Modelling of Reusable Product Knowledge in Terminological Logics:

a Case Study

Thorsten Liebig
Dietmar Rösner

Otto-von-Guericke-Universität Magdeburg
Institut für Informations- und Kommunikationssysteme
P.O. Box 4120, D-39016 Magdeburg, Germany
{tliebig|roesner}@iik.cs.uni-magdeburg.de

Abstract

Our project aims to automatically generating multilingual text for product maintenance, and
documentation tasks from a structured knowledge representation. The substructure consists of
both product specific as well as general technical domain knowledge. In particular, this is reusable
knowledge about product functions, geometry and their composition, represented in a terminologi-
cal knowledge representation system. The language independent representation is also designed to
allow for the qualitative simulation of the documentation steps.

This paper describes the implementation of some key modules, illustrated by a household toaster
example. Our main implementation goals were reusability, extensibility and understandability of
the representation. We discuss problems concerning the identification of relevant domain concepts,
the realisation of part/whole relationships with the capability to propagate qualities (from parts
to whole or vice versa), and the definition of electrical devices in order to allow for the qualitative
simulation of process steps.

1 Introduction

The availability of technical documentation for consumer goods and machinery in multiple languages
is becoming increasingly important, especially in the context of the European common market. The
huge amount of material to be translated often exceeds the capacities of human translators, and there
does not yet exist a sufficiently powerful machine translation technology. In addition, documentation
conforming to the product reliability regulations must guarantee the proper and safe use of a particular
product.

The TECHDOC project\(^1\) [14], [19] aims to automatically generate at least parts of multilingual doc-
umentation from a language independent representation of form and content. The generation is based
on language and product specific knowledge, as well as general technical domain knowledge, repre-
sented in a terminological logic. The knowledge has been acquired through the comparative analysis
of manuals in different languages [15]. When modelling product and technical domain knowledge, our
main goals were independence from specific natural languages, reusability, understandability, and ex-
tensibility of the representation, considered to be crucial factors of modelling quality in object-oriented
systems [12]. From the representational point of view, this is the basis for the following advantages:

\(^1\)The TECHDOC project is in progress at the FAW Ulm (Research Institute for Applied Knowledge Processing, Ulm,
Germany) since mid 1991. Most of the work is now continued at the ‘knowledge-based systems and document processing
group’ at the University of Magdeburg.
• Fast knowledge base updates in cases where a product changes only in details. The new documentation can then be generated just in time to reflect the actual status of product development.

• A new product of an existing application domain is easy to integrate into the knowledge base by composing or changing previously implemented parts.

• Hyper-Media outputs are easy to integrate into the generation process (text/graphics combinations, animation, etc.), by associating them with concepts in the knowledge base.

The underlying knowledge structure of instruction and maintenance texts can be well formalized by means of plans [15]. These plans consist of one or more actions, or other plans. \(^2\) Actions are characterized by pre- and postconditions. These conditions interact with the underlying actual object and situation status of the knowledge base. The availability of adequate and expressive actions depends, therefore, directly on the chosen model for objects, qualities and situations.

A detailed representation of actions allows for the qualitative simulation of the instruction steps in order to check for completeness and applicability of a plan (which is the basis for the document generation), or to automatically derive hints and warnings. Consider, for example, the consequences of the instruction for toasting bread using an household toaster: Parts of the toaster become extremely hot and/or are energized. With the background knowledge that the heating wire is a hot object which is not completely shielded from external access, the reasoning system classifies that as a dangerous situation and prompts the generator to create a warning. This example illustrates that for those kinds of reasoning tasks one will need a domain representation based on a model of physical structuring, as well as on dynamically changing qualities and status properties.

To be able to support both — the demands of the domain representation and the requirements of text generation — we employ knowledge representation on three levels.

The language dependent knowledge is organized on the topmost level. Our prototypical software system uses the sentence generator PENMAN [11] to produce natural language output\(^3\) from a semantic specification. PENMAN's domain and task independent ontology, called the upper model \(^2\), contains knowledge about objects, processes and qualities on a very abstract level. Below such a general, linguistically motivated classification we have been modelling a level which we call the middle model. This level contains knowledge about physical and functional objects (e.g. electrical objects, devices, connections) and their potential states (e.g. power-supplied, on, tightened) in the domain of technical products. Additionally, this level includes the principles which describe how these states can be achieved and how they influence other objects which are combined to form a complex whole. Finally, the domain model represents the lowest level and contains knowledge about particular technical devices in a concrete application.

In this paper we present a representation of reusable product knowledge suitable for our application purpose. Further we address the difficulties that confronted us when implementing that model in terminological logics. In the following section we argue for the chosen representation language. Then we present and discuss parts of the modelled knowledge of our example application, which is concerned with the maintenance of a household toaster.

2 The Representation Language

To realize the requirements listed above, a powerful formalism with appropriate semantics and inferential capacities is needed. Knowledge representation systems based on terminological logics are

\(^2\)The plans do contain additional information such as intended goal or a priori hints etc.

\(^3\)Currently in English, German and French.
qualified to build the formal framework for this kind of task. Our core model is a network of concepts and relations, connected to each other in a functionally and physically motivated manner.

We are using the knowledge representation tool **LOOM** [8], a descendent of the KL-ONE family, based on Common Lisp [6] and CLOS (Common Lisp Object System). **LOOM** is a good candidate for our representation purpose, because it offers more than just a very powerful terminological logic. Besides this, **LOOM** is already used (although not really exploited) for the 'upper model' of the sentence generator. In addition, **LOOM** provides a powerful deductive support with forward- and backward-chaining (including both strict and default reasoning, and automatic consistency checking), procedural programming, a first-order query language, production rules, multiple knowledge bases, and object-oriented methods [10]. In fact, the empirical analysis of six terminological representation systems in [5] showed **LOOM** to be the most expressive one.\(^4\)

The second important feature of a representation and reasoning system is, of course, its runtime performance. For terminological representation systems the classification time (time to compute the subsumption hierarchy of the concepts) is an interesting parameter of performance. The comparison with other terminological systems indicated **LOOM** to be one of the fastest systems (for detailed results see [5]).

It should be pointed out, however, that high expressiveness in connection with fast runtime performance has a price. **LOOM**'s classification algorithm is incomplete [9]. **LOOM** is explicitly advertised as supporting a very expressive terminological language in an incomplete manner. This deliberate decision of **LOOM**'s developers is not surprising, since it has been shown that determining subsumption between terms, which is needed for the classification process, is NP-hard [3] or even undecidable [17] for reasonably expressive languages.

### 3 A Case Study: Modelling a Toaster

For any adequate modelling of physical systems, it is important to identify the relevant phenomena, and to identify just the appropriate level of detail to model each phenomenon. Adequate models, therefore, incorporate abstractions and approximations that are well suited to the problem solving task [13]. Hence, a knowledge engineer must have the a priori information about the inferences to be drawn in the current application.

In the following discussion we will be dealing with a small world in which a toaster is prepared to work and used to toast bread. This scenario includes the installation of the toaster by establishing the power supply, the adjustment of the toast intensity selector, etc.

For identification of relevant phenomena and objects for an adequate model, we focused on objects from two different perspectives:

The **structural** perspective is used to organize objects as a result of their visible topology, in order to identify complex wholes, components and relations between them. To perform classification from this kind of perspective, it is necessary to identify the following:

- What is the appropriate level of detail for objects?
- What are the components of a complex object (e.g. the toaster's control appliance consists of a switch and a toast intensity selector)?
- What relations exist between objects (e.g. the electrical connection between the switch and the toast intensity selector)?

---

\(^4\)The terminological representation systems were: **BACK**, **CLASSIC**, **KRIS**, **LOOM**, **MESON** and **SB-ONE**.
The functional perspective specifies the result or goal of a device’s or component’s activity. Questions concerning this perspective are:

- What is the purpose of the particular object (e.g., the purpose of a toaster is to toast bread; the lever’s purpose is to switch the toaster on/off (see section 3.2))?
- How is the goal accomplished (e.g., connecting the connector with a plug establishes the toaster’s power-supply (see section 3.4 and 3.3))? 

3.1 Parts and Wholes

The part–whole relation plays a fundamental role in the description of complex objects. Franconi [4] defines the part–whole relation as a partial-ordering binary relation $\preceq$ with reflexive, anti-symmetric and transitive properties:

\[
\forall x. \; x \preceq (x, x).
\]

\[
\forall x, y. \; (x, y) \preceq (y, x) \rightarrow x = y.
\]

\[
\forall x, y, z. \; (x, y) \preceq (y, z) \rightarrow (x, z).
\]

A common way to interpret this role is by the ‘has–a’ paraphrase. But, as discussed in [1], part–whole relations cannot simply be modelled by ordinary attributes like price or color. The representation formalism should take into account their specific meaning and their transitivity rules. To illustrate this, consider the following example taken from [1]: “an arm is part of a musician, the musician is part of an orchestra, but it would sound a bit strange to state that the arm is part of the orchestra”.

Winston, Chaffin and Herrmann [20] proposed a distinction between six kinds of specialized part–whole relations, to overcome such apparent transitivity paradoxes: Component/Integral-Object, Member/Collection, Portion/Mass, Stuff/Object, Feature/Activity and Place/Area. The strangeness of the musician’s arm example is due to a mixing of two different part–whole relations [1].

The above distinctions allow more suggestive and differentiated reasoning behaviour using part–whole relationships, when only one single kind of part–whole relation is involved. It should be men-
tioned, however, that compositions of different part-whole relations can have relevant meanings too, but they are more difficult to handle.\textsuperscript{5}

The most important part-whole relation for our purposes is the component/integral-object relation, called has-direct-component in our system.\textsuperscript{6} This relation is characterized by the relationship between integral-objects which have a structure, and their components which have a specific functionality and which are separable. An example in our application domain is: ‘The control-appliance is a component of the toaster’, or ‘the darkness selector is a component of the control-appliance’. Incorporating these facts in our knowledge base one can ask if the instance darkness-selector-01 is part of the instance toaster-01 in the sense of the component/integral-object relationship:

\begin{verbatim}
> (ask (has-direct-component* toaster-01 darkness-selector-01))
\end{verbatim}

\textbf{T}

All specific part-whole relationships are modelled as specializations of a general part-whole relation has-direct-part. The transitive version of this relation does, therefore, hold between parts and wholes which have chains of different kinds of part-whole relations between them. The part-whole relations are modelled in Loom as follows:

\begin{verbatim}
(defrelation has-direct-part
 :domain object
 :range object
 :characteristics multiple-valued)

(defrelation has-d-component
 :is-primitive has-direct-part)

(defrelation is-d-integral-object
 :is (inverse has-d-component))

For the determination of transitive part-whole relationships, we defined additional relations capturing the transitive closure of each of the six specialized relations introduced above:\textsuperscript{7}

\begin{verbatim}
(defrelation has-d-component*
 :is (:satisfies (?x ?z)
 (or (has-d-component ?x ?z)
 (for-some (?y)
 (and (:not (:same-as ?x ?y))
 (has-d-component ?x ?y)
 (has-d-component* ?y ?z))))))

(defrelation is-d-integral-object*
 :is (:inverse has-d-component*))
\end{verbatim}

\textsuperscript{5}A partially filled composition table can be found in [16].

\textsuperscript{6}The transitive versions of the part-whole relations are marked with a ‘*’ as suffix.

\textsuperscript{7}The definition of the transitive closure includes the filtering of the reflexivity property of the part-whole relations in order to prevent cyclic paths.
As in Sattler's [16] concept language $\mathcal{P}$ for engineering applications, our part-whole relations are *direct* part-whole relations in the sense that they must satisfy the *immediate inferior* definition. As noted in [1], this constraint should affect the ABox reasoning process in order to discard non-intended models. In contrast to Sattler's application, which lacks ABox reasoning, we use data-driven rules in order to conform to the above definitions of immediate inferiors and anti-symmetry. The production rule for recognizing symmetric part-whole relationships is defined as follows:

```
(defproduction anti-symmetric-test
 :when (.:detects (:and (has-direct-part ?x ?y)
 (has-direct-part ?y ?x)
 (:not (:same-as ?x ?y))))
 :do ((format t "^Warning: symmetric part-whole relationship
 detected between '" a and '" a" ?x ?y)))
```

Part-whole relations can also be defined as *essential* by inheriting the role `has-essential-part` (see definition of `has-electrical-power-supply-facility` below). For example, one essential part of a household toaster is its power supply. Instances in the range of essential relations are then automatically classified as `Essential-Parts`. From the conceptual point of view, the presence of essential relations is useful for determining relevant components, in order to support the knowledge engineer in building his or her domain. Knowledge about essential parts is available at the concept level, so even when there are no instances created, the user can ask for the essential parts of a toaster. For example (compare with figure 1):

```
> (get-essential-parts 'toaster)
Role |R|HAS-TYPE-SPECIFICATION to: |C|OBJECT&TYPE-SPECIFICATION
Role |R|HAS-ROAST-APPLIANCE to: |C|ROAST-APPLIANCE
Role |R|HAS-CONTROL-APPLIANCE to: |C|CONTROL-APPLIANCE
Role |R|HAS-ELECTRICAL-POWER-SUPPLY-FACILITY to:
|C|ELECTRICAL-POWER-SUPPLY-FACILITY
```

Two of these parts are introduced in the definition of the concept `toaster`:

```
(defconcept Toaster
 :is-primitive (:and Electrical-Device Mechanical-Device
 (:exactly 1 has-roast-appliance)
 (:exactly 1 has-control-appliance))
```

The essential roles `has-electrical-power-supply-facility` and `has-type-specification` are inherited from the more abstract concepts `Electrical-Device` and `Device` respectively:

```
(defconcept Electrical-Device
 :is-primitive (:and Device Electrical-Object
 (:at-least 1 has-electrical-power-supply-facility)
 (:exactly 1 has-operation-status))
```

```
(defconcept Device
 :is-primitive (:and Object
 (:at-least 1 has-type-specification)))
```

---

*Given a partially ordered set $P$, we say that $a$ is an immediate inferior of $b$ if $a < b$, but $a < x < b$ for no $x \in P$. *
In the following definition, the relation has-electrical-power-supply-facility is characterized as a specialisation of the part-whole relation has-d-component, and its filler will be an essential part:

\[
\begin{align*}
& \text{(defrelation has-electrical-power-supply-facility} \\
& \quad \text{:is-primitive (and has-d-component has-essential-part)} \\
& \quad \text{:domain Electrical-Device} \\
& \quad \text{:range Electrical-Power-Supply-Facility} \\
& \quad \text{:characteristics (:single-valued))}
\end{align*}
\]

3.2 Functional Aspects

The representation we propose combines structural and functional information about a complex object (e.g., a device). As discussed in [7], functional structuring is useful for problem-solving mechanisms, which must often decompose the device's function into the functions of the components. The component/integral-object relation is an example of a structuring based on a physical organisation of components. In contrast, the functional specification describes the device's goals at a level of abstraction which is of interest at the object level [7].

We used an alternative model of a structural organization based on functional components. Consider, for example, the abstract concept Electrical-Power-Supply-Facility from above, which is not a physical object in the sense of a visible topology. Our model is therefore a mixture of physical and functional representation aspects (see also figure 1).

The function of a device is its intended purpose, which is achieved by behaviours [7]. Our model represents behaviour as the causal sequence of transitions of partial states/predicates. In the case of the Electrical-Power-Supply-Facility, the intended states are either power-supplied or not-power-supplied. The CONTROL-APPLIANCE achieves its purpose in having the status on or off (the default).

The determination of the actual states depends on the individual abstract object. A CONTROL-APPLIANCE is on if all of its switches are closed. The control appliance of a toaster usually consists of a binary switch (the lever) and a tuner (the toast intensity selector). The tuner is an exceptional switch in the sense that it is always closed. The concept on is therefore defined as follows:

\[
\begin{align*}
& \text{(defconcept on} \\
& \quad \text{:is (:satisfies ?x} \\
& \quad \quad \text{(and (Control-Appliance ?x))} \\
& \quad \quad \text{(for-all ?y (implies (has-d-component ?x ?y) (closed ?y))))})
\end{align*}
\]

The Electrical-Power-Supply-Facility, on the other hand, changes to power-supplied if the reasoning system detects a electrical-connection to one of its components.

The structural organization based on functional components supports the identification of relevant domain elements and the explicit introduction of (complex) wholes in the model.

According to [1], an additional minimal requirement of a conceptual model able to capture the ontological nature of both parts and wholes, is the ability to express “vertical” and “horizontal” relationships and constraints. We will address these issues in the following sections.
3.3 **Vertical Relationships**

If we consider the conceptual modelling perspective of our application, vertical dependencies can be differentiated into relationships between the *existence* of the whole and the existence of certain parts, and relationships between the *properties* of a whole and the properties of its parts.

Existential dependencies have been addressed in section 3.1 by proposing relations as essential. According to [18], an individual is *rigidly dependent* on another individual if the former cannot exist unless the latter exists; further, an individual is *generically dependent* on a class if, in order for the individual to exist, an instance of such class must exist. Since the essential parts of a toaster are generically dependent parts, we implemented “creation” methods for complex devices in order to instantiate essential parts automatically when instantiating the whole.

According to [1], the class of relationships between the properties of a whole and the properties of its parts (and vice versa) can be differentiated into:

- Properties which the parts inherit from the whole,
- Properties which the whole inherits from its parts,
- Properties of the parts which are systematically related to properties of the whole.

We describe each of these variants in turn:

**Properties which the parts inherit from the whole**

Certain locative properties of the whole also hold for its parts. Asserting `kitchen-01` as the place in which `toaster-01` is located at should also hold for all parts of this device:

```prolog
> (tellm (has-location toaster-01 kitchen-01))
T
> (retrieve ?x (has-location binary-switch-01 ?x))
(KITCHEN-01)
```

Note that the answer of the reasoning system to the retrieve statement should be interpreted to be non-exclusive. The `binary-switch-01` is, of course, located primarily in the toaster, but at a certain granularity and as a result of the part–whole relationships, the switch is also located in the kitchen. To get more detailed information, it is possible to write alternative retrieve statements to extract that `binary-switch-01` is part of `toaster-01` which is located in `kitchen-01`.

To enable the reasoning services discussed above, we need only define the relation `has-location` and call a macro:

```
(defunrelation has-location
 :range Location
 :characteristics (:single-valued :clip-roles))
```

The macro `left-distributive` creates a rule\(^9\) that strictly implies that the relation `has-location` holds between `?x` and `?y` if `?x` is a part of `?z` and the relation `has-location` holds between `?z` and `?y` for an object `?z` (see also figure 2):\(^10\)

\(^9\)The rule contains additional restricting clauses extracted from the domain and range of the underlying relation to prevent the reasoning mechanism from scanning the whole knowledge base in order to execute the implication rule.

\(^10\)In the terms from the example from above, `?x` would be `binary-switch-01`, `?y` would be `kitchen-01` and `?z` would be `toaster-01`.
Figure 2: The thin lines indicate part–whole relations; the dashed lines represent the has-location relation which is inherited left–distributive to all parts of ?z (dotted arrows).

(implies
 (:satisfies (?x ?y)
   (:and (object ?x)
     (location ?y)
     (:for-some ?z
       (:and (object ?z)
         (has-location ?z ?y)
         (is-direct-part-of* ?x ?z))))))

This macro expansion is very similar to the semantics of Franconi’s left distributive reading of his qualified plural quantifiers from the language ALC$S$ defined in [4]:

\[ \forall C.R(a, b) \text{ iff } \forall x. (\geq (a, x) \land C(x)) \rightarrow R(x, b) \]
\[ \exists C.R(a, b) \text{ iff } \forall x. (\geq (b, x) \land C(x)) \rightarrow R(a, x) \]
\[ \forall C.R(a, b) \text{ iff } \forall x. (\geq (a, x) \land C(x)) \rightarrow (\exists s. \geq (s, x) \land R(s, b)) \]
\[ \exists C.R(a, b) \text{ iff } \forall x. (\geq (b, x) \land C(x)) \rightarrow (\exists s. \geq (s, x) \land R(a, s)) \]

The $\forall$ and $\exists$ operators express the left and right distributive readings. They can be qualified by a qualification predicate $C$, which specifies that the relation necessarily holds only for all parts of a certain type $C$.\(^{11}\) We extend the terminological logic with macros that realize the left and right distributive operators by adding backward–chaining implication rules to the knowledge base.

Analogous to the plural quantifiers, the $\forall$ and $\exists$ operators introduce the cumulative plural quantifiers which hold for some parts covering all the qualified parts. These operators do not play any important role in our application framework and are therefore not implemented.

Since we also express properties in the form of concepts, we extend the qualified plural quantifiers with a distributive reading for concepts:

\[ \forall C(a) \text{ iff } \forall x. (\geq (a, x) \rightarrow C(x)) \]

\(^{11}\)Our macros assume $C \equiv \top$. 

Properties which the whole inherits from its parts

An example of properties which the whole inherits from its parts, in the toaster context, are on or power-supplied, which the toaster inherits from its power-supply-facility or control-appliance.

> (ask (on toaster-01))
NIL
> (ask (open binary-switch-01))
T

As the queries above show, the toaster toaster-01 is not on because binary-switch-01 is open in the current state of the knowledge base. In order to switch on the toaster the action adjust must be executed:

> (perform (adjust (find-instance 'binary-switch-01) 'on))
Setting binary switch |I|BINARY-SWITCH-01 to ON
T
> (ask (on toaster-01))
T
> (ask (open binary-switch-01))
NIL
> (ask (closed binary-switch-01))
T

Closing binary-switch-01 causes the control appliance to change to status on as defined in the non-primitive concept on from section 3.2. To inherit this property from this component to the whole we again use macros:

(inherit-up on)

With respect to relations we defined two additional macros called inherit-left-up and inherit-right-up. In all cases, the macro expansion adds additional backward-chaining implication rules to the knowledge base. The macro call from above would lead to the following rule:

(implies
 (:satisfies (?x)
  (:and (object ?x)
   (:for-some ?a
    (:and (on ?a)
     (has-direct-part* ?x ?a))))))

Implications of this kind are used in our application to realize the inheritance of properties from parts to the whole.

Properties of the parts which are systematically related to properties of the whole

For instance, the weight of a single part is always less than or equal to the weight of the whole. Capturing such phenomena is possible, but our model does not yet explore this kind of reasoning.
3.4 Horizontal Relationships

Horizontal relationships are composed of constraints among parts which characterize the integrity of the whole. Although they are important for capturing the notion of a whole, they find little attention in current modelling formalisms [1].

To determine the energized or hot parts of the toaster it is necessary to model the electrical connections of this device. The symmetric relation has-d-electrical-connection is used to model the electric connections between electrical objects. Electrical-Objects are objects whose energy type is electric:

```
(defconcept Electrical-Object
  :is (:and Object
         (:filled-by has-energy-type 'electric)
         (:exactly 1 has-electrical-consumer-status)
         (:exactly 1 has-voltage-specification)
         (:exactly 1 has-currency-specification)))
```

We distinguish between three kinds of electrical objects depending on their consumer status, namely producer, neutral and consumer. According to the requirement introduced in section 3.2, the toaster becomes power-supplied if there exists an electrical connection between the power-supply-facility and an electrical producer:

```
(defconcept Electrical-Power-Supply
  :is (:and Electrical-Object
       (:the has-electrical-consumer-status 'producer)))
```

Energized objects are thus characterized as electrical objects with any electrical connection to some Electrical-Power-Supply by taking the status of any switch on this path into account. The determination of a path along the (transitive) has-d-electrical-connection relation between electrical-objects is directly encoded in LISP as a depth-first search extended with additional filtering procedures. The following sequence shows the increasing number of energized objects due to switching on the toaster.\(^\text{12}\)

```
> (ask (on toaster-01))
NIL

> (retrieval ?x (and (object ?x) (energized ?x)))
( |I|PLUG-01 |I|CONNECTOR-01 |I|CABLE-01 |I|BINARY-SWITCH-01)

> (perform (adjust (find-instance 'binary-switch-01) 'on))
Setting binary switch |I|BINARY-SWITCH-01 to ON
T

> (retrieval ?x (and (object ?x) (energized ?x)))
( |I|PLUG-01 |I|CONNECTOR-01 |I|CABLE-01 |I|BINARY-SWITCH-01
 |I|TUNE-01 |I|HEATING-WIRE-01)
```

\(^\text{12}\)The additional inquiry clause (object ?x) in the retrieve statements is added for performance reasons; otherwise the whole knowledge base would have to be scanned in order to test on affiliation for the concept energized.
Since the HEATING-WIRE-01 is a converter from electrical to thermal energy, the inference system deduces the following:

> (ask (hot HEATING-WIRE-01))
T
> (perform (adjust (find-instance 'binary-switch-01) 'off))
Setting binary switch |I|BINARY-SWITCH-01 to OFF
T
> (ask (hot HEATING-WIRE-01))
NIL

This is implied by the rule which asserts that all energized electric to thermal converters are hot:

(implies (and energized electric-to-thermal-converter)
  hot)

In order to propagate this property upward to the toaster, we can use the macro introduced in section 3.3:

(inherit-up hot)

Switching the toaster on or off does now simultaneously influence the temperature status of the heating wire and the toaster. But this does not model the real world adequately to be able to derive knowledge about hot objects. For example, the heating wire does not become hot until it has been energized for a little while. Analogously, the wire remains hot for a few minutes after it is no longer energized.

To capture these regularities, we used Loom’s support for temporal concepts and relations, which was added in release 2.1. The temporal extensions allow us to make factual assertions about role fillers that hold only over specified intervals, rather than being universally true. It is also possible to define temporal concepts, where instances satisfy a concept only over some period of time. Loom’s model of time is based on fixed time points linked to an underlying time line.

Assuming that a heating wire becomes hot circa five seconds after being energized, and remains hot for at least two minutes after finishing toasting, we wrote the following production rules:  

(defconcept Hot :characteristics :temporal)

(defconcept Active-Thermical-Emiter)

(implies (:and Electrical-to-Thermical-Converter energized)
  Active-Thermical-Emiter)

(left-upwards Active-Thermical-Emiter)

(defproduction detecting-hot-object
  :when (:detects (Active-Thermical-Emiter ?x))
  :do ((let* ((now (get-universal-time))
          (start (+ now 5)))
       (tellm (:begins-at start (hot ?x))))))

13 We were forced to write production rules because Loom does not allow temporal operators in defined concepts.
(defproduction undetecting-hot-object
  :when (:undetects (Active-Thermical-Emiter ?x))
  :do ((let* ((now (get-universal-time))
             (end (+ now 120)))
       (tellm (:ends-at end (hot ?x)))))

Now, temporal queries must be used to ask for the temperature status of the heating wire and the toaster:

\[
\begin{align*}
&> (ask \ (on \ toaster-01))
&&\text{NIL} \\
&> (perform \ (adjust \ (find-instance \ 'binary-switch-01) \ 'on))
&&\text{Setting binary switch |I|BINARY-SWITCH-01 to ON}
&&T \\
&> (setq now \ (get-universal-time))
&&3048939527 \\
&> (ask \ (:holds-at \ now \ (hot \ heating-wire-01)))
&&\text{NIL} \\
&> (sleep \ 5)
&&\text{NIL} \\
&> (setq now \ (get-universal-time))
&&3048939535 \\
&> (ask \ (:holds-at \ now \ (hot \ heating-wire-01)))
&&T
\end{align*}
\]

4 Concluding Remarks

One of our first observations while modelling our application domain was the high start up costs. It takes a lot of concepts and relations to adequately model even simple objects. Later, having a basic set of concepts at hand made adding new complex objects much easier.

As a guideline for structuring complex objects, we propose a model using a physically oriented organization based on functional components. The transitive part-whole relation and the ability to inherit properties vertically and horizontally play key roles in this structuring. We enriched the representation language in order to simplify their definitions.

This skeleton can be used as a basis for a large variety of different modelling purposes, not only in the domain of technical products.

As a result of the expressiveness and the different paradigms which coexist within the LOOM knowledge base infrastructure, our model allows for a large variety of inferences. This is beneficial not only for qualitative simulation but also for other problem solving tasks.

Suggestions for future work includes a more conformed integration of temporal extensions into the model. Since a lot of properties are time-dependent, it seems inadequate to write production rules for every particular temporal concept or relation in order to assert temporal facts.

\[14\text{The lisp function get-universal-time returns the number of seconds since midnight, January 1, 1900 GMT [6].}\]
References


