mathfrak{I}} : G \rightarrow 2^{\mathrm{A}^{\mathfrak{I}}}$ is a gateway-definition interpretation function that maps each gateway-definition to a sub-set of $A^{\mathfrak{I}}$;
- $W^{\mathfrak{I}} : W \rightarrow 2^{\Delta^{\mathfrak{I}}}$ is a worker-definition interpretation function that maps each worker-definition to a sub-set of the domain set;
- $F^{\mathfrak{I}} : F \rightarrow 2^{\Delta^{\mathfrak{I}}}$ is an interface-definition interpretation function that maps each interface-definition to a sub-set of the domain set;
- $P^{\mathfrak{I}} : P \rightarrow \Delta^{\mathfrak{I}}$ is an instance interpretation function that maps each priority value to a single element in the domain set;
- $S^{\mathfrak{I}} : S \rightarrow \Delta^{\mathfrak{I}}$ is an instance interpretation function that maps each state to a single element in the domain set;
- $C^{\mathfrak{I}} : C \rightarrow 2^{\Delta^{\mathfrak{I}}}$ is an instance interpretation function that maps each domain concept to a sub-set of the domain set;
- $I^{\mathfrak{I}} : I \rightarrow \Delta^{\mathfrak{I}}$ is an instance interpretation function that maps each instance to a single element in the domain set.

An interpretation is a model of CompFlow if it satisfies the general set properties and instantiation properties. The general set properties are:

- $\forall a, i : a \in A \wedge i \in instA(a) \Rightarrow I^{\mathfrak{I}}(i) \in A^{\mathfrak{I}}(a)$, which implies:
  - $\forall e, i : e \in E \wedge i \in instE(e) \Rightarrow I^{\mathfrak{I}}(i) \in E^{\mathfrak{I}}(e)$;
    - $\forall t, i : t \in T \wedge i \in instT(t) \Rightarrow I^{\mathfrak{I}}(i) \in T^{\mathfrak{I}}(t)$;
    - $\forall ev, i : ev \in EV \wedge i \in instEV(ev) \Rightarrow I^{\mathfrak{I}}(i) \in EV^{\mathfrak{I}}(ev)$;
  - $\forall g, i : g \in G \wedge i \in instG(g) \Rightarrow I^{\mathfrak{I}}(i) \in G^{\mathfrak{I}}(g)$;
- $\forall w, i : w \in W \wedge i \in instW(w) \Rightarrow I^{\mathfrak{I}}(i) \in W^{\mathfrak{I}}(w)$;
- $\forall f, i : f \in F \wedge i \in instF(f) \Rightarrow I^{\mathfrak{I}}(i) \in F^{\mathfrak{I}}(f)$;
- $\forall c, i : c \in C \wedge i \in instC(c) \Rightarrow I^{\mathfrak{I}}(i) \in C^{\mathfrak{I}}(c)$.

The instantiation properties define the rules for workflow definition instantiations. They are:

- $\forall a1, a2, i, j : a1 \in A \wedge a2 \in A \wedge (a1, a2) \in AAR \wedge i \in instA(a1) \wedge j \in instA(a2) \Rightarrow transitionTo\left(I^{\mathfrak{I}}(i), I^{\mathfrak{I}}(j)\right)$;
- $\forall t, i, j : t \in T \wedge i \in instT(t) \wedge hasInput\left(I^{\mathfrak{I}}(i), I^{\mathfrak{I}}(j)\right) \Rightarrow \exists c : j \in c \wedge c \in C \wedge (t, c) \in CIN$;

- $\forall t, i, j : t \in T \wedge i \in instT(t) \wedge hasOutput\left(I^{\mathfrak{T}}(i), I^{\mathfrak{T}}(j)\right) \Rightarrow \exists c : j \in c \wedge c \in C \wedge (t, c) \in COUT$;
- $\forall t, i, j : t \in T \wedge i \in instT(t) \wedge perfBy\left(I^{\mathfrak{T}}(i), I^{\mathfrak{T}}(j)\right) \Rightarrow \exists w : j \in w \wedge w \in W \wedge (t, w) \in TR$;
- $\forall a, i, j : a \in A \wedge i \in instA(a) \wedge hasPriority\left(I^{\mathfrak{T}}(i), P^{\mathfrak{T}}(j)\right) \Rightarrow j \in P \wedge APR(a) = j$;
- $\forall e, i, j : e \in E \wedge i \in instE(e) \wedge execThrough\left(I^{\mathfrak{T}}(i), I^{\mathfrak{T}}(j)\right) \Rightarrow \exists f : j \in f \wedge f \in F \wedge (e, f) \in ER$.

4 Applications of CompFlow

The CompFlow upper ontology (as represented in the meta-model layer) establishes the building blocks for every workflow definition. With it, workflow-definition ontologies can be built by importing and extending the upper ontology to a specific domain while fully focusing on its specific logics. By extending tasks, events or interfaces, domain specific classes with different purposes can be created.

CompFlow can be applied to a variety of domains with several use case scenarios (e.g., human-machine computation scenarios and business workflows). In the scope of this work, a proof of concept will be presented regarding AAL. AAL scenarios normally incorporate a wide number of passive and active interaction modalities, such as sensors or actuators, touch, gestures or speech interfaces, or even location-tracking or fall-detection services [13, 16]. Given the highly dynamic nature of AAL environments, CompFlow provides the flexibility required to incorporate and maintain the cooperation of all the necessary components.

4.1 Scenario

For the purpose of a demonstration, imagine a scenario in which researchers want to assess the relevance of a help option within a certain application. For that, they establish the need to ask a simple question whenever the user interacts with the help option, thus building the workflow in fig. 3.

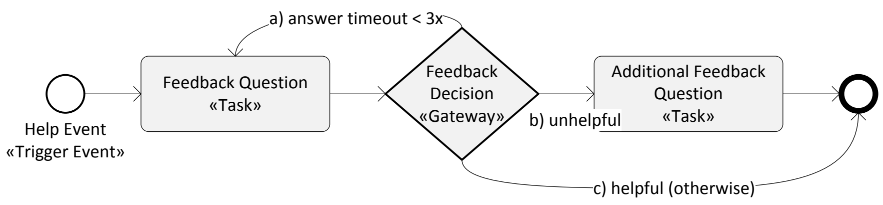

Fig. 3. AAL workflow for assessing the relevance of a help option in an application.

Given the AAL paradigm, and to introduce some level of redundancy, interaction can be made via multiple interfaces such as speech and graphical interfaces. Furthermore, it is envisaged that similar additional workflows (placing questions to users) will be required as the AAL environment evolves.

4.2 Workflow-Definition

CompFlow allows a straightforward implementation of this scenario through ontologies. It establishes the building blocks for creating semantically enriched workflow-definitions, while leaving the specificities of the domain entirely up to the developer. A CompFlow execution engine is then used to instantiate and execute any concrete workflow-definition.

Fig. 4 shows the implementation of the help option AAL scenario using CompFlow. The abstract workflow-definition establishes all domain constructs required to place questions to users. At this level, and upon the initialization of the execution engine, the code blocks that handle and know the domain must be associated with each task-definition. Analogously, each interface-definition has an associated code block that handles the specificities of the interface. This not only allows the execution engine to scale through the inclusion of interface components, but also the re-usability of domain specific task-definitions that may be included in different concrete workflow-definitions. Furthermore, it is possible to define composite interface-definitions, which allow multimodal interaction through the aggregation of multiple interface-definitions. This, in turn, makes it possible to achieve concurrency, redundancy and cooperation between multiple interface components.

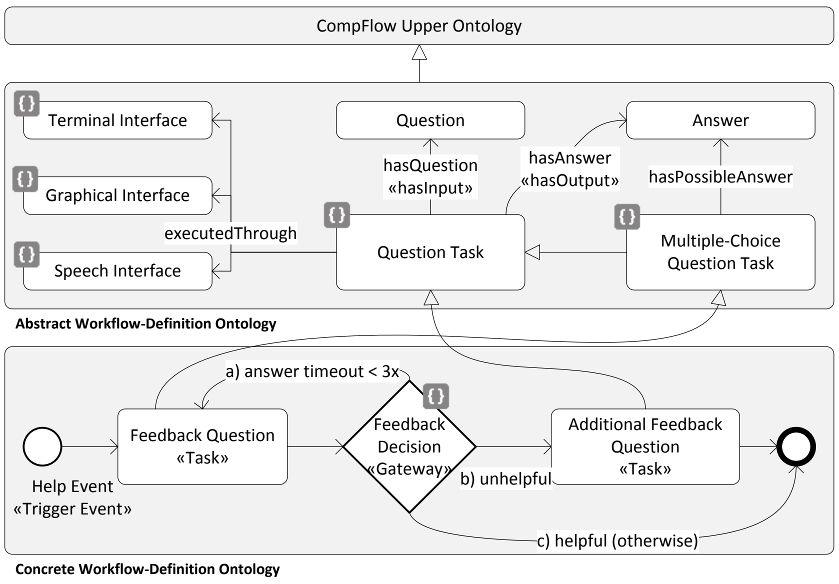

Fig. 4. CompFlow workflow-definitions for the help option AAL scenario.

The execution flow starts by making the execution engine wait for a specified event. Since it is a trigger event, each manifestation of the event will result in a new instantiation of the concrete workflow-definition. The event reaches the execution engine through its associated interface, typically following a publish-subscribe pattern.

After the event is processed, a Feedback Question Task is triggered. At this stage, the task is delivered to workers (of the specified role or worker-definition) using one of two strategies: (i) the slave strategy or (ii) the agent strategy. If the engine follows the slave strategy (i), a worker is automatically selected from the set of available workers. Otherwise, if the engine follows the agent strategy (ii), the task must be delivered to all available workers, which in turn will decide if they want to participate in the task.

When an answer to the Feedback Question Task is received, the workflow continues onto the Feedback Decision Gateway. The Feedback Decision Gateway is a DecisionOutputGateway, meaning that each output path will only be followed if a specific condition is met. The logics that decide this are introduced in an associated code block that evaluates the answer from the previous task. Depending on the answer, the workflow may end, or the Additional Feedback Question may be triggered. The later results in a process very similar to that of the Feedback Question Task, although it refers to a free text question instead of a multiple-choice question.

5 Conclusions and Future Work

The CompFlow process and upper ontology represents an approach to workflow-definition, instantiation and execution that exploits the benefits of Description Logic ontologies and technologies. Its nature allows the straightforward definition of semantically enriched workflows that are scalable and re-usable. The benefits of the CompFlow are relevant to multiple application domains and scenarios such as human-machine computation in general and AAL, in particular.

A formal definition of the meta-model (upper ontology) is given, which can be followed for the implementation of CompFlow compatible workflow execution engines. The structure and semantics provided by the ontology definitions allow these execution engines to deliver tasks to both human and machine workers able to understand the domain of knowledge. An early implementation of the CompFlow execution engine proves the applicability of CompFlow in AAL environments through the given scenario.

Given the minimalistic structure of the CompFlow upper ontology, new features will be added in the near future, which can be either considered as an application of CompFlow or, if generic enough, be assimilated into the proposed upper ontology and process. In the particular case of the later, a more thorough definition of gateway and task is envisaged in order to avoid the requirement of code blocks in gateway-definitions and task-definitions.

Acknowledgements. This work is partially funded by FEDER Funds and by the ERDF (European Regional Development Fund) through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT (Portuguese Foundation for Science and Technology) under the projects AAL4ALL (QREN13852) and FCOMP-01-0124-FEDER-028980 (PTDC/EEI-SII/1386/2012).

References

1. Eshuis R, Wieringa R (2001) A formal semantics for UML Activity Diagrams-Formalising workflow models.
2. Van Der Aalst WM, Ter Hofstede AH (2005) YAWL: yet another workflow language. Inf Syst 30:245–275.
3. Weske M (2001) Formal foundation and conceptual design of dynamic adaptations in a workflow management system. Syst. Sci. 2001 Proc. 34th Annu. Hawaii Int. Conf. On. IEEE.
4. Wodtke D, Weikum G (1997) A formal foundation for distributed workflow execution based on state charts. Database Theory—ICDT97. Springer, pp 230–246.
5. Vieira TAS, Casanova MA, Ferrao LG (2004) An ontology-driven architecture for flexible workflow execution. WebMedia -Web 2004 Proc. IEEE, pp 70–77.
6. Van der Aalst WMP (2003) Don't go with the flow: Web services composition standards exposed. IEEE Intell Syst 18:72–76.
7. Martin D, Paolucci M, McIlraith S, et al. (2005) Bringing semantics to web services: The OWL-S approach. Semantic Web Serv. Web Process Compos. Springer, pp 26–42.
8. Obrst L, Liu H, Wray R (2003) Ontologies for Corporate Web Applications. AI Mag 24:49.
9. Happel H-J, Seedorf S (2006) Applications of ontologies in software engineering. Proc Workshop Semat. Web Enabled Softw. Eng. ISWC. Citeseer, pp 5–9.
10. Oberle D (2009) How ontologies benefit enterprise applications. Semantic Web Journal - Interoperability, Usability, Applicability. IOS Press.
11. Luz N, Silva N, Novais P Construction of Human-Computer Micro-Task Workflows using Domain Ontologies. ESWC 2014 (Submitted).
12. Quinn AJ, Bederson BB (2011) Human Computation: A Survey and Taxonomy of a Growing Field. Proc. SIGCHI Conf. Hum. Factors Comput. Syst. ACM, New York, NY, USA, pp 1403–1412.
13. Teixeira A, Rocha N, Pereira C, et al. (2011) The Living Lab Architecture: Support for the Development and Evaluation of New AAL Services for the Elderly. Ambient Assisted Living Book. Taylor and Francis, USA (Accepted).
14. Eugster PT, Felber PA, Guerraoui R, Kermarrec A-M (2003) The many faces of publish/subscribe. ACM Comput Surv CSUR 35:114–131.
15. Baader F, Calvanese D, McGuinness DL, et al. (2007) The Description Logic Handbook: Theory, Implementation, and Applications, 2nd ed. Cambridge University Press.
16. Abraham A (2012) Special Issue: Hybrid Approaches for Approximate Reasoning. Journal of Intelligent and Fuzzy Systems 23:41-42.