Towards an ontology for substances and related actions

Björn Höfling[1], Thorsten Liebig[2], Dietmar Rösner[1], and Lars Webel[1]

[1] Otto-von-Guericke-Universität Magdeburg,
Institut für Wissens– und Sprachverarbeitung,
P.O.Box 41 20, D-39016 Magdeburg, Germany,
(hoefling,roesner,webel)@iws.cs.uni-magdeburg.de
[2] Abteilung Künstliche Intelligenz, Fakultät für Informatik,
Universität Ulm, D-89069 Ulm, Germany
liebig@ki.informatik.uni-ulm.de *

Abstract. Modelling substances in knowledge representation has to be different from the treatment of discrete objects. For example liquids need a different approach to individuation. We propose an ontology which represents physical states and other properties of substances in a uniform way. Based on this we describe how to model a hierarchy of actions that can deal with such substances. For these actions a general distinction is made with respect to the type of properties the actions are changing. Further we describe an implementation in description logic allowing especially the definition of actions by specialization of more abstract actions and the inheritance of pre- and postconditions.

1 Introduction

When knowledge in a technical application area is made explicit, i.e. represented formally in a computer readable way, substances often play an important role, e.g. to have a detailed model of the material a technical part is made of, or the substances which are necessary for its use. It would be helpful to have this knowledge available in a sharable and reusable way suitable for different purposes. One way to do this is by specifying an *ontology* for this domain. Such an explicit specification of a conceptualization ([Gru95]) helps to clarify the meaning of relevant entities in the domain and therefore allows a shared understanding between different applications.

We are modelling knowledge about products, e.g. for the automatic generation of multilingual technical documentation from a language independent representation of the relevant domain knowledge. For some technical devices the relevant domain knowledge may be completely represented through a model comprising the resp. object, their parts and actions for manipulating those parts (e.g. checking, replacing). In this case modelling discrete objects is sufficient.

* Part of this author's contribution was funded by a PhD scholarschip of the program "Graduiertenförderung des Landes Sachsen-Anhalt".

But in many realistic applications we have to adequately model substances that are part of a product and play a functional role there (e.g. engine oil, coolant, ...). In addition related maintenance actions operating with these substances need to be modeled (e.g. checking an appropriate substance level, adding some fluid, replacing a fluid, ...).

A traditional way of representing substances is as properties of concrete objects. For well delimited objects made of solid material this might be a sufficient approach. But it will run into problems when trying to take into account the physical state of liquids. For this state one has to decide which amount of a substance may be called an instance or object, because every part of a liquid also fulfills the necessary and sufficient conditions for being an object. This *individuation problem* has to be handled in a uniform way in order to be able to represent all kinds of substances independent of their physical state or other properties in a single ontology.

As objects and their substances may change some of their properties over time, like e.g. the physical state as a consequence of a rising or lowering of temperature, it becomes also necessary to decide upon the behaviour of the individuated substances. Some of the questions that arise include: What happens to two substances when they are mixed together? What happens when a liquid is distributed over several containers? These questions belong to a general category of the modelling of *actions* which are related to substances.

We should clarify the role of our application in the design decisions which have been made in the ontology: Our goal, the automatic generation of multilingual technical documentation, requires a language independent representation of the domain knowledge (due to multilinguality) and reusability via specialization of general concepts (in order to be able to adapt the generated documents to different kinds of users, levels of detail or discourse situations). In addition the qualitative simulation of the represented actions is a requirement, as it allows the testing of feasibility and completeness of sequences of actions and can even lead to the automatic generation of warning instructions when possible dangerous events are detected. Nevertheless we believe that our approach is a general one which can be reused in cases where the modelling of non-discrete objects and related actions is necessary.

The paper is structured as follows: Section 2 analyses in more detail the domain of substances and related actions. Originating from this analysis a toplevel ontology for substances is presented in section 3 which has been implemented in description logic. Next, a taxonomy for actions related to this ontology is introduced in section 4. The consequences which follow for the implementation of action hierarchies in description logics are the topic of section 5. The paper concludes with remarks on related work, a summary and outlook.

2 Domain analysis

A substance can be defined as a physical material from which something is made or which has a discrete existence. To illustrate major problems in the modelling of substances we give a simple example: A cup of water, standing in front of someone. What are the substances which are of importance in this situation and how should the be represented? The object referred to by the first noun 'cup' is made of a certain material (e.g. china), which could for example be represented as a property of the instance 'cup'. The second noun 'water' directly describes some amount of a substance. Is this to be modelled as an instance as well?

In linguistics a disctinction is made between *count expressions*, which refer to a discrete, well-delineated group of entities and *mass expressions*, which refer to something without making it explicit how its referent is to be individuated or divided into objects [PS89]. Mass and count expressions are in most cases nouns, but some authors classify also other expressions (like verbs) as count or mass expressions. Even if in natural language the type of referent of mass expressions can be left unspecified, for explicit representations in ontologies or knowledge bases one has to solve this individuation problem. For the cup we can say there is a cup-object, but can we say that there is a water-object (i.e. the amount of water in the cup)? Such a water-object is a fundamentally different kind of object because any part of it is also a water-object, which is not the case for the cup.

To create an ontology and to be able to distinguish between individuals and their categories we have to examine properties of substances. The question arises whether a property belongs to the material or the object made of the material. The following definitions manifest this distinction ([RN95]): *Intrinsic properties* belong to the very substance of the object rather than to the object as a whole (examples: density, boiling point, composition of its chemical elements). *Extrinsic properties* are specific for an indivualised object (examples: volume, weight, shape). Intrinsic properties remain the same for every part of an object because it is made of the same material. On the other hand extrinsic properties are not retained under subdivision.

In the following we will discuss only those properties of substances which we consider important enough to be represented at a very high level in a substance ontology, which help to solve the individuation problem and which are essential for categorizing operations on substances. One important distinction is *pure* vs. *mixed substances.* For pure substances general properties like composition of its chemical elements, melting point and boiling point are important. Mixed substances should be represented as a list of the included pure substances. Unfortunately many properties of mixed substances cannot be deduced from the properties of its (pure) components. Since the components may also be in different physical states (example: sparkling water as a mixture of a liquid and a gaseous substance) it can even be difficult to specify the physical state of a mixed substance.

Nevertheless the *physical state* is a very important distinctive attribute, because in physics most other properties of substances are related to whether they are in a solid, liquid or gaseous state. In which physical state an object of a specific substance manifests itself depends on its temperature and on its pressure which we neglect here for the sake of simplicity. A general difference between most solid substances on the one hand and liquid and gaseous substances on the other hand is that the latter are not bound to a certain shape and may require a container to avoid dispersion. For all three physical states there are other possible distinctions or types of appearance [Web98]:

solid: depending on cohesive and adhesive forces
  - powderous substances (like flour); no identifiable shape, so mass or volume or an embracing container have to be specified
  - granular substances (like sugar); either like powderous substances, or by external influence or forces pressed into a shape (lump sugar)
  - substances with tight connection (like iron); shape plays an important role, can only be changed by external forces
  - malleable substances (like plasticine); hold together but their shape can be easily modified

liquid: [Hay85] distinguishes 15 possible states of liquid substances categorized along the following dimensions: (lazy still, lazy moving, energetic moving); (bulk, divided);(on surface, in space, unsupported)

gaseous: Like liquid substances they do not have a predefined shape and require a container to be kept together. To specify a certain amount of a gas one has to mention pressure and temperature (or to use normalized values for both) in addition to volume.

These top-level distinctions are sufficient to solve the individuation problem for substances in a general way and to be able to model related actions. In this context an *action* can be defined as the discrete change of one or more properties of an object or a substance. In this paper we will not describe continous processes for substances (like flowing of water), instead we restrict ourselves to discrete states of substances and to actions where the state changes can be modelled in a discrete way. As with substances we will not be able to make a complete classification of actions but will analyse some major categories.

We distinguish between the following categories of actions based on the type of properties of substances that they are changing:

Substance-preserving actions: Only extrinsic properties of the objects are changed. The intrinsic properties of the related substances are preserved.

Substance-changing actions: Intrinsic (i.e. substance-specific) properties are changed, which means that the participating substances before and after the execution of the action differ (examples: mixing of different substances, chemical reaction between substances).

Instance-preserving actions: In these actions the participating instances remain the same before and after the execution of the action (examples: movement of an object, or pouring of a liquid into another container).

Instance-changing actions: They modify essential properties of an object and also result in the destruction or creation of instances (examples: division or putting together quantities of substances).

The last distinction between instance-changing and instance-preserving actions is also motivated by a distinction of the extrinsic properties changed. Those extrinsic properties which are essential for an object (i.e. when they are changed, the instance will not remain the same; we will call them *existential properties*) must be distinguished from those which have no fundamental influence on the existence of an instance (we will call them *non-existential properties*). It ist dependent on the context whether a substance property is existential or not. In solid or liquid substances changing the property 'volume' is normally an instance-changing action because some part of it has been separated from the original object. As gases can be easily compressed changing the volume can also be an instance-preserving action for gaseous substances. In the former case the volume would be an existential in the latter case a non-existential property.

3 An ontology for substances

In this section we will propose a toplevel ontology for substances. Before describing our major distinctions and the reasons for these decisions we should clarify the requirements which lead to our ontology. They can be summarized as follows:

– Discrete objects and those for which an individualization is not obvious should be handled in a uniform way.
– For discrete objects the traditional way of instantiation and reference to a substance must be supported in order to be able to reuse existing representations.
– The ontology should be usable in dynamic contexts (i.e. the change of substance-related properties during actions).

The first requirement needs additional explanation. Intuitively people often treat all kinds of substances the same way. Therefore a separation of discrete objects and other kinds of substances seems artificial. Especially when not only static aspects but also dynamic changes are relevant. Why should an ice cube only begin to exist in the moment when the water freezes? Although the physical state has changed the individuated substance remains the same. In addition the modelling of e.g. a liquid only as a property of its container together with the degree of filledness (similar to other properties of the container) would complicate the treatment of transfering this liquid to another container or its identification in relation to a substitute (e.g. the oil in a motor before and after a change).

An ontology may be defined in an abstract way without using a concrete knowledge representation mechanism. However, since we want to be able to make actions related to substances executable we need an implementation basis. For this reason we chose PowerLoom, which is a very expressive description logic system. PowerLoom accepts expressions using the full predicate calculus, extended with sets, cardinality, equality, and predicate variables [Mac94].

In the traditional approach to abstraction and inheritance there is the basic distinction between instances, i.e. the individual objects (e.g. my car – identified by its type and license number – or my dog, identified by its owner and name), and concepts, i.e. the collection or class of all objects sharing certain properties (e.g. the concept CAR as the class of all cars or the class DOG). Individual objects or instances are elements of their concepts (seen as sets); specialisation is a subset relation between classes.

For a uniform treatment of both discrete objects and substances we first have to work out a generalized concept of what may constitute an 'instance' and how it relates to the resp. concept (the individuation problem). A solution can be summarized as follows: The concept of a substance e.g. the concept 'water' is the abstraction comprising all occurrences of this substance in the universe which share their intrinsic properties. Substances are instantiated by specifying their extrinsic properties like a definite amount of the substance or by relating it to some container that contains the substance and thus implicitly restricts its amount.

This approach to the individuation of substances is in accordance with the linguistic treatment of the phenomena, especially the use of definite referring expressions:

– In a recipe you may e.g. first introduce the amount of ingrediences needed (e.g. 250 cl of milk, 25 gramm of butter, ...) and later use definite noun phrases to refer to these substances as if they were instances (e.g. Melt the butter ... Pour the milk ...).
– As soon as a container is introduced into a discourse, an amount of substance contained in it behaves as an instance (e.g. Warm up the engine ! CAUTION: The oil gets hot!)

We will call the most general substance concept which specifies only intrinsic properties *stuff* and the most general concept specifying only extrinsic properties *thing*. A category with both intrinsic and extrinsic properties has to be defined using (sub)concepts from both. The advantages of this factorisation of our ontology are:

– It is possible to augment already existing discrete objects with information about their substance by referring to the stuff hierarchy only.
– If one is interested only in intrinsic properties of a substance, for example to decide which material is particularly well suited for a certain function of an object, this can be described without using the thing-part of the ontology.
– A combination (through inheritance) of both hierarchies allows the uniform modelling of individuals for all kinds of substances.

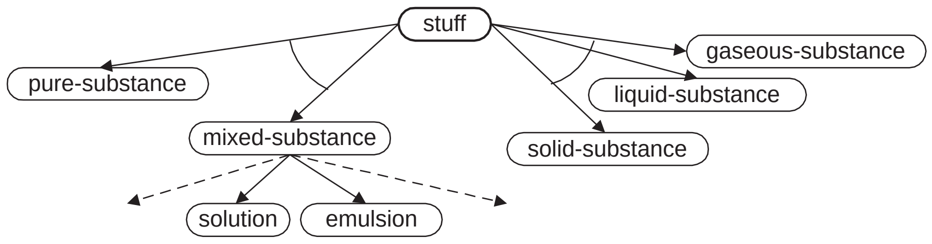

Fig. 1. *stuff hierarchy*

The stuff hierarchy[1] (cf. Figure 1) distinguishes at the toplevel between the following subconcepts: Pure and mixed substances are important for being able to model actions where more than one substance participate and because of the mentioned problem of not being able to make general inferences from properties of the components. For mixed substances only two examples are given, emulsion and solution. The distinction between three physical states is made because typical intrinsic properties often depend on their physical state (e.g. colour, conductivity, chemical reactivity, etc.). The divisions in the abstract ontology have been useful in modelling examples from specific technical domains [Web97].

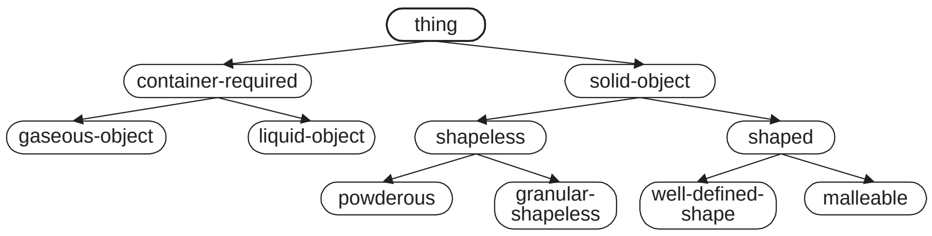

Fig. 2. *thing hierarchy*

Within the thing hierarchy (cf. Figure 2) a distinction is made between objects that require a container and those which do not. The former are specified by their mass or volume and can inherit properties like shape from their container. The temperature of an object (cf. Figure 3) is modelled as an extrinsic property

[1] We use the following notational conventions: Normal arrows describe a class/subclass relation (in the sense of subset of the instances). Arrows connected by an arc describe a disjunctive partition. For the concept `mixed-substance` we only mention two subconcepts as examples, the dashed arrows indicate that there exist other subconcepts which are not shown.

(it is only relevant for concrete instances) and as the physical state depends on this fact (in relation to the intrinsic properties melting point and boiling point) the latter may also be seen as an extrinsic property. Therefore we decided to model the physical state in the thing hierarchy, too but as a direct consequence of the extrinsic property temperature. Solid objects that do not require a container can be subdivided into shapeless and shaped which are generalizations of the four categories powderous, granular, malleable and well defined shape (cf. section 2). We do not consider shapeless objects as being inevitably container dependent because we should also be able to model a pile of sand without needing a container. A container is an example of an object with well defined shape.

Figure 3 shows the definition of some of the upper concepts of the stuff and thing hierarchy (figure 1 and 2 resp.) in PowerLoom. The syntax of PowerLoom is a variant of KIF3.0 [GF92]. `?self` is the default variable used to refer to the concept itself.

```
(defclass stuff ()
  :slots ((melting-point :type Integer)
          (boiling-point :type Integer)
          (ingredients :type (set of chemical-substance))))

(defclass pure-substance (stuff)
  :<=> (= (cardinality (ingredients ?self)) 1))

(defclass mixed-substance (stuff)
  :<=> (> (cardinality (ingredients ?self)) 1))

(defclass thing ()
  :slots ((made-of :type stuff)
          (temperature :type Integer)))

(defclass solid-object (thing)
  :<=> (> (melting-point (made-of ?self)) (temperature ?self)))

(defclass gaseous-object (container-required)
  :<=> (< (boiling-point (made-of ?self)) (temperature ?self)))

(defclass liquid-object (substance-thing)
  :<=> (and (> (boiling-point (made-of ?self)) (temperature ?self))
            (< (melting-point (made-of ?self)) (temperature ?self))))
```

Fig. 3. *Excerpts from the thing and stuff ontology in* PowerLoom.

4 Towards a taxonomy for substance-related actions

Based on the ontology for substances and its factorisation into the stuff and thing hierarchy we can now describe how actions related to substances can be modelled. The main distinction for actions (cf. section 2) is between substance-preserving actions (where intrinsic properties remain the same and extrinsic may change) and substance-changing actions (where intrinsic properties can change). Concerning the extrinsic properties a change of existential properties leads to instance-changing actions and if only non-existential properties are changed to instance-preserving actions.

Figure 4 shows the general taxonomy for substance related actions and an example for each type of action. There may exist several intermediate action categories between the top-level action categories and the examples (indicated by pointed arrows).

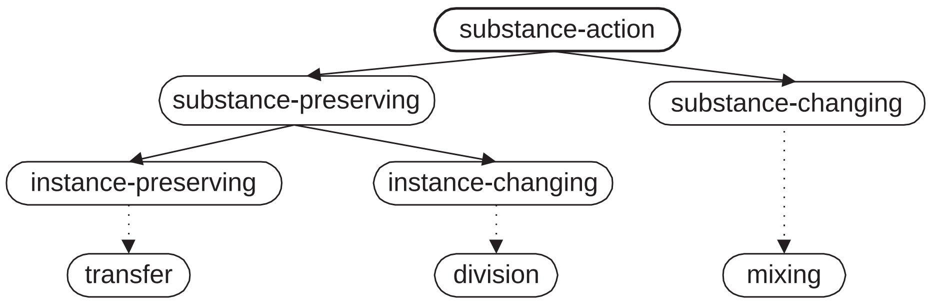

Fig. 4. *taxonomy for substance related actions*

In addition to this top-level taxonomy for actions we want to illustrate how actions on substances can be represented by giving a more specific example. It describes the different kinds of transfer of liquids from one container into another (cf. figure 5). How this can be implemented in PowerLoom is described in the following section.

The resp. concepts in the hierarchy for transfer actions are the following:

transfer: represents the transportation of a substance from one container into another. The second container may be filled partially with the same type of substance before the action has been carried out. This is a substance-preserving action (the same holds for all other subtypes) because only extrinsic properties like the referred container and potentially the volume are changed but the substances remain the same.

complete-transfer: specializes transfer in the aspect that the whole quantity of the first container is transferred to the second.

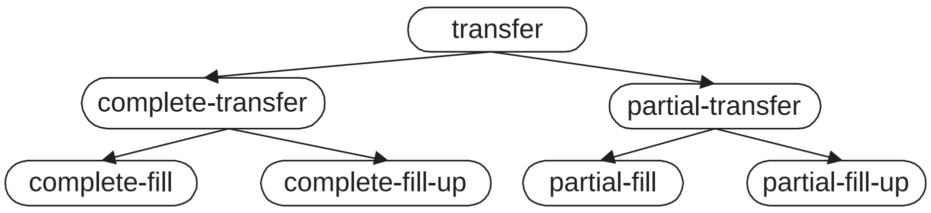

Fig. 5. *taxonomy for transfer actions*

complete-fill: has the additional constraint that the second container must be empty before the transfer. Since only non-existential properties of the substance (container) are changed, this is an instance-preserving action.

complete-fill-up: requires that the second container is filled by an amount of the same type of substance. Therefore it is an instance-changing action, because the substances in both containers are merged to one new substance in the second container.

partial-transfer: In contrast to complete-transfer it transfers only part of the substance from the first container. An existential extrinsic property (amount or volume) is changed and we have an instance-changing action (which is inherited by the actions partial-fill and partial-fill-up).

partial-fill: requires that the second container is empty before the transfer. The old substance is divided into two parts, one in the first and one in the second container.

partial-fill-up: requires that the second container must contain an amount of the same substance.

To illustrate the structure and the naming conventions of this taxonomy: The distinctive property for the first level is whether the first container is empty after the action has been carried out (named `complete-...`) or not (named `partial-...`). The distinction at the second level depends on the filledness of the second container before the action. If it has been empty it is named `...-fill`, in the other case `...-fill-up`. More complex actions like the distribution a substance into several empty new containers can be composed starting from these actions.

5 Action hierarchies in description logics

There are many different approaches for representing actions in object centered systems. For example, there are hierarchically organized action descriptions in systems for natural language processing (e. g. Penman Upper Model [BKMW90]). These descriptions classify actions by focusing mainly on the verb as the relevant object for classification. Action descriptions of this kind are well suited for natural language processing, but not sufficient for simulated execution. Other action descriptions in AI are related to the field of planning or plan recognition (e. g. RAT [HKNP92], T-REX [WL92]) and follow the STRIPS [FN71] approach. There, actions are interpreted as operators, mapping one world description into another. As a result of their operational description, actions can be executed for planning or simulation purposes. But there is no satisfying approach for defining actions by specializing more abstract actions.

In order to support the qualitative simulation of actions and to fulfill the requirements of object–centered languages, which are reusability, extensibility and understandability [Mey88], we propose an action representation which

– is *declarative* and *executable* (operational),
– allows *action* definitions by *specialization*,
– supports the *inheritance of pre- and postconditions* and
– results in a *hierarchical organization* of action descriptions.

Such an action representation allows the underlying inference mechanism to reason about actions in multiple dimensions. Performable actions, for example, are those which have a precondition which is true with respect to the current state of the world. Or one could ask for all those actions which fulfill a particular goal. All those answers are implicitly encoded in the action hierarchy and can be inferred by the classifier without much additional effort.

Consider the following fraction of a simplified action hierarchy as shown graphically in figure 5. Let us assume that all `transfer` actions change the location attribute of their action object[2], referenced here by the function `has-action-object`. The action `complete-transfer` inherits all properties (slots, pre- and postcondition, etc.) of `transfer`. The most relevant difference between these actions is, that the latter is defined to perform a complete transfer of the action object while specializing the former one. `complete-fill` is again more specific because this action assumes that the target container is empty.

Our work showed that one should be able to express conditions about attributes which are not known explicitly at time of description. This is useful in order to express abstract knowledge (consider an action `change` for example), shared by many different actions, but reified by different attributes (e. g. `transfer`, `change-temperature`). At the hierarchical level of `change` we have to abstract from the attribute we want to change because this could be either `has-location`, `has-temperature` or others. Nevertheless we want to specify pre- and postconditions for this abstract action. However, this requires second order features because we need to work with referenced relations, which are predicates in fact.[3] Second order features are not present in ordinary description logic sys-

[2] An action object, i.e. the object whose property is changed by an action, should not be confused with an instance of the category 'action'.
[3] What we actually need is unqualified existential quantification on predicates.

tems. In PowerLoom (as well as in KIF [GF92]) this can be done via the `holds` predicate[4]. The abstract action `change` could then look as in figure 6.[5]

```
(defaction change (action)
 :slots ((affected-attribute :type RELATION)
         (has-new-value :type UNKNOWN)
         (has-old-value :type UNKNOWN))
 :precondition (holds (affected-attribute ?self)
                      (has-action-object ?self)
                      (has-old-value ?self))
 :postcondition (holds (affected-attribute ?self)
                       (has-action-object ?self)
                       (has-new-value ?self)))
```

Fig. 6. Definition of `change`

The action `transfer` could then be defined as a specialization of `change` inheriting all slots, pre- and postconditions of `change`. According to the action hierarchy of figure 5 we define `complete-fill` (which is itself an indirect descendant of `transfer`) in figure 7.

```
(defaction transfer (change)
 :constraints (= (affected-attribute ?self) has-location))

(defaction complete-fill (complete-transfer)
 :slots ((has-new-value :type container)
         (has-old-value :type container))
 :precondition (and (empty (has-new-value ?self))
                    (>= (capacity (has-new-value ?self))
                        (amount (has-action-object ?self))))
 :postcondition (empty (has-old-value ?self))
```

Fig. 7. Definition of `transfer` and `complete-fill`

Due to the inheritance of the action parameter, pre– and postcondition and concretion of the affected attribute in `transfer`, the action `complete-fill` has actually the internal definition, given in figure 8 for the sake of completeness.

[4] The semantics of `holds` is defined in KIF and PowerLoom in the following way: If τ denotes a relation, then the sentence `(holds` τ τ_1 `...` τ_k`)` is true if and only if the list of objects denoted by τ_1,...,τ_k is a member of that relation.

[5] For sake of simplicity we omit all potential actions which may exist in the hierarchy between `action` and `change`.

```
(defaction complete-fill (complete-transfer)
  :slots ((has-action-object :type (and stuff thing))
          (has-new-value :type container)
          (has-old-value :type container))
  :precondition (and (empty (has-new-value ?self))
                     (has-location (has-action-object ?self)
                                   (has-old-value ?self))
                     (>= (capacity (has-new-value ?self))
                         (amount (has-action-object ?self))))
  :postcondition (and (empty (has-old-value ?self))
                      (has-location (has-action-object ?self)
                                    (has-new-value ?self))))
```

Fig. 8. Actual definition of `complete-fill`

Intuitively, the semantics of actions in general, and pre- and postconditions in particular, are straightforward with respect to PowerLoom semantics. Pre- and postconditions are semantically different from ordinary slots or relations for at least two reasons. First, there is an inherent relationship between them in the sense of a temporal ordering. Second, they characterize the action concept by expressing conditions about an instance (the action object) different from the action concept itself. Consequently the relationship between two actions has more dimensions than the relation between ordinary concepts. As a result, there are different subsumption relations between actions conceivable [LRn97]. The keywords `:precondition` and `:postcondition` were introduced in order to express these differences syntactically.

6 Related work

With respect to the analysis of *substances*, the distinction between count and mass expressions has for a long time been a subject in linguistic and philosophical literature (for an overview cf. [PS89]). The ontological distinction between thing and stuff motivated by intrinsic and extrinsic properties is adopted by many authors, for instance in the AI textbook of [RN95]. There exist many approaches for modelling special kinds of substances for domains which are motivated by the role substances play in certain application areas (e.g. the Plinius ontology for ceramic materials [vdVSM94]).

Concerning the *modelling of actions*, a system for the representation of actions and plans in a description logic (Rat – representation of actions using terminological logics, [HKNP92]) was developed in the WIP project [WAB$^+$92]. Pre- and postconditions of atomic actions are described by using a subset of the underlying description logic. They define conjunctions of feature restrictions, agreements, and disagreements. However, Rat does not support the specialization of actions, as it is not possible to define similar actions as special cases of a general action. In contrast to the Rat sytem, actions in Clasp [DL91] as well as in T-Rex [WL92] are primitive non-decomposable units. Yet, their language for composing plans is much richer. Another approach using Allen's temporal constraints is proposed in [AF97]. Action specialization is not possible in any of these systems.

The modelling of *actions related to substances* has been investigated by [Ter95] in the broader context of ontologies concerning processes or causes and effects. Especially the production or consumption of stuff has been treated but without considering the individuation problem for all physical states. Patrick Hayes was the first to define a detailed ontology for liquids [Hay85]. He solved the individuation problem for liquids by referring to a container and discussed actions by defining functions for modelling change and movements. He did, however, only consider what we call substance-preserving actions. [Dal92] analyses actions related to substances in the context of recipes. He argues that any mass object can be converted into a countable object by packaging operations. Further he proposes a representation for actions which admit decomposition and planning. [NH98] have created an ontology for the domain of experimental molecular biology where both substances and processes play a major role. In this domain, it is necessary to track substances through a series of experimental processes including transformations, which are modelled with the help of object histories.

None of the proposals just mentioned is able to define actions as a specialization of abstract actions. For the individuation of substances we have shown that all kinds of substances (especially in different physical states) can be treated uniformly in an ontology by a factorisation into an 'intrinsic' part (the stuff hierarchy) and an 'extrinsic' part (the thing hierarchy). One might not need this general approach for specific application domains but ignoring it could make extensions to include other kinds of substances very difficult.

7 Summary and outlook

In this paper we have described a proposal for the individuation of substances and for the modelling of actions in dynamic contexts. Due to technical reasons (the implementation status of PowerLoom is still very unstable and incomplete) up to now we have not been able to fully implement our ideas in a more or less complete ontology especially with respect to actions. Nevertheless we consider our approach an important step towards an ontology for substances and related actions.

Aspects similar to the individuation problem for substances can be found in other domains. The action of assembling a (technical) object from its parts will result in the creation of a corresponding instance. On the other hand, disassembling an object – e.g. for recycling – has the effect that the lifecycle of the object ends and the instance of the composite object ceases to exist.

There are some subtle issues related to the questions of what constitutes the identity of an instance and when the identity of an instance should change. Some even lead to paradoxa. For example we probably do not want to give up the identity of a non-trivial object (e.g. a car) when we replace a minor part of it (e.g. a spark-plug). But what about the case – already discussed by Greek philosophers – where we would step by step replace all parts that make up a compound object?

A related question with respect to instances of a substance: Assume that a fluid in a container continuously looses small amounts of substance. Will we create a new instance when we refill the more or less insignificant amount lost? Is there a difference to doing a significant fill-up (e.g. when more than half of the required amount has to be refilled)?

As often in issues of modelling there is no single and simple answer to these questions. The adequacy of the chosen granularity of a model has to be judged from the perspective of the application and the inferences needed. For the more abstract levels of an ontology this gives support for a 'strategy of least commitment', i.e. only those decisions should already be fixed on the ontological level that will not vary between different applications.

References

[AF97] Alessandro Artale and Enrico Franconi. A temporal description logic for reasoning about action and plans. In *Journal of Artificial Intelligence Research*, 1997.

[BKMW90] J. Bateman, R. Kasper, J. Moore, and R. Whitney. A general organization of knowledge for natural language processing: the penman upper model. Technical report, USC/ISI, 1990.

[Dal92] Robert Dale. *Generating Referring Expressions, Constructing Descriptions in a Domain of Objects and Processes.* MIT Press Cambridge, Massachusetts, 1992.

[DL91] Premkumar T. Devanbu and Diane J. Litman. Plan-based terminological reasoning. In J. F. Doyle, R. Files, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference (KR '91)*, pages 128 – 138, Cambridge, MA, April 1991. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Inteligence*, 2(3-4):189 – 208, 1971.

[GF92] Michael R. Genesereth and Richard E. Fikes. *Knowledge Interchange Format, V. 3.0, Reference Manual.* Stanford University, June 1992.

[Gru95] Thomas R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human Computer Studies*, 43:907 – 928, 5/6 1995. Also available as Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University.

[Hay85] Patrick J. Hayes. *Formal theories of the commensense world*, chapter Naive physics I: Ontology for Liquids, pages 71–107. Ablex Publishing Corporation, 1985.

[HKNP92] J. Heinsohn, D. Kudenko, B. Nebel, and H. Profitlich. RAT: representation of actions using terminological logics. Technical report, DFKI, Saarbrücken, 1992.

[LRn97] Thorsten Liebig and Dietmar Rösner. Action hierarchies for the automatic generation of multilingual technical documents. In Rémi Zajac, editor, *IJCAI-97 Workshop Ontologies and Multilingual NLP*, Nagoya, Japan, August 1997. International Joint Conference on Artificial Intelligence.

[Mac94] Robert M. MacGregor. A description classifier for the predicate calculus. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 213 – 230, 1994.

[Mey88] Bertrand Meyer. *Object–oriented Software Construction*. Prentice Hall, New York, 1988.

[NH98] Natalya Fridman Noy and Carole D. Hafner. Representing Scientific experiments: Implications for Ontology Design and Knowledge Sharing. In *15th National Conference on Artificial Intelligence (AAAI98)*, Madison Wisconsin, July 1998. AAAI Press.

[PS89] Francis Jeffry Pelletier and Lenhart K. Schubert. *Handbook of philosophical logic*, chapter Mass Expressions, pages 327–407. D. Reidel Publishing Company, 1989.

[RN95] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*, pages 241 – 243. Prentice Hall, 1995.

[Ter95] Paolo Terenziani. Towards a causal ontology coping with the temporal constraints between causes and effects. *International Journal for Human-Computer Studies*, 43(5/6):847–863, 1995.

[vdVSM94] Paul E. van der Vet, Piet-Hein Speel, and Nicolaas J. I. Mars. The plinius ontology of ceramic materials. *Workshop Notes ECAI'94 in Amsterdam, Workshop Comparison of Implemented Ontologies*, pages 187 – 205, 1994.

[WAB$^+$92] W. Wahlster, E. André, S. Bandyopadhyay, W. Graf, and T. Rist. WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation. In A. Ortony, J. Slack, and O. Stock, editors, *Communication from an Artificial Intelligence Perspective: Theoretical and Applied Issues*, pages 121 – 144. Springer-Verlag, New York, Berlin, Heidelberg, 1992.

[Web97] Lars Webel. Modellierung eines Teilgebiets der Domäne Werkstoffe für technische Produkte und Implementation in LOOM. Technical report, Otto-von-Guericke Universität Magdeburg, Institut für Informations- und Kommunikationssysteme, 1997.

[Web98] L. Webel. Untersuchungen zur Modellierung von Substanzen. Diplomarbeit, Otto-von-Guericke Universität Magdeburg, 1998.

[WL92] R. Weida and D. Litman. Terminological reasoning with constraint networks and an application to plan recognition. In Nebel, Swartout, and Rich, editors, *Proceedings of Principles of Knowledge Representation and Reasoning (KR'92)*, 1992.