Ontology-Extended Component-Based Workflows : A Framework for Constructing Complex Workflows from Semantically Heterogeneous Software Components

Jyotishman Pathak, Doina Caragea, and Vasant G Honavar

Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA
{jpathak, dcaragea, honavar}@cs.iastate.edu

Abstract. Virtually all areas of human endeavor involve workflows - that is, coordinated execution of multiple tasks. In practice, this requires the assembly of complex workflows from semantically heterogeneous components. In this paper, we develop ontology-extended workflow components and semantically consistent methods for assembling such components into complex ontology-extended component-based workflows. The result is a sound theoretical framework for assembly of semantically well-formed workflows from semantically heterogeneous components.

Keywords– Workflows, Ontologies, Components

1 Introduction

Almost all areas of human activity - education, business, health care, science, engineering, entertainment, defense - involve use of computers to store, access, process, and use information. The term *workflow* typically refers to coordinated execution of multiple tasks or activities [1, 12, 16]. Processing of an invoice, the protocol for data acquisition and analysis in an experimental science, responding to a natural disaster, could all be viewed as workflows.

Examination of workflows in specific domains reveals that many activities (e.g., the task of credit evaluation in financial services workflows) are common to several workflows. Encapsulation of such activities in the form of reusable modules or workflow *components*, which can be assembled to create complex workflows, can greatly reduce the cost of developing, validating, and maintaining such workflows [21]. Hence, component-based approaches to designing workflows has begun to receive considerable attention in the literature [6, 8, 15, 21].

A component [10, 19] is a piece of software that can be independently developed and delivered as a unit. Well-defined interfaces allow a component to be connected with other components to form a larger system. Component-based software development [9] provides a flexible and cost-effective framework for reuse of software components. Compositionality ensures that global properties of

the resulting system can be verified by verifying the properties of the constituent components. By analogy with software components, a workflow component can be viewed as a workflow module (i.e., an executable piece of software) which can be connected to other workflow modules through well-defined interfaces.

A simple workflow consisting of two simple components: F-Sensor and Weather Description is shown in Figure 1. The function of this workflow is to determine

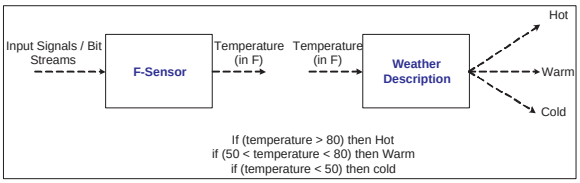

Fig. 1. Weather Description with F-Sensor

whether the day is hot, or warm or cold based upon the temperature. The input to the F-Sensor component consists of signals from one or more sensors and its output is the current temperature (in degree F) and the input to the Weather Description component is the current temperature (in degree F from the output of F-Sensor component) and its output is a description of the day (hot or warm or cold). Note that in this example, the output produced by the F-Sensor component has the same semantics as the input of the Weather Description component; furthermore, the name *Temperature* used in the vocabulary associated with the F-Sensor component has the same *meaning* as the term *Temperature* in the vocabulary associated with the Weather Description component. In the absence of syntactic or semantic mismatches between components, their composition is straightforward.

However, it is unrealistic to expect such syntactic and semantic consistency across independently developed workflow components libraries. Each such workflow component library is typically based on an implicit *ontology*. The workflow component ontology reflects assumptions concerning the objects that exist in the world, the properties or attributes of the objects, the possible values of attributes, and their intended meaning from the point of view of the creators of the components in question. Because workflow components that are created for use in one context often find use in other contexts or applications, syntactic and semantic differences between independently developed workflow components are unavoidable. For example, consider a scenario where we replace the F-Sensor component with a new component: C-Sensor. Suppose C-Sensor behaves very much like F-Sensor except that it outputs the temperature, denoted by *Temp*, and measured in degrees Centigrade instead of degrees Fahrenheit. Now we can no longer compose C-Sensor and Weather-Description components into a simple workflow, because of the syntactic and semantic differences between the two components. Effective use of independently developed components in a given context requires reconciliation of such syntactic and semantic differences between

the components. Because of the need to define workflows in different application contexts in terms of vocabulary familiar to users of the workflow, there is no single privileged ontology that will serve all users, or for that matter, even a single user, in all context.

Recent advances in networks, information and computation grids, and WWW have made it possible, in principle, to assemble complex workflows using a diversity of autonomous, semantically heterogeneous, networked information sources and software components or services. Existing workflow management systems (WfMS) allow users to specify, create, and manage the execution of complex workflows. The use of standard languages for defining workflows [1] provides for some level of portability of workflows across different WfMS. A major hurdle in the reuse of independently developed workflow components in new applications arise from the *semantic* differences between the independently developed workflow components. Hence, realizing the vision of the Semantic Web [2], i.e., supporting seamless access and use of information sources and services on the web, in practice calls for principled approaches to the problem of assembly of complex workflows from semantically heterogeneous components - the *workflow integration* problem.

The workflow integration problem can be viewed as a generalization of the *Information Integration* problem [14]. Hence, we build on recent developments in component-based workflow design [6, 8, 15, 21] to extend ontology-based solutions of the information integration problem [7, 18] to develop principled solutions to the workflow integration problem. Specifically, in this paper, we develop *ontology-extended workflow components* and *mappings* between ontologies to facilitate assembly of *ontology-extended component-based workflows* using semantically heterogeneous workflow components. The proposed ontology-extended component-based workflows provide a sound theoretical framework for assembly of semantically well-formed workflows from semantically heterogeneous information sources and software components.

2 Ontology Extended Component Based Workflows

2.1 Ontologies and Mappings

An *ontology* is a specification of *objects*, *categories*, *properties* and *relationships* used to conceptualize some domain of interest. In what follows, we introduce a precise definition of ontologies.

Definition (hierarchy) [4]: Let S be a partially ordered set under ordering $\leq$. We say that an ordering $\preceq$ defines a *hierarchy* for S if the following three conditions are satisfied:
(1) $x \preceq y \rightarrow x \leq y$; $\forall\ x,\ y \in S$. We say $(S, \preceq)$ is *better than* $(S, \leq)$),
(2) $(S, \leq)$ is the reflexive, transitive closure of $(S, \preceq)$,
(3) No other ordering $\sqsubseteq$ satisfies (1) and (2).

Example: Let $S = \{$*Weather, Wind, WindSpeed*$\}$. We can define the partial ordering $\leq$ on S according to the *part of* relationship. For example, *Wind* is part

of the *Weather* characteristics, *WindSpeed* is part of the *Weather* characteristics, and *WindSpeed* is also part of *Wind* characteristics. Besides, everything is part of itself. Thus, $(S, \leq) = \{(Weather, Weather), (Wind, Wind), (WindSpeed, WindSpeed), (Wind, Weather), (WindSpeed, Weather), (WindSpeed, Wind)\}$. The reflexive, transitive closure of $\leq$ is the set: $(S, \preceq) = \{(Wind, Weather), (WindSpeed, Wind)\}$, which is the only hierarchy associated with $(S, \leq)$.

Definition (ontologies) [4]: Let Δ be a finite set of strings that can be used to define hierarchies for a set of terms S. For example, Δ may contain strings like *isa, part-of* corresponding to *isa* or *part-of* relationships, respectively. An *Ontology* O over the terms in S with respect to the partial orderings contained in Δ is a mapping Θ from Δ to hierarchies in S defined according to the orderings in Δ. In other words, an ontology associates orderings to their corresponding hierarchies. Thus, if *part-of* $\in \Delta$, then $\Theta(part\text{-}of)$ will be the *part-of* hierarchy associated with the set of terms in S.

Example: Suppose a company K_1 records information about weather in some region of interest (see Figure 2). From K_1's viewpoint, weather is described by the attributes *Temperature, Wind, Humidity* and *Outlook* which are related to weather by *part-of* relationship. For example, *Wind* is described by *WindSpeed*. The values *Cloudy, Sunny, Rainy* are related to *Outlook* by the *is-a* relationship. In the case of a measurement (e.g., *Temperature, WindSpeed*) a unit of measurement is also specified by the ontology. In K_1's ontology, O_1, *Temperature* is measured in degrees Fahrenheit and the *WindSpeed* is measured in miles per hour. For contrast, an alternative ontology of weather O_2 from the viewpoint of a company K_2 is shown in Figure 3.

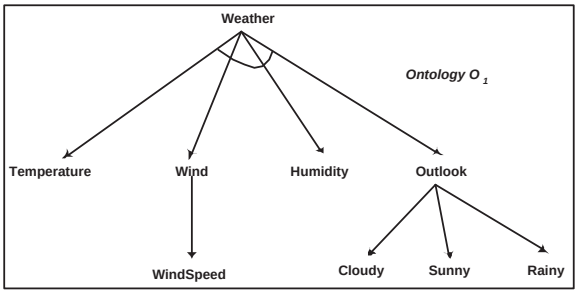

Fig. 2. Weather Ontology of Company K_1

Suppose O_1,...,O_n are ontologies associated with components C_1,...,C_n, respectively. In order to compose workflow using those semantically heterogeneous components, the user needs to specify the mappings between these ontologies of the various components. For example, a company K_3, with ontology O_3 uses meteorology workflow components supplied by K_1 and K_2. Suppose in O_3, *Weather* is described by *Temperature* (measured in degrees Fahrenheit), *WindSpeed* (measured in mph), *Humidity* and *Outlook*. Then, K_3 will have to specify a suitable

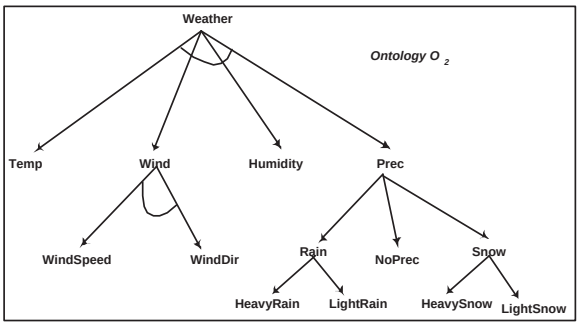

Fig. 3. Weather Ontology of Company K_2

mapping $M_{O_1 \mapsto O_3}$ from K_1 to K_3 and a mapping $M_{O_2 \mapsto O_3}$ from K_2 to K_3. For example, *Temperature* in O_1 and *Temp* in O_2 may be mapped by $M_{O_1 \mapsto O_3}$ and $M_{O_2 \mapsto O_3}$ respectively to *Temperature* in O_3. In addition, conversion functions to perform unit conversions (e.g. *Temp* values in O_2 from degrees Centigrade to degrees Fahrenheit) can also be specified. Suppose K_3 considers *Precipitation* in O_2 is equivalent to *Outlook* in O_3 and maps *Rain* in O_2 to *Rainy* in O_3. This would implicitly map both *LightRain* and *HeavyRain* in O_2 to *Rainy* in O_3. These mappings between ontologies are specified through *interoperation constraints*.

Definition (interoperation constraints) [4, 7]: Let $(H_1, \preceq_1)$ and $(H_2, \preceq_2)$, be any two hierarchies. We call set of *Interoperation Constraints (IC)* the set of relationships that exists between elements from two different hierarchies. For two elements, $x \in H_1$ and $y \in H_2$, we can have *one* of the following Interoperation Constraints:

- $x : H_1 = y : H_2$
- $x : H_1 \neq y : H_2$
- $x : H_1 \leq y : H_2$
- $x : H_1 \not\leq y : H_2$

Example: For the weather domain, if we consider *part-of* hierarchies associated with the companies K_1 and K_2, we have the following interoperation constraints, among others- *Temperature* : 1 = *Temp* : 2, *Outlook* : 1 = *Prec* : 2, *Humidity* : 1 $\neq$ *Wind* : 2, *WindDir* : 2 $\not\leq$ *Wind* : 1, and so on.

Definition (type, domain, values) [4, 7]: We define $T = \{\tau \mid \tau \text{ is a string}\}$ to be a set of *types*. For each type τ, $D(\tau) = \{v | v \text{ is a value of type } \tau\}$ is called the *domain* of τ. The members of $D(\tau)$ are called *values* of type τ.

Example: A type τ could be a predefined type, e.g. *int* or *string* or it can be a type like *USD* (US Dollars) or *kmph* (kilometers per hour).

Definition (type conversion function) [4, 7]: We say that a total function $f(\tau_1, \tau_2)$: $D(\tau_1) \mapsto D(\tau_2)$ that maps the values of τ_1 to values of τ_2 is a *type conversion function* from τ_1 to τ_2. The set of all type conversion functions satisfy the following constraints:

– For every two types τ_i, $\tau_j \in T$, there exists *at most* one conversion function $f(\tau_i, \tau_j)$.
– For every type $\tau \in T$, $f(\tau, \tau)$ exits. This is the identity function.
– If $f(\tau_i, \tau_j)$ and $f(\tau_j, \tau_k)$ exist, then $f(\tau_i, \tau_k)$ exists and $f(\tau_i, \tau_k) = f(\tau_i, \tau_j) \circ f(\tau_j, \tau_k)$ is called a composition function.

In the next few sections, we incorporate these definitions into our framework.

2.2 Component-Based Workflows

Definition (primitive component): A *primitive component* is a coherent package of software, that can be independently developed and delivered as a unit, and that offers interfaces by which it can be connected, unchanged, with other components.

Definition (component): A *component* can be recursively defined as follows:

– A primitive component is a component.
– The composition of two or more components is a component.

Definition (component-based workflow): A *component-based workflow* can be recursively defined as follows:

– A component is a workflow.
– The composition of two or more workflows is a workflow.

We can see that components are the building blocks upon which a workflow can be designed and composed.

Workflow Languages [1] facilitate precise formal specification of workflows. *Graph-Based Workflow Language* (GBWL) [20] allows us to model various aspects of traditional workflows which are relevant to our work. A GBWL specification of a workflow, known as *workflow schema* (WFS), describes the components of the workflow and the characteristics of the environment in which the workflow will be executed. The workflow schemas are connected to yield directed graphs of workflow schemas, called *workflow schema graphs* (WSG). The nodes of a WSG correspond to the workflow components and edges specify the constraints between the components. Figure 4 shows a WSG consisting of three components. Note that each workflow component trivially has a WSG description. When a workflow is to be executed, a WFS is instantiated resulting in the creation of a *workflow instance* (WFI). Each WFI created from a well-formed WFS is guaranteed to conform to the conditions specified by the WFS. The *functional aspect* of a workflow schema specifies the task to be performed by the corresponding workflow instances. The *information aspect* of a WSG specifies the data flow between the individual components. Associated with each component is a set of typed inputs and outputs. At the initiation of a workflow, the inputs are read, while on termination the results of the workflow are written to the outputs. The data flow which is defined in terms of the inputs and outputs, models the transfer of information through the workflow. For example, in Figure 4, component 1 has

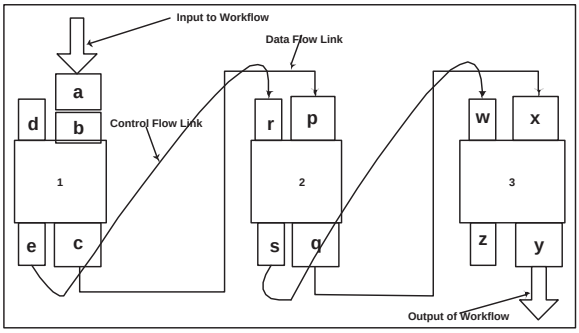

Fig. 4. Workflow Schema Graph

inputs a and b and an output c, and component 2 has an input p and an output q. Note that the data flow between components 1 & 2 is represented by the data flow link (c, p). The *behavioral aspect* of a WFS specifies the conditions under which an instance of the component will be executed. The behavior of a workflow is determined by two types of conditions: *Control* conditions and *Instantiation* conditions. The relation between the components is determined by the control conditions, which are expressed by the control flow links. These control flow links specify the execution constraints. For example, Figure 4 shows control flow links (e, r) specifying that the execution of component 1 has to precede the execution of component 2. In order for a workflow component to be executed, its instantiation conditions have to be set to True. Specifically, the existence of a control flow link from 1 to 2 does not imply that 2 will necessarily be executed as soon as 1 is executed (unless the instantiation conditions are satisfied). In general, it is possible to have cyclic data and control flow links. However, in the interest of simplicity, we limit the discussion in this paper to acyclic WSG.

2.3 Ontology-Extended Workflow Components

From the preceding discussion it follows that a workflow can be encapsulated as a component in a more complex workflow. Thus, to define *ontology-extended component-based workflows*, it suffices to show how components can be extended with ontologies and how the resulting ontology-extended components can be composed to yield more complex components (or equivalently, workflows).

Recall that a component has associated with it, input, output and control flow attributes. The control flow attributes take values from the domain $D(CtrlType)$ = {true, false, ϕ}, where the value of ϕ corresponds to the initial value of a control flow attribute indicating that the control flow link is yet to be signaled.

Definition (ontology-extended workflow component): An *ontology-extended workflow component*, s, consists of (see Figure 5):

- An associated ontology O_s.
- A set of data types τ_1, τ_2,..., τ_n, such that $\tau_i \in O_s$, for $1 \leq$ i $\leq$ n.

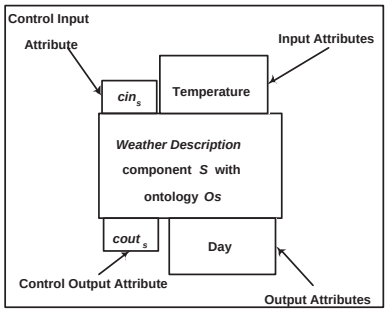

Fig. 5. Ontology-Extended Workflow Component

- A set of input attributes $input_s$ represented as an r-tuple (A_{1_s}:τ_{i_1},...,A_{r_s}:τ_{i_r}) (e.g., *Temp:C* is an input attribute of type Centigrade).
- A set of output attributes $output_s$ represented as a p-tuple (B_{1_s}:τ_{j_1},...,B_{p_s}:τ_{j_p}) (e.g., *Day:DayType* is an output attribute of type *DayType* whose enumerated domain is {*Hot, Warm, Cold*}).
- A control input attribute, cin_s, such that $\tau(cin_s) \in$ *CtrlType*. A true value for cin_s indicates that the component s is ready to start its execution.
- A control output attribute, $cout_s$, such that $\tau(cout_s) \in$ *CtrlType*. A true value for $cout_s$ indicates the termination of the execution of component s.

The composition of two components specifies the data flow and the control flow links between the two components. In order for the meaningful composition of ontology-extended workflow components to be possible, it is necessary to resolve the semantic and syntactic mismatches between such components.

Definition (ontology-extended workflow component composition): Two components s (source) (with an associated ontology O_s) and t (target) (with an associated ontology O_t) are *composable* if some (or all) outputs of s are used as inputs for t. This requires that there exists:

- A directed edge, called control flow link, $C_{link}(s,t)$, that connects the source component s to the target component t. This link determines the flow of execution between the components. We have:

$$C_{link}(s,t) \in cout_s \times cin_t,$$

  which means that there exists $x \in cout_s$ and $y \in cin_t$ such that $\tau(x) \in$ *CtrlType* and $\tau(y) \in$ *CtrlType*. For example, in Figure 4, (e, r) is a control flow link between the components 1 and 2.
- A set of data flow links, $D_{link}(s,t)$ from the source component s to the target component t. These links determine the flow of information between the components. We have:

$$D_{link}(s,t) \subseteq output_s \times input_t,$$

which means that there exist attributes $x \in output_s$ and $y \in input_t$, such that $\tau(x) = \tau_i \in O_s$ and $\tau(y) = \tau_j \in O_t$. For example, in Figure 4, $(c,\ p)$ is a data flow link between the components 1 and 2.
- A set of (user defined) interoperation constraints, $IC(s,t)$, that define a mappings set $MS(s,t)$ between outputs of s in the context of the ontology O_s and inputs of t in the context of the ontology O_t. Thus, if $x : O_s = y : O_t$ is an interoperation constraint, then x will be mapped to y, and we write $x \mapsto y$.
- A set of (user defined) conversion functions $CF(s,t)$, where any element in $CF(s,t)$ corresponds to one and only one mapping $x \mapsto y \in IC(s,t)$. The identity conversion functions may not be explicitly specified. Thus, $|IC(s,t)| \leq |CF(s,t)|$.

Note that, in general, a component may be connected to more than one *source* and/or *target* component(s). The mappings set $MS(s,t)$ and the conversion functions $CF(s,t)$ together specify a *mapping component*, which performs the mappings from elements in O_s to elements in O_t.

Definition (mapping component): A *mapping component*, $MAP(s,t)$, which maps the output and the control output attributes of the source s to the input and the control input attributes of the target t respectively, consists of:

- Two ontologies, O_s and O_t, where O_s is associated with the inputs of $MAP(s,t)$, and O_t is associated with its outputs.
- A set of mappings $MS(s,t)$ and their corresponding conversion functions $CF(s,t)$ that perform the actual mappings and conversions between inputs and outputs.
- A set of data inputs $input_{map}$=$(A_{1_M} : \tau_{s_1}, \cdots, A_{p_M} : \tau_{s_p})$, which correspond to the output attributes of component s, that is, $input_{map} \equiv output_s$. Also, τ_{s_1},..., τ_{s_p} is a set of data types such that $\tau_{s_i} \in O_s$, $\forall\ 1 \leq \mathrm{i} \leq p$.
- A set of data outputs $output_{map}$=$(B_{1_M} : \tau_{t_1}$,...,$B_{r_M} : \tau_{t_r})$, which correspond to the input attributes of component t, that is, $output_{map} \equiv input_t$. Also, τ_{t_1},...,τ_{t_r} is a set of data types such that $\tau_{t_i} \in O_t$, $\forall\ 1 \leq \mathrm{i} \leq r$.
- A control input cin_{map}, which corresponds to the control output attribute, $cout_s$ of component s. Also, $\tau(cin_{map}) = CtrlType$.
- A control output $cout_{map}$, which corresponds to the control input attribute, cin_t of component t. Also, $\tau(cout_{map}) = CtrlType$.

Ontology-extended workflow component instances (see Figure 6) are obtained by instantiating the ontology-extended workflow components at execution time. This entails assigning values to each of the component attributes. These values need to be of the type specified in the component schema. If a component instance *ins* is based on a component schema *sch* of the component s, we say that $hasSchema(ins) = sch$. We also say that for a given attribute, p, $v(p) \in D(t)$ refers to its value, if $\tau(p) = t \in O_s$.

Definition (ontology-extended workflow component instance): The instance corresponding to an ontology-extended workflow component s has to satisfy the following constraints:

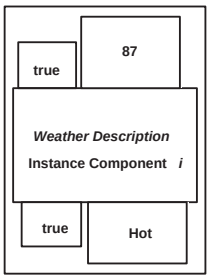

Fig. 6. Ontology-Extended Workflow Component Instance

- For every input attribute $x \in input_s$, $v(x) \in D(t)$, if $\tau(x) = t \in O_s$ (e.g., $Temperature = 87$).
- For every output attribute $y \in output_s$, $v(y) \in D(t)$, if $\tau(y) = t \in O_s$ (e.g., $DayType = Hot$).
- For the control input attribute, $cin_s \in$ {true, false, ϕ}, a true value indicates that the component s is ready for execution.
- For the control output attribute, $cout_s \in$ {true, false, ϕ}, a true value indicates that the component s has finished its execution.
- For an instantiation condition, $insc_s \in \{true, false\}$. If the evaluation of this condition returns *true*, then the execution of the component begins. This condition is defined as:

$$insc_s \equiv \{(cin_s)\ \Lambda\ (\forall\ x \in input_s, \exists\ v(x)\ )\},$$

such that $\tau(x) = t$ and $t \in O_s$.

Semantic Consistency of composition of ontology extended workflow components is necessary to ensure the soundness of component-based workflow assembly.

Definition (consistent workflow component composition): The composition of any two ontology-extended workflow components s (source) and t (target) is said to be *consistent*, if the following conditions are satisfied:

- The *data* & *control flow* between s and t must be consistent, i.e., control flow should follow data flow.
- The data and control flow links must be *syntactically consistent* i.e., there should be no syntactic mismatches for data flow links.
- The data and control flow links must be *semantically consistent* consistency, i.e., there should be no unresolved semantic mismatches along the data & control flow links. (Semantic mismatches between workflow components are resolved by mapping components)
- Data & control flow links should be *acyclic* (free of cycles).

Definition (ontology-extended workflow consistency): An ontology-extended workflow W is semantically consistent if the composition of each and every pair of *source* and *target* components is consistent.

Recall that the composition of two components s and t is consistent if it ensures data and control flow, syntactic, semantic and acyclic consistencies.

- *Data and Control Flow Consistency*: By the definition of the ontology-extended component composition, for any composition there exists a set data flow links $\in D_{link}(s,t)$ and there exists a control link $\in C_{link}(s,t)$. According to the definition of the ontology-extended component instance, the instantiation conditions $insc_t$ for t have to be satisfied, which means that all the inputs $\in input_t$ are instantiated when cin_t becomes true (it also means that $cout_s = true$). Thus, the control flow follows the data flow.
- *Syntactic and Semantic Consistency*: For every data flow link *(x, y)* $\in D_{link}(s,t)$, there exists a conversion function corresponding to a mapping introduced by an interoperation constraint (if such a function is not defined, it is assumed to be the identity). Thus, all the syntactic and semantic mismatches are resolved by the mapping component corresponding to the components s and t, and the syntactical and semantical consistency is ensured. Note that, for $C_{link}(s,t)$ there exists no syntactic differences.
- *Acyclic Consistency*: Our framework does not allow any cycles for data or control flow links.

3 Weather Description Workflow Example

In this section we illustrate ontology-extended component-based workflows using a sample workflow whose goal is to determine whether the day is hot, or warm or cold based upon the temperature (see Figure 7). This workflow is composed of two main components: C-Sensor component which calculates the current temperature upon the reception of the signals/bit streams from some external sensors and Weather Description component which determines the type of day (hot, warm, cold) based on the temperature. The two components are semantically heterogeneous and we show how they can be composed into a workflow using our framework.

Ontology-Extended Workflow Components: The components used in the sample workflow are described as follows:

- For the *source* component, C,
  - O_C is the associated ontology, which describes a *Sensor* by *Signals* and *Temp*, where $\tau(Signals) = Bits$ and $\tau(Temp) = Centigrade$.
  - $Bits, Centigrade \in O_C$ are the data types.
  - $input_C = (Signals : Bits)$.
  - $output_C = (Temp : Centigrade)$.
  - c_1 and c_2 are the control input and control output attributes, respectively.
- For the *target* component, W,
  - O_W is the associated ontology, which describes the *Weather* by *Temperature* and *Day*, where $\tau(Temperature) = Fahrenheit$ and $\tau(Day) = DayType$ and $D(DayType) = \{$Hot, Warm, Cold$\}$.

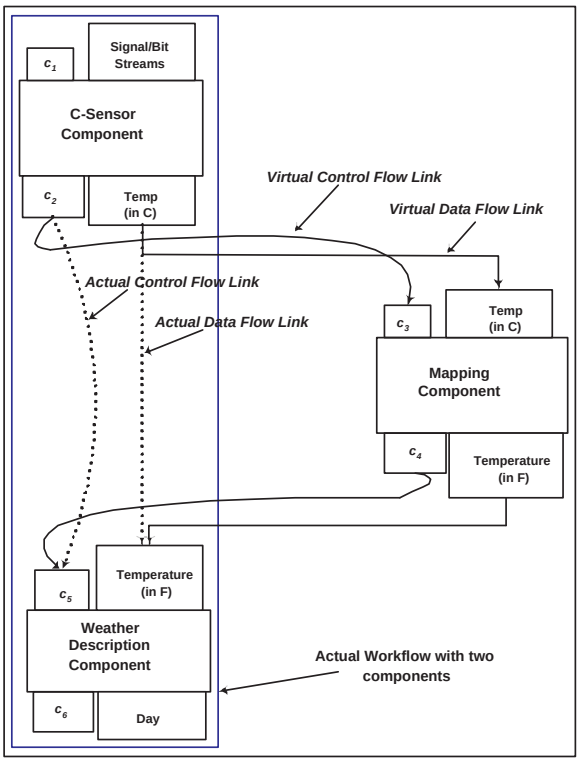

Fig. 7. Sample Workflow for Weather Description

- *Fahrenheit, DayType* $\in O_W$ are the data types.
- $input_W = (Temperature : Fahrenheit)$.
- $output_W = (Day : DayType)$.
- c_5 and c_6 are the control input and control output attributes, respectively.

Composition of Ontology-Extended Workflow Components: The composition of the components C and W is defined as follows:

- Ontologies O_C and O_W (defined above).
- The control flow link, $(c_2, c_5) \in C_{link}(C, W)$, where $c_2 \in cout_C$ and $c_5 \in cin_W$.
- The data flow link, $(Temp, Temperature) \in D_{link}(C, W)$, where $Temp \in output_C$ and $Temperature \in input_W$.
- The interoperation constraint, $Temp{:}O_C = Temperature{:}O_W$. Thus, there also exist a mapping from *Temp* to *Temperature*, denoted as, $(Temp \mapsto Temperature) \in MS\ (C, W)$.
- The conversion function, $f\ (Temp, Temperature) \in CF\ (C, W)$, which converts a value in Centigrade (*Temp*) to Fahrenheit (*Temperature*).

Mapping Component: Based on the interoperation constraints mentioned above, the mapping component, $MAP\ (C,\ W)$, can be defined as follows:

- Ontologies O_C and O_W are associated with its inputs and outputs, respectively.
- $(Temp \mapsto Temperature) \in MS(C,\ W)$ and $f\ (Temp,\ Temperature) \in CF(C,\ W)$ are the mapping and conversion function.
- $input_{map} = (Temp\ :\ Centigrade)$.
- $output_{map} = (Temperature\ :\ Fahrenheit)$.
- Attribute $c_3 \in cin_{map}$ is the control input attribute.
- Attribute $c_4 \in cout_{map}$ is the control output attribute.

Instances of Ontology-Extended Workflow Components: The instantiation condition for each of the component are shown below:

- *C-Sensor component*: The instantiation condition of the C-Sensor component is given by:

$$insc_C \equiv \{(c_1)\ \Lambda\ (\ \exists\ v(signals)\ )\} = \text{true}$$

  The value of c_1 is considered to be true whenever the C-Sensor receives signal/bit streams from some external sensor(s). That is, the instantiation condition of the component instance is evaluated when the component receives signals, and if it evaluates to true, the execution of the instance begins. The component does some internal processing with its input *signal streams* and outputs the current temperature *Temp*. Also, a true value at c_2 indicates the termination of execution of the component instance.
- *Mapping component*: The instantiation condition of the Mapping component is given by:

$$insc_{map} \equiv \{(c_3)\ \Lambda\ (\ \exists\ v(Temp)\ )\} = \text{true}$$

  The presence of a true value at c_2 (when C-Sensor component terminates its execution), results in a true value at c_3, of the mapping component. Upon successful evaluation of the instantiation condition of the component instance, the execution is initiated. On the termination of the component instance, it writes its output attribute *Temperature* (in degree F). Also, a true value at c_4, indicates the termination of execution of the component instance.
- *Weather Description component*: The instantiation condition for the Weather Description component is given by:

$$insc_W \equiv \{(c_5)\ \Lambda\ (\ \exists\ v(Temperature)\ )\} = \text{true}$$

  The termination of the execution of the mapping component places a true value in c_5 at an instance of the workflow component. A true evaluation of the instantiation condition of the component instance, initiates its execution. The input attribute of this component, *Temperature*, corresponds to the output attribute of the mapping component. Note that both these attributes

are syntactically and semantically identical. On termination of instance, it writes its output attribute *Day*. Finally, a true value at c_6 indicates the termination of execution of the component instance and also the termination of the workflow.

From the definitions above, the workflow is *consistent* because:

- The instantiation conditions ascertain that the control flow follows data flow.
- The mapping component guarantees that there are no syntactic and semantic mismatches.
- There are no cycles between data or control flow links.

4 Summary and Discussion

4.1 Summary

Recent advances in networks, information and computation grids, and WWW have made it possible, in principle, to access and use multiple, autonomous, semantically heterogeneous, networked information sources and software components or services. Development of tools that can contribute to substantial gains in productivity in specific application domains such as scientific discovery (e.g., bioinformatics), national defense (e.g., security informatics), business (e.g., e-commerce), manufacturing (e.g., virtual enterprises) and government calls for theoretically sound approaches for assembly of complex networks of coordinated activities or workflows from independently developed, semantically heterogeneous information sources and software components. Against this background, the framework of ontology-extended component-based workflows developed in this paper builds on recent advances in ontology-driven information integration [4, 7, 17, 18] and component-based workflows [6, 8, 15, 21] to address the need for a theoretically sound basis for composition of semantically heterogeneous workflow components into semantically consistent workflows.

4.2 Related Work

Benatallah *et al.* [3] introduce the Self-Serv framework for Web services composition. Their approach is based on two main concepts, namely, *composite service* and *service container*. The function of a composite service is to bring together various other services that collaborate to implement a set of operations, whereas, a service container facilitates the composition of a potentially large and changing set of services. However, the emphasis in this work has been more on the dynamic and scalable aspects of web service composition, and less on resolving semantic heterogeneity among the Web services, which remains as a major challenge in realizing the vision of the Semantic Web [2]. Cardoso *et al.* [8] provide metrics to select web services for composition into complex workflows. These metrics take into account various aspects like purpose of the services, quality of service (QOS) attributes, and the resolution of structural and semantic conflicts.

Fileto [11] designed the POESIA framework for Web service composition using an ontological workflow approach. POESIA uses domain specific ontologies for ensuring semantic consistency in the composition process.

Our approach is similar to the approach in [11], where ontologies are used for component (or Web service) composition, and hence, for bridging the semantic gap between them. However, we allow users to specify the interoperation constraints and define the type conversion functions between attributes in different domains, thereby supporting flexible resolution of semantic mismatches between the distributed, heterogeneous and autonomous components.

4.3 Future Work

Some directions for future work include:

- Design and implementation of an environment for workflow assembly and execution from semantically heterogeneous software components, ontologies, and user-supplied mappings between ontologies.
- Development of an approach to verification of consistency of user-specified interoperation constraints using Distributed Description Logics [5, 13].
- Development of workflows for specific data-driven knowledge acquisition from autonomous, distributed information sources in computational molecular biology applications (e.g., discovery of protein sequence-structure-function relationships).
- Analyzing the dynamics and behavioral aspects of workflow execution.

Acknowledgment. This research was supported in part by grants from the National Science Foundation (0219699) and the National Institutes of Health (GM066387) to Vasant Honavar.

References

[1] The Workflow Reference Model: (http://www.wfmc.org/)
[2] Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (2001)
[3] Benatallah, B., Sheng, Q., Dumas, M.: The self-serv environment for web services composition. IEEE Internet Computing **7** (2003) 40–48
[4] Bonatti, P., Deng, Y., Subrahmanian, V.: An ontology-extended relational algebra. In: Proc. IEEE International Conference of Information Reuse and Integration. (2003)
[5] Borgida, A., Serafini, L.: Distributed description logics: Assimilating information from peer sources. Journal of Data Semantics (2003) 153–184
[6] Bowers, S., Ludascher, B.: An ontology-driven framework for data transformation in scientific workflows. In: Intl. Workshop on Data Integration in the Life Sciences. (2004)
[7] Caragea, D., Pathak, J., Honavar, V.: Learning from Semantically Heterogeneous Data. In: 3rd International Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems (2004)

[8] Cardoso, J., Sheth, A.: Semantic e-workflow composition. Journal of Intelligent Information Systems **21** (2003) 191–225
[9] Cox, P., Song, B.: A formal model for component-based software. In: Proc. IEEE Symposia on Human Centric Computing Languages and Environments. (2001)
[10] D'Souza, D., Wills, A.: Object, Components and Frameworks with UML - The Catalysis Approach. Addison-Wesley, Reading, MA (1997)
[11] Fileto, R.: POESIA: An Ontological approach for Data And Services Integration on the Web. PhD thesis, Institute of Computing, University of Campinas, Brazil (2003)
[12] Fischer, L.: Workflow Handbook. Future Strategeis Inc., Lighthouse Point, FL (2004)
[13] Ghidini, C., Serafini, L.: Distributed first order logics. In: Frontiers of Combining Systems 2. Volume 7. (2000) 121–139
[14] Levy, A.: Logic-Based Techniques in Data Integration. Kluwer Academic Publishers, Norwell, MA (2000)
[15] Ludascher, B., Altintas, I., Gupta, A.: A modeling and execution environment for distributed scientific workflows. In: 15th Intl. Conference on Scientific and Statistical Database Management. (2003)
[16] Marinescu, D.: Internet-Based Workflow Management: Toward a Semantic Web. Wiley, New York (2002)
[17] Reinoso-Castillo, J.: Ontology driven information extraction and integration from autonomous, hetergoneous, distributed data sources - a federated query centric approach. MS. thesis, Department of Computer Science, Iowa State University, USA (2002)
[18] Reinoso-Castillo, J., Silvescu, A., Caragea, D., Pathak, J., Honavar, V.: Information extraction and integration from heterogeneous, distributed and autonomous sources: A federated ontology-driven query-centric approach. In: Proc. IEEE International Conference of Information Reuse and Integration. (2003)
[19] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman, Reading, MA (1998)
[20] Weske, M.: Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects (Habilitationsschrift). PhD thesis, Fachbereich Mathematik und Informatik, Universitt Mnster, Germany (2000)
[21] Zhuge, H.: Component-based workflow systems development. Decision Support Systems **35** (2003) 517–536