INTERNATIONAL STANDARD

ISO 18629-44

First edition
2006-08-15

Industrial automation systems and integration — Process specification language —

Part 44: Definitional extension: Resource extensions

Systèmes d'automatisation industrielle et intégration — Langage de spécification de procédé —

Partie 44: Extension de définition: Extensions de ressource

Reference number
ISO 18629-44:2006(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Figures

Foreword

The International Organisation for Standardisation (ISO) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organisations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 18629-44 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data.*

A complete list of parts of ISO 18629 is available from the Internet :

http://www.tc184-sc4.org/titles

Introduction

ISO 18629 is an International Standard for the computer-interpretable exchange of information related to manufacturing processes. Taken together, all the parts contained in the ISO 18629 Standard provide a generic language for describing a manufacturing process throughout the entire production process within the same industrial company or across several industrial sectors or companies, independently from any particular representation model. The nature of this language makes it suitable for sharing process specifications and properties related to manufacturing during all the stages of a production process.

This part of ISO 18629 provides a description of the definitional extensions of the language related to activity extensions defined within ISO 18629.

All parts of ISO 18629 are independent of any specific process representation model used in a given application. Collectively, they provide a structural framework for improving the interoperability of these applications.

Industrial automation systems and integration — Process specification language —

Part 44: Definitional extension: Resource extensions

1 Scope

This part of ISO 18629 provides a specification of non-primitive concepts of the language, using a set of definitions written in the language of ISO 18629. These definitions provide an axiomatization of the semantics for terminology in this part of ISO 18629.

The following is within the scope of this part of ISO 18629:

— definitions of concepts specified in ISO 18629-11, ISO 18629-12 and ISO 18629-14 that are related to resources and resource sets and relations between resources and activities;

— definitions of concepts specified in ISO 18629-11, ISO 18629-12 and ISO 18629-14 that characterize relations between resources and activities.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 8824-1, *Information technology - Abstract Syntax Notation One (ASN.1) - Part 1: Specification of basic notation*

ISO 15531-1, *Industrial automation systems and integration - Industrial manufacturing management data - Part 1: General overview*

ISO 18629-1: 2004, *Industrial automation systems and integration – Process specification language – Part 1 : Overview and basic principles*

ISO 18629-11: 2005, *Industrial automation systems and integration – Process specification language – Part 11: PSL core*

ISO 18629-12, *Industrial automation systems and integration – Process specification language – Part 12: Outer core*

ISO 18629-14, *Industrial automation systems and integration – Process specification language – Part 14: Resource theories*

3 Terms, definitions, and abbreviations

3.1 Terms and definitions

For the purpose of this document, the following terms and definitions apply:

3.1.1
axiom
well-formed formula in a formal language that provides constraints on the interpretation of symbols in the lexicon of a language

[ISO 18629-1]

3.1.2
defined lexicon
set of symbols in the non-logical lexicon which denote defined concepts

NOTE Defined lexicon is divided into constant, function and relation symbols.

EXAMPLE terms with conservative definitions.

[ISO 18629-1]

3.1.3
definitional extension
extension of PSL-Core that introduces new linguistic items which can be completely defined in terms of the PSL-Core

NOTE: Definitional extensions add no new expressive power to PSL-Core but are used to specify the semantics and terminology in the domain application.

[ISO 18629-1]

3.1.4
extension
augmentation of PSL-Core containing additional axioms

NOTE 1 The PSL-Core is a relatively simple set of axioms that is adequate for expressing a wide range of basic processes. However, more complex processes require expressive resources that exceed those of the PSL-Core. Rather than clutter the PSL-Core itself with every conceivable concept that might prove useful in describing one process or another, a variety of separate, modular extensions need to be developed and added to the PSL-Core as necessary. In this way a user can tailor the language precisely to suit his or her expressive needs.

NOTE 2 All extensions are core theories or definitional extensions.

[ISO 18629-1]

3.1.5
grammar
specification of how logical symbols and lexical terms can be combined to make well-formed formulae

[ISO 18629-1]

3.1.6
language
combination of a lexicon and a grammar

[ISO 18629-1]

3.1.7
lexicon
set of symbols and terms

NOTE The lexicon consists of logical symbols (such as Boolean connectives and quantifiers) and non-logical symbols. For ISO 18629, the non logical part of the lexicon consists of expressions (constants, function symbols, and relation symbols) chosen to represent the basic concepts of the ontology.

[ISO 18629-1]

3.1.8
manufacturing
function or act of converting or transforming material from raw material or semi-finished state to a state of further completion

[ISO 15531-1]

3.1.9
manufacturing process
structured set of activities or operations performed upon material to convert it from the raw material or a semifinished state to a state of further completion

NOTE Manufacturing processes may be arranged in process layout, product layout, cellular layout or fixed position layout. Manufacturing processes may be planned to support make-to-stock, make-to-order, assemble-to-order, etc., based on strategic use and placements of inventories.

[ISO 15531-1]

3.1.10
primitive concept
lexical term that has no conservative definition

[ISO 18629-1]

3.1.11
primitive lexicon
set of symbols in the non-logical lexicon which denote primitive concepts

NOTE Primitive lexicon is divided into constant, function and relation symbols.

[ISO 18629-1]

3.1.12
process
structured set of activities involving various enterprise entities, that is designed and organised for a given purpose

NOTE The definition provided here is very close to that given in ISO 10303-49. Nevertheless ISO 15531 needs the notion of structured set of activities, without any predefined reference to the time or steps. In addition, from the point of view of flow management, some empty processes may be needed for a synchronisation purpose although they are not actually doing anything (ghost task).

[ISO 15531-1]

3.1.13
resource
any device, tool and means at the disposal of the enterprise to produce goods or services

NOTE 1 Adapted from ISO 15531-1. The concept of resource as defined in ISO 15531-1 includes an assumption seeing that resources except raw material, products and components that are considered from a system theory point of view as parts of the environment of the system and then do not belong to the system itself. That is not the case here. Furthermore ISO 15531-1 definition encompasses ISO 10303-49 definition but is included in the definition that applies for this part of ISO 18629 (In addition to ISO 15531 resources of this part of ISO 18629 resources include raw materials and consumables as well as in ISO 18629-14).

NOTE 2 Resources as they are defined here include human resources considered as specific means with a given capability and a given capacity. Those means are considered as being able to be involved in the manufacturing process through assigned tasks. That does not include any modelling of an individual or common behaviour of human resource excepted in their capability to perform a given task in the manufacturing process (e.g.: transformation of raw material or component, provision of logistic services). That means that human resources are only considered, as the other, from the point of view of their functions, their capabilities and their status (e.g.: idle, busy). That excludes any modelling or representation of any aspect of individual or common «social» behaviour.

[ISO 15531-1]

3.1.14
theory
set of axioms and definitions that pertain to a given concept or set of concepts

NOTE this definition reflects the approach of artificial intelligence in which a theory is the set of assumptions on which the meaning of the related concept is based.

[ISO 18629-1]

3.2 Abbreviations

— **KIF** Knowledge Interchange Format.

4 General information on ISO 18629

Part 41 to 49 of ISO 18629[1] specify definitional extensions needed to give precise definitions and the axiomatization of non-primitive concepts of ISO 18629. Definitional extensions are extensions of ISO 18629-11 and ISO 18629-12 that introduce new items for the lexicon. The items found in definitional extensions can be completely defined using theories of ISO 18629-11 and ISO18629-12. The definitional extensions provide precise semantic definitions for elements used in the specification of individual applications or types of applications for the purpose of interoperability. Definitional extensions exist in the following categories:

— Activity Extensions;

— Temporal and State Extensions;

— Activity Ordering and Duration Extensions;

— Resource Roles;

— Resource Sets;

[1] Certain parts are under development

— Processor Activity Extensions.

Individual users or groups of users of ISO 18629 may need to extend ISO 18629 for specifying concepts that are currently absent in parts 41 to 49 of ISO 18629. They shall use the elements presented in ISO 18629 for doing so. User-defined extensions and their definitions constitute definitional extensions but shall not become part of parts 41 to 49 of ISO 18629.

Note: User-defined extensions must conform to ISO 18629 as defined in ISO 18629-1:2004, 5.1 and 5.2.

Parts 41 to 49 of ISO 18629 provide:

— the semantic definitions, using concepts in ISO 18629-11and ISO 18629-12, of elements that are specific to the six concepts outlined above;

— a set of axioms for constraining the use of elements in definitional extensions.

They do not address:

— definitions and axioms for concepts that are part of ISO 18629-11 and ISO 18629-12;

— elements that are not defined using the elements in ISO 18629-11 and ISO 18629-12;

— user-defined extensions.

5 Organization of this part of ISO 18629

The definitional extensions that constitute this part of ISO 18629 are:

— Resource Roles;

— Capacity-based Concurrency;

— Resource Sharability;

— Resource Set-based Activities;

— Substitutable Resources;

— Homogeneous Sets;

— Inventory Resource Sets;

— Resource Pools;

— Processor Activities;

— Resource Paths.

All theories in this part of ISO 18629 are extensions of ISO 18629-14, itself an extension of ISO 18629-12 and ISO 18629-11.

6 Resource roles

This clause characterizes all definitions pertaining to Resource Roles.

In the context of ISO 18629 an object is a resource only with respect to some activity that requires the resource. Therefore no axiomatization of any other properties of resources, such as the issue of discrete vs continuous resources, is provided . Resource roles are one way of formalizing the way in which an activity requires the resource. The intuition behind the axiomatization of resource roles is the classification of interactions among activities with respect to the resources that they share. In particular, the set of resource roles defined in this clause are a classification of interfering actions, i.e., the effects of one action falsify the preconditions of another action.

NOTE : In the context of the system theory that supports standards such as ISO 15531 the resources are the means at the disposal of the system to transform input into output. Therefore they can not be considered as input or output of the system. They appear as a subset of the resources as they are defined here. Accordingly, in standards such as ISO 15531 they are not associated a priori to a given activity. A resource may be idle, associated to an activity A in the process 1 or to activity B in process 2. That is obvious for human resources, but it is also true for a lathe (for example) that may be associated to the activity “manufacture a cylinder” in process 1 or to the activity “drill a hole” in process 2. Anyway all definitions of this part of ISO 18629 apply to this type of resources also.

6.1 Primitive lexicon of the Resource roles

No primitive relations are introduced by the lexicon of Resource roles.

6.2 Defined lexicon for concepts of Resource roles

The following relations are defined in this clause:

— (reusable ?r ?a);

— (possibly_reusable ?r ?a);

— (renewable ?r ?a);

— (weakly_reusable ?r ?a);

— (consumable ?a);

— (possibly_consumable ?r ?a);

— (weakly_consumable ?r ?a);

— (wearable ?r ?a).

Each concept is described by informal semantics and a KIF axiom.

6.3 Core theories required by Resource roles

This extension requires

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

6.4 Definitional extensions required by Resource roles

No definitional extensions are required by the Resource roles.

6.5 Definitions of concepts for Resource roles

The following concepts are defined for Resource roles.

6.5.1 reusable

A resource ?r is reusable by an activity ?a if any other activity that also requires ?r is still possible to perform after ?a completes its occurrence, in every possible future.

EXAMPLE: A reusable resource is a machine that does not require setup between activities. As soon as one activity occurs, it is always possible to perform the next activity.

```
(forall (r ?a1 ?a2 ?a ?occ1 ?occ2) (iff (reusable ?r ?a1)
    (implies  (and  (common ?a1 ?a2 ?r)
                    (subactivity ?a1 ?a)
                    (subactivity ?a2 ?a)
                    (occurrence_of ?occ2 ?a1))
              (forall (?b)
                    (implies  (forall (?occ3)
                                    (implies  (and  (subactivity_occurrence ?occ3 ?b)
                                                    (occurrence_of ?b ?a)
                                                    (precedes ?occ2 ?occ3))
                                              (poss ?a2 ?occ3)))))))
```

6.5.2 possibly_reusable

A resource ?r is possibly reusable by an activity ?a iff for any other activity that also requires ?r is still possible to perform after ?a completes its occurrence, in some possible future situation.

EXAMPLE: A possibly reusable is a machine that requires some setup between different activities. After the first activity occurs, it is possible for the other activity, but only if the setup activity occurs first.

```
(forall (?r ?a1) (iff (possibly_reusable ?r ?a1)
(forall (?a2 ?occ1 ?occ2)
    (implies  (and  (common ?a1 ?a2 ?r)
                    (subactivity ?a1 ?a)
                    (subactivity ?a2 ?a)
                    (occurrence_of ?occ2 ?a1))
              (exists (?b)
                    (and    (exists (?occ3)
                                  (and    (subactivity_occurrence ?occ3 ?b)
                                          (occurrence_of ?b ?a)
                                          (precedes ?occ2 ?occ3))
                                          (poss ?a2 ?occ3)))))))
```

6.5.3 renewable

A resource ?r is renewable with respect to an activity ?a iff for any other activity that also requires ?r is still possible to perform after ?a completes its occurrence, in every possible future situation unless it is prevented.

EXAMPLE: A renewable resource is a solar-charged battery. Once it is depleted, there will always exist a future situation where the sun recharges the battery so that it can be used again.

```
(forall (?r ?a1) (iff (renewable ?r ?a1)
(forall (?a2 ?occ1 ?occ2)
    (implies  (and  (common ?a1 ?a2 ?r)
                    (subactivity ?a1 ?a)
                    (subactivity ?a2 ?a)
                    (occurrence_of ?occ2 ?a1))
              (forall (?b)
                    (implies  (exists (?occ3)
                                  (and    (subactivity_occurrence ?occ3 ?b)
                                          (occurrence_of ?b ?a)
                                          (precedes ?occ2 ?occ3))
                                          (poss ?a2 ?occ3))))))
```

6.5.4 weakly_reusable

A resource ?r is weakly reusable by an activity ?a iff for any other activity that also requires ?r is still possible to perform after ?a completes its occurrence, in every possible future situation unless it is prevented.

EXAMPLE: A weakly reusable resource is one where the renewing of the resource can be prevented. For example, a paintbrush is reusable only if it is put into a solvent after use; otherwise, it is not reusable.

```
(forall (?r ?a1) (iff (weakly_reusable ?r ?a1)
```



```
(forall (?a2 ?occ1 ?occ2)
    (implies  (and  (common ?a1 ?a2 ?r)
                    (subactivity ?a1 ?a)
                    (subactivity ?a2 ?a)
                    (occurrence_of ?occ2 ?a1))
              (exists (?b)
                    (and    (forall (?occ3)
                                  (implies  (and  (subactivity_occurrence ?occ3 ?b)
                                                  (occurrence_of ?b ?a)
                                                  (precedes ?occ2 ?occ3))
                                            (poss ?a2 ?occ3))))))
```

6.5.5 consumable

A resource ?r is consumable by an activity ?a if any other activity that also requires ?r is not possible to perform after ?a completes its occurrence.

EXAMPLE: A consumable resource is wood in a fire, or raw materials in a manufacturing production process.

```
(forall (?r ?a1) (iff (consumable ?r ?a1)
(forall (?a2 ?occ1 ?occ2)
    (implies  (and  (common ?a1 ?a2 ?r)
                    (subactivity ?a1 ?a)
                    (subactivity ?a2 ?a)
                    (occurrence_of ?occ2 ?a1))
              (forall (?b)
                    (implies  (forall (?occ3)
                                  (implies  (and  (subactivity_occurrence ?occ3 ?b)
                                                  (occurrence_of ?b ?a)
                                                  (precedes ?occ2 ?occ3))
                                            (not (poss ?a2 ?occ3))))))))
```

6.5.6 possibly_consumable

A resource ?r is possibly consumable with respect to an activity ?a1 iff after the occurrence of ?a1, there exists a future situation in which any activity that requires ?r is no longer possible.

```
 (forall (?r ?a1) (iff (possibly_consumable ?r ?a1)
(forall (?a2 ?occ1 ?occ2)
    (implies  (and  (common ?a1 ?a2 ?r)
                    (subactivity ?a1 ?a)
                    (subactivity ?a2 ?a)
                    (occurrence_of ?occ2 ?a1))
```

```
(exists (?b)
    (and    (exists (?occ3)
                (and    (subactivity_occurrence ?occ3 ?b)
                    (occurrence_of ?b ?a)
                    (precedes ?occ2 ?occ3))
                    (not (poss ?a2 ?occ3)))))))
```

6.5.7 weakly_consumable

A resource ?r is weakly consumable with respect to an activity ?a1 iff after the occurrence of ?a1, there always exists a possible future along which any other activity that requires ?r will never be possible.

EXAMPLE: A weakly consumable resource is a paintbrush; if we do put it into solvent after using it, then any activity that requires the brush will no longer be possible.

```
 (forall (?r ?a1) (iff (weakly_consumable ?r ?a1)
(forall (?a2 ?occ1 ?occ2)
    (implies  (and  (common ?a1 ?a2 ?r)
            (subactivity ?a1 ?a)
            (subactivity ?a2 ?a)
            (occurrence_of ?occ2 ?a1))
        (exists (?b)
            (and    (forall (?occ3)
                    (implies  (and  (subactivity_occurrence ?occ3 ?b)
                            (occurrence_of ?b ?a)
                            (precedes ?occ2 ?occ3))
                        (not (poss ?a2 ?occ3)))))))
```

6.5.8 wearable

A resource ?r is wearable with respect to an activity ?a1 if and only if after the occurrence of ?a1 there is always a situation in every possible future where any other activity that requires ?r is no longer possible.

EXAMPLE: A wearable resource is a drill bit; in every possible future, there will exist a situation where the bit has worn down to the point where it can no longer be used.

```
(forall (?r ?a1) (iff (wearable ?r ?a1)
(forall (?a2 ?occ1 ?occ2)
    (implies  (and  (common ?a1 ?a2 ?r)
            (subactivity ?a1 ?a)
            (subactivity ?a2 ?a)
            (occurrence_of ?occ2 ?a1))
        (forall (?b)
```

```
(implies  (exists (?occ3)
                  (and    (subactivity_occurrence ?occ3 ?b)
                          (occurrence_of ?b ?a)
                          (precedes ?occ2 ?occ3))
                          (not (poss ?a2 ?occ3)))))))
```

7 Capacity-based concurrency

This clause characterizes all definitions pertaining to Capacity-based concurrency.

7.1 Primitive lexicon of Capacity-based concurrency

No primitive relations are introduced by the lexicon of Capacity-based concurrency.

7.2 Defined lexicon for concepts of Capacity-based concurrency

The following relations are defined in this clause:

— (exclusive_use ?a ?r);

— (capacity_based ?a ?r);

— (unary_resource ?r);

— (capacitated ?r);

— (uniform_demand ?r ?q);

— (layput ?r ?a).

Each concept is described by informal semantics and a KIF axiom.

7.3 Theories required by Capacity-based Concurrency

This theory requires

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

7.4 Definitional extensions required by Capacity-based concurrency

No Definitional Extensions are required by Capacity-based concurrency.

7.5 Definitions of Capacity-based concurrency

The following concepts are defined for Capacity-based concurrency.

7.5.1 exclusive_use

A resource is exclusive use for an activity if and only if its demand for the resource is equal to its resource point.

```
(forall (?a ?r) (iff (exclusive_use ?a ?r)
(forall (?q1 ?q2 ?occ ?occp)
    (implies  (and  (do ?a ?occ ?occp)
                    (holds (demand ?a ?r ?q1) ?occ)
                    (holds (resource_point ?r ?q2) ?occ))
          (= ?q1 ?q2))))))
```

NOTE: If a resource is exclusive use with respect to two activities, then the two activities cannot be concurrent.

7.5.2 capacity_based

A resource is capacity-based for an activity if and only if its demand for the resource is less than its resource point.

```
(forall (?a ?r) (iff (capacity_based ?a ?r)
(forall (?q1 ?q2 ?occ ?occp)
    (implies    (and    (Do ?a ?occ ?occp)
                    (holds (demand ?a ?r ?q1) ?occ)
                    (holds (resource_point ?r ?q2) ?occ))
          (lesser ?q1 ?q2))))))
```

NOTE: Capacity-based resources allow the possibility that multiple activities can share the resource.

7.5.3 unary_resource

A resource is unary if and only if all activities that require it are exclusive use.

```
(forall (?r) (iff (unary_resource ?r)
(forall (?a)
    (implies  (res_requires ?a ?r)
          (exclusive_use ?a ?r))))))
```

7.5.4 capacitated_resource

A resource is capacitated if and only if all activities that require it are capacity-based.

```
(forall (?r) (iff (capacitated_resource ?r)
```

(forall (?a)

(implies (res_requires ?a ?r)

(capacity_based ?a ?r))))))

7.5.5 uniform_demand

An activity has uniform demand for a resource if and only if the demand for the resource is the same in all situations.

(forall (?a ?r ?q) (iff (uniform_demand ?a ?r ?q)

(forall (?occ)

(holds (demand ?a ?r ?q) ?occ)))))

7.5.6 layout

An activity is a layout activity for a resource if and only if the demand varies depending on the effects of other activities.

(forall (iff (layout ?r ?a)

(forall (?q ?occ1 ?occ2)

(not (iff (holds (demand ?r ?a ?q) ?occ2)

(holds (demand ?r ?a ?q) ?occ1)))))))

EXAMPLE: Two baking activities can be concurrent if the cakes are placed along the edges of the oven, but cannot concurrent if the cakes must be placed in the centre of the oven.

8 Resource sharability

This clause characterizes all definitions pertaining to Resource sharability.

It is important to realize that the notion of quantity referred to in this extension is independent of the notion of resource existence or identity, as well as the distinction between discrete and continuous resources. The only concern is the availability of the resource for a future activity. The notion of resource point is a constraint on such availability. The quantity specifies the sharability of the resource with respect to multiple concurrent activities. The resource point determines the maximal set of concurrent activities that require the resource. Thus, if the quantity is zero, then no activity that requires the resource is possible.

8.1 Primitive lexicon of Resource sharability

No primitive relations are introduced by the lexicon of Resource sharability.

8.2 Defined lexicon of Resource sharability

The following relations are defined in this clause:

— (consumes_quantity ?s1 ?s2 ?a);

— (strict_consumes_quantity ?s1 ?s2 ?a);

— (produces_quantity ?occ1 ?occ2);

— (strict_produces_quantity ?occ);

— (uses_quantity ?occ);

— (creates ?occ);

— (destroys ?occ);

— (fixed_quantity ?r);

— (nonreplenishable ?r);

— (uses ?a ?r);

— (consumes ?a ?r);

— (strict_consumes ?a ?r);

— (produces ?a ?r);

— (strict_produces ?a ?r);

— (provides ?a ?r);

— (provides_quantity ?a ?r).

Each concept is described by informal semantics and a KIF axiom.

8.3 Theories required by Resource sharability

This theory requires the following extensions:

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

8.4 Definitional extensions required by Resource sharability

No Definitional Extensions are required by Resource sharability.

8.5 Definitions of Resource sharability

The following concepts are defined for Resource sharability.

8.5.1 consumes_quantity

An activity consumes some quantity ?q of a resource if and only if the demand for the resource is ?q and the resource point of the resource is decremented by ?q after the occurrence of the activity.

```
(forall (?a ?r ?q) (iff (consumes_quantity ?a ?r ?q)
(forall (?q1 ?occ ?occ1 ?occ2)
    (implies  (and  (do ?a ?occ1 ?occ2)
                    (holds (demand ?a ?r ?q) ?occ1)
                    (holds (resource_point ?r ?q1) ?occ1))
              (holds (resource_point ?r (- ?q1 ?q)) ?occ2)))))
```

8.5.2 strict_consumes_quantity

An activity strictly consumes some quantity of a resource if and only if the resource is nonreplenishable.

```
(forall (?a ?r ?q) (iff (strict_consumes_quantity ?a ?r ?q)
(and    (consumes_quantity ?a ?r ?q)
    (nonreplenishable ?r))))))
```

8.5.3 produces_quantity

An activity produces some quantity ?q of a resource if and only if the resource point of the resource is incremented by ?q after the occurrence of the activity.

```
(forall (?a ?r ?q) (iff (produces_quantity ?a ?r ?q)
(forall (?q1 ?q2 ?occ ?occ1 ?occ2)
    (implies  (and  (do ?a ?occ1 ?occ2)
                    (holds (resource_point ?r ?q1) ?occ1)
                    (= ?q2 (plus ?q1 ?q)))
              (holds (resource_point ?r ?q2) ?occ2))))))
```

8.5.4 strict_produces_quantity

An activity strictly produces some quantity of a resource if and only if there is no other activity that consumes some quantity of it.

```
(forall (?a ?r ?q) (iff (strict_produces_quantity ?a ?r ?q)
(exists (?q)
    (and    (produces_quantity ?a ?r ?q)
        (not (exists (?a2 ?q2)
                (and    (subactivity ?a2 ?a)
                    (consumes_quantity ?a2 ?r ?q2)))))))))
```

8.5.5 uses_quantity

An activity uses some quantity ?q of a resource if and only if the demand for the resource is ?q and the resource point of the resource is unchanged by the occurrence of the activity.

EXAMPLE: A construction activity that requires five workers, since the number of workers is not changed by the activity.

```
(forall (?a ?r ?q) (iff (uses_quantity ?a ?r ?q)
(forall (?q1 ?q2 ?q3 ?occ1 ?occ2)
    (implies  (and  (do ?a ?occ1 ?occ2)
                (holds (demand ?a ?r ?q) ?occ1))
                (holds (resource_point ?r ?q1) ?occ1)
                (holds (resource_point ?r ?q2) ?occ2)
            (= ?q2 ?q1)))
```

8.5.6 creates

An activity creates a resource if and only if it produces some quantity of the resource, and the quantity of the resource before the occurrence of the activity was zero_quantity.

```
(forall (?a ?r) (iff (creates ?a ?r)
(exists (?q1)
    (and    (produces_quantity ?a ?r ?q1)
        (forall (?q2 ?occ)
            (implies  (and  (occurrence_of ?occ ?a)
                        (prior (resource_point ?r ?q2) ?occ)
                    (= ?q2 zero_quantity)))))))
```

8.5.7 destroys

An activity destroys a resource if and only if it consumes some quantity of the resource, and the quantity of the resource after the occurrence of the activity is zero_quantity.

```
(forall (?a ?r) (iff (destroys ?a ?r)
(exists (?q1)
    (and    (consumes_quantity ?a ?r ?q1)
        (forall (?q2 ?occ)
            (implies  (and  (occurrence ?occ ?a)
                        (prior (resource_point ?r ?q2) ?occp)
                    (= ?q2 zero_quantity)))))))
```

8.5.8 fixed_quantity

A resource has fixed quantity ?q if and only if the resource point of the resource is the same in all situations, that is, it does not change.

```
(forall (?r ?q) (iff (fixed_quantity ?r ?q)
```

```
(forall (?occ)
    (holds (resource_point ?r ?q) ?occ)))))
```

8.5.9 nonreplenishable

A resource is nonreplenishable if and only if the resource point for the resource cannot be increased after the occurrence of activities that require the resource.

```
 (forall (?r) (iff (nonreplenishable ?r)
(forall (?a ?q1 ?q2 ?occ1 ?occ2 ?occ3)
    (implies  (and  (implies     (do ?a ?occ1 ?occ2)
                                 (holds (resource_point ?r ?q1) ?occ2))
                    (precedes ?occ2 ?occ3)
                    (holds (resource_point ?r ?q2) ?occ3)))
                   (or   (greater ?q1 ?q2)
                         (= ?q1 ?q2)))))
```

8.5.10 uses

An activity uses a resource if and only if it uses some quantity of the resource.

```
 (forall (?a ?r) (iff (uses ?a ?r)
(exists (?q)
    (uses_quantity ?a ?r ?q)))))
```

8.5.11 consumes

An activity consumes a resource if and only if it consumes some quantity of the resource.

```
 (forall (?a ?r) (iff (consumes ?a ?r)
(exists (?q)
    (consumes_quantity ?a ?r ?q)))))
```

8.5.12 strict_consumes

An activity strictly consumes a resource if and only if it strictly consumes some quantity of the resource.

```
 (forall (?a ?r) (iff (strict_consumes ?a ?r)
(exists (?q)
    (strict_consumes_quantity ?a ?r ?q))))
```

8.5.13 produces

An activity produces a resource if and only if it produces some quantity of the resource.

```
 (forall (?a ?r) (iff (produces ?a ?r)
(exists (?q)
    (produces_quantity ?a ?r ?q))))
```

8.5.14 strict_produces

An activity strictly produces a resource if and only if it strictly produces some quantity of the resource.

```
(forall (?a ?r) (iff (strict_produces ?a ?r)
(exists (?q)
    (strict_produces_quantity ?a ?r ?q))))
```

8.5.15 provides_quantity

An activity provides some quantity of a resource if and only if there exists a subactivity that produces some quantity of the resource and another subactivity that consumes some quantity of the resource.

```
(forall (?a ?r ?q) (iff (provides_quantity ?a ?r ?q)
(and  (exists (?a1)
        (and  (subactivity ?a1 ?a)
              (produces_quantity ?a1 ?r ?q)))
    (exists (?a2)
        (and  (subactivity ?a2 ?a)
              (consumes_quantity ?a2 ?r ?q))))))
```

8.5.16 provides

An activity provides a resource if and only if it provides some quantity of a resource.

```
(forall (?a ?r) (iff (provides ?a ?r)
(exists (?q)
    (provides_quantity ?a ?r ?q))))
```

9 Resource set-based activities

This clause characterizes all definitions pertaining to Resource set-based activities.

9.1 Primitive lexicon of Resource set-based activities

No primitive relations are required by the lexicon of Resource set-based activities.

9.2 Defined lexicon of Resource set-based activities

The following relations are defined in this clause:

— (nondet_select ?a)

— (nondet_set_select ?a)

— (nondet_quantity_select ?a)

— (res_requires_set ?a)

— (res_requires_full_set ?a)

— (nondet_res_activity ?a)

Each concept is described by informal semantics and a KIF axiom.

9.3 Theories required by Resource Set-based Activities

This theory requires

— res_set.th;

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

9.4 Definitional extensions required by Resource set-based activities

This extension requires the following definitional extension:

— strong_poset.def.

9.5 Definitions of Resource set-based activities

The following concepts are defined for Resource set-based activities.

9.5.1 nondet_select

An activity is a nondeterministic selection activity with respect to a resource set ?r1 if and only if the occurrence of the activity is equivalent to the occurrence of a subactivity which requires a resource that is an element of the set associated with ?r1.

```
(forall (?a1 ?r1) (iff (nondet_select ?a1 ?r1)
(forall (?occ)
    (iff    (occurrence_of ?occ ?a1)
        (exists (?r2 ?i ?a2 ?occ2)
            (and    (subactivity ?a2 ?a1)
                (holds (resource_set ?i ?r1) (root_occ ?occ))
```

(holds (in ?r2 ?i) (root_occ ?occ))

(res_requires ?a2 ?r2)

(occurrence_of ?occ2 ?a2)

(subactivity_occurrence ?occ2 ?occ)))))))

9.5.2 nondet_set_select

An activity is a nondeterministic set selection activity with respect to a resource set ?r1 if and only if the occurrence of the activity is equivalent to the occurrence of a subactivity which requires a subset of resources that are elements of the set associated with ?r1.

(forall (?a1 ?r1) (iff (nondet_set_select ?a1 ?r1)

(forall (?occ)

(iff (occurrence_of ?occ ?a1)

(exists (?r2 ?a2)

(and (subactivity ?a2 ?a1)

(holds (resource_subset ?r2 ?r1) (root_occ ?occ))

(res_requires ?a2 ?r2)

(occurrence_of ?occ2 ?a2)

(subactivity_occurrence ?occ2 ?occ)))))))

9.5.3 nondet_quantity_select

An activity is a nondeterministic quantity selection activity with respect to a resource set ?r if and only if it is a nondeterministic set selection activity, and the cardinality of the selected subset is equal to ?q.

(forall (?a ?r ?q) (iff (nondet_quantity_select ?a ?r ?q)

(and (nondet_set_select ?a ?r)

(= ?q (cardinality ?r)))))

9.5.4 requires_set

An activity requires the resource set ?r if and only if every subactivity requires some resource which is an element of the set associated with ?r.

(forall (?a ?r) (iff (res_requires_set ?a ?r)

(forall (?occ1)

(iff (occurrence_of ?occ1 ?a)

(forall (?a1 ?i)

(implies (and (holds (resource_set ?i ?r) (root_occ ?occ))

(subactivity ?a1 ?a))

(exists (?r1 ?occ2 ?s2)

(and (occurrence_of ?occ2 ?a1)

(holds (in ?r1 ?i) (root_occ ?occ2))

(res_requires ?a1 ?r1)

```
(subactivity_occurrence ?occ1 ?occ1)))))))))
```

9.5.5 requires_full_set

An activity requires the full resource set ?r if and only if every resource which is an element of the set associated with ?r is required by some subactivity .

```
(forall (?a ?r) (iff (res_requires_full_set ?a ?r)
(forall (?occ1)
    (iff    (occurrence_of ?occ1 ?a)
        (forall (?r1)
            (implies  (holds (in_resource_set ?r1 ?r) (root_occ ?occ1))
                (exists (?a1 ?occ2)
                    (and    (subactivity ?a1 ?a)
                        (res_requires ?a1 ?r1)
                        (occurrence_of ?occ2 ?a1)
                        (subactivity_occurrence ?occ2 ?occ1)))))))))
```

9.5.6 nondet_res_activity

An activity is a nondeterministic resource activity if and only if the reason that the activity is nondeterministic is because it is a nondeterministic selection activity with respect to some resource set.

```
(forall (?a) (iff (nondet_res_activity ?a)
(implies  (choice_poset ?a)
    (exists (?r1)
        (nondet_select ?a ?r1)))))
```

10 Substitutable resources

This clause characterizes all definitions pertaining to Substitutable resources.

10.1 Primitive lexicon of Substitutable resources

No primitive relations are introduced by the lexicon of Substitutable resources.

10.2 Defined lexicon of Substitutable resources

The following relations are defined in this clause:

— (superpose_select ?a);

— (homogeneous_set ?a);

— (set_contention ?a).

Each concept is described by informal semantics and a KIF axiom.

10.3 Theories required by Substitutable resources

This theory requires

— res_set.th;

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

10.4 Definitional extensions required by Substitutable resources

This Extension requires set_action.def.

10.5 Definitions of Substitutable resources

The following concepts are defined for Substitutable resources.

10.5.1 superpose_select

An activity ?a is a superpose_select activity with respect to a resource set ?r if and only if every subactivity nondeterministically selects a subset of resources from ?r.

EXAMPLE: An activity that requires a worker and a lathe machine may select the lathe machine from a pool of machines. The activities that select each lathe form a superpose_select activity.

```
(forall (?a ?r) (iff (superpose_select ?a ?r)
(forall (?a1 ?occ1)
    (implies  (and  (occurrence_of ?occ1 ?a)
                    (subactivity ?a1 ?a)
                    (primitive ?a1))
              (exists (?a2 ?r1 ?occ2)
                  (and    (subactivity ?a1 ?a2)
                          (subactivity ?a2 ?a)
                          (occurrence_of ?occ2 ?a2)
                          (holds (resource_subset ?r1 ?r) (root_occ ?occ2))
```

(nondet_select ?a2 ?r1)))))))

10.5.2 homogeneous_set

A resource set is homogeneous if and only if it is required by a superpose_select activity.

EXAMPLE: For an activity that requires one worker and one lathe machine, a set of three workers form a homogeneous set and a set of four lathe machines form a homogeneous set, but a worker and a lathe machine together do not form a homogeneous set.

(forall (?a ?r) (iff (homogeneous_set ?r ?a)

(exists (?a2)

(and (superpose_select ?a2 ?r)

(subactivity ?a ?a2)))))

10.5.3 set_contention

An activity is a set_contention activity with respect to a resource ?r if and only if the preconditions for the activity require that ?r is a homogeneous set and ?r is available.

(forall (?r ?s)

(implies (poss (set_contention ?r) ?s)

(and (forall (?a)

(implies (subactivity ?a (set_contention ?a))

(prior (homogeneous_set ?r ?a) ?s)))

(prior (available ?r (set_contention ?r)) ?s)))))

11 Homogeneous sets

This clause characterizes all definitions pertaining to Homogeneous sets.

11.1 Primitive lexicon of Homogeneous sets

No primitive relations are required by the lexicon of Homogeneous sets.

11.2 Defined relations of Homogeneous ets

The following relations are defined in this clause:

— (pile ?a);

— (stock ?a);

— (pool ?a);

— (pool_demand ?a);

— (uses_pile ?a ?r);

— (consumes_pile ?a ?r);

— (produces_pile ?a ?r).

Each concept is described by informal semantics and a KIF axiom.

11.3 Core Theories required by Homogeneous sets

This theory requires the following core theories:

— res_set.th;

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

11.4 Definitional extensions required by Homogeneous sets

This extension requires the following definitional extensions: subst_res.def,res_set_action.def, res_divisible.def.

11.5 Definitions of Homogeneous Sets

The following concepts are defined for Homogeneous sets.

11.5.1 pile

A resource set is a pile with respect to an activity if and only if it is a homogeneous set with respect to the activity, and the resource point is equal to the cardinality of the set.

```
 (forall (?a ?r) (iff (pile ?r ?a)
(and    (homogeneous_set ?r ?a)
       (forall (?q ?occ)
             (iff    (prior (resource_point ?r ?q) ?occ)
                   (= ?q (cardinality ?i)))))
```

11.5.2 stock

A resource set is a stock with respect to an activity if and only if it is a homogeneous set with respect to the activity, there exists a resource set whose cardinality is equal to the demand, and the aggregated demand is equal to the cardinality of the set.

```
(forall (?r ?a) (iff (stock ?r ?a)
(and    (homogeneous_set ?r ?a)
      (forall (?q ?occ)
            (implies  (prior (demand ?a ?r ?q) ?occ)
                  (exists (?i1 ?r1)
                        (and    (= ?q (cardinality ?i1))
                              (prior (resource_set ?i1 ?r1) ?occ)
                              (res_requires_set ?a ?r1)))))
                (forall (?q3)
                        (implies  (prior (agg_demand ?r ?q3) ?occ)
                              (= ?q3 (cardinality ?i)))))
```

11.5.3 pool

A resource set is a pool with respect to an activity if and only if it is a homogeneous set with respect to the activity, and the demand for the pool is equal to the cardinality of a subset.

```
 (forall (?r ?a) (iff (pool ?r ?a)
(and    (homogeneous_set ?r ?a)
      (forall (?q ?occ)
            (implies  (prior (demand ?a ?r ?q) ?occ)
                  (exists (?i1 ?r1)
                        (and    (subset ?i1 ?i)
                              (= ?q (cardinality ?i1))
                              (prior (resource_set ?i1 ?r1) ?occ)
                              (res_requires_set ?a ?r1))))))
```

11.5.4 pool_demand

A pool is used by some activity if and only if the activity uses some quantity of the resource pool (remember that resource pools are themselves resources).

```
 (forall (?a ?r ?q) (iff (pool_demand ?a ?r ?q)
(and    (pool ?r ?a)
      (forall (?q1 ?occ)
            (implies  (holds (demand ?a ?r ?q1) ?occ)
                  (= ?q ?q1))))
```

11.5.5 uses_pile

A pile is used by some activity if and only if the activity uses some quantity of the resource pool (remember that resource pools are themselves resources).

```
 (forall (?a ?r ?q) (if (uses_pile ?a ?r ?q)
(and    (pile ?r ?a)
```

(uses_quantity ?a ?r ?q))))

11.5.6 consumes_pile

A pile is consumed with respect to some activity if and only if the activity consumes quantity of resources in the resource pool (remember that resource pools are themselves resources).

(forall (?a ?r ?q) (iff (consumes_pile ?a ?r ?q)

(and (pile ?r ?a)

(consumes_quantity ?a ?r ?q)))))

11.5.7 produces_pile

A pile is produced with respect to some activity if and only if the activity produces some quantity of resources in the resource pool (remember that resource pools are themselves resources).

(forall (?a ?r ?q) (iff (produces_pile ?a ?r ?q)

(and (pile ?r ?a)

(produces_quantity ?a ?r ?q)))))

12 Resource pools

This clause characterizes all definitions pertaining to Resource pools.

12.1 Primitive lexicon of Resource pools

No primitive relations are introduced by the lexicon of Resource pools.

12.2 Defined lexicon of Resource pools

The following relations are defined in this clause:

— (resource_pool ?a);

— (conservative_pool ?a);

— (material_pool ?a).

Each concept is described by informal semantics and a KIF axiom.

12.3 Theories required by Resource pools

This theory requires

— res_set.th;

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

12.4 Definitional extensions required by Resource pools

The following definitional extensions are required by Resource pools:

— homogeneous_setdef;

— subst_res.def;

— res_set_action.def.

12.5 Definitions of Resource pools

The following concepts are defined for Resource pools.

12.5.1 resource_pool

A resource pool is a resource set which is homogeneous with respect to some activity ?a, and whose capacity constraints satisfy the following conditions:

The resource point of the resource pool is equal to the cardinality of the associated set.

The demand for the resource pool by an activity is equal to the cardinality of the subset of resources required by the activity.

The minimum capacity of the resource pool is equivalent to the minimum cardinality of the associated set.

EXAMPLE: In typical manufacturing activities, resource pools are sets of reusable resources, such as a set of lathe machines or set of screwdrivers that may be selected to perform the activity.

```
(forall (?r ?a) (iff (resource_pool ?r ?a)
(forall (?i ?occ)
    (implies  (holds (resource_set ?i ?r) ?occ)
        (and  (pile ?r ?a)
            (pool ?r ?a)
            (forall (?q3)
                (implies  (holds (min_capacity ?a ?r ?q3) ?occ)
                    (lesser ?q3 (cardinality ?i)))))))))
```

12.5.2 conservative_pool

A resource pool is conservative with respect to an activity if and only if the demand for the pool is equal to the quantity which is used or consumed.

```
(forall (?a ?r) (iff (conservative_pool ?a ?r)
(and    (resource_pool ?r ?a)
        (forall (?q)
                (iff    (pool_demand ?a ?r ?q)
                        (or     (uses_pile ?a ?r ?q)
                                (consumes_pile ?a ?r ?q)))))))
```

12.5.3 material_pool

A material pool is a resource pool with respect to some activity if and only if the resource pool provides a quantity with respect to the activity.

```
(forall (?r ?a) (iff (material_pool ?r ?a)
(and    (resource_pool ?r ?a)
        (exists (?q)
                (provides_quantity ?a ?r ?q))))
```

13 Inventory resource sets

This clause characterizes all definitions pertaining to Inventory resource sets.

13.1 Primitive lexicon of Inventory resource sets

No primitive relations are introduced by the lexicon of Inventory resource sets.

13.2 Defined lexicon of Inventory resource sets

The following relations are defined in this clause:

— (inventory_resource ?s1 ?s2 ?a);

— (inventory_pool ?s1 ?s2 ?a);

— (inventory_contains ?s1 ?s2 ?a).

Each concept is described by informal semantics and a KIF axiom.

13.3 Theories required by Inventory resource sets

This theory requires

— res_set.th;

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

13.4 Definitional extensions required by Inventory resource sets

The following definitional extensions are required by Inventory Resource Sets:

— homogeneous_set.def;

— subst_res.def;

— res_set_action.def;

— processor.def.

13.5 Definitions of Inventory resource sets

The following concepts are defined for Inventory resource sets.

13.5.1 inventory_resource

Inventory is either input material or output material for some activity.

```
(forall (?r) (iff (inventory_resource ?r)
(exists (?a)
    (or    (input_material ?r ?a)
           (output_material ?r ?a)))))
```

13.5.2 inventory_pool

An inventory pool is a resource set which is homogeneous with respect to some activity ?a, and whose capacity constraints satisfy the following conditions:

The resource point of the inventory pool is equal to the maximum cardinality of the associated set.

The aggregate demand of the resource pool before an activity occurrence is equal to the cardinality of the set associated with the resource in that state.

The demand for the resource pool by an activity is equal to the cardinality of a set of resources required by the activity.

The minimum capacity of the resource pool is equivalent to the minimum cardinality of the associated set.

EXAMPLE: In typical manufacturing activities, inventory pools are buffers and other sets of input or output materials.

```
(forall (?r ?a) (iff (inventory_pool ?r ?a)
(forall (?i ?occ)
    (implies  (holds (resource_set ?i ?r) ?occ)
        (and  (inventory_resource ?r)
            (homogeneous_set ?r ?a)
            (forall (?q1)
                (implies  (holds (resource_point ?r ?q1) ?occ)
                    (greaterEq ?q1 (cardinality ?i))))
            (stock ?r ?a)
            (forall (?a ?q4)
                (implies  (holds (min_capacity ?a ?r ?q4) ?occ)
                    (lesserEq ?q4 (cardinality ?i)))))))))))
```

13.5.3 inventory_contains

A resource is contained in an inventory pool iff it is a member of the resource set associated with the inventory pool.

```
(forall (?r1 ?r2 ?occ)
    (iff    (state (inventory_contains ?r1 ?r2) ?occ)
        (exists (?a ?i)
            (and    (inventory_pool ?r2 ?a)
                (holds (resource_set ?i ?r2) ?occ)
                (holds (in ?r1 ?i) ?occ))))))
```

14 Processor activities

This clause characterizes all definitions pertaining to Processor activities.

14.1 Primitive lexicon of Processor activities

No primitive relations are introduced by the lexicon of Processor activities.

14.2 Defined lexicon of Processor activities

The following relations are defined in this clause:

— (processor_activity ?occ);

— (processor_resource ?r ?a);

— (input_material ?r ?a);

— (output_material ?r ?a).

Each concept is described by informal semantics and a KIF axiom.

14.3 Theories required by Processor activities

This theory requires

— additive.th;

— requires.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

14.4 Definitional extensions required by Processor activities

This extension requires res_role.def.

14.5 Definitions of Processor activities

The following concepts are defined for Processor activities, which form a subclass of activities that are defined with respect to resource roles.

14.5.1 processor_activity

A processor activity is a class of activity which uses some set of resources, consumes some other set of resources, and produces a set of objects.

EXAMPLE: Processor activities are typically manufacturing processes, particularly as considered to be the subactivities of process plans and supply chain models.

```
(forall (?a) (iff (processor_activity ?a)
(exists (?r1 ?r2 ?r3)
      (and    (or     (reusable ?r1 ?a)
                      (possibly_reusable ?r1 ?a))
              (or     (consumable ?r2 ?a)
                      (possibly_consumable ?r2 ?a)
               (or     (consumable ?r3 ?a)
                      (possibly_consumable ?r3 ?a)
(creates ?a ?r3)))))
```

14.5.2 processor_resource

An object ?r is a processor resource for an activity ?a iff ?a is a processor activity which uses ?r.

EXAMPLE: In a typical manufacturing activity, a processor resource is a machine or a tool.

```
(forall (?r ?a) (iff (processor_resource ?r ?a)
(and    (processor_activity ?a)
        (reusable ?r1 ?a)
        (possibly_reusable ?r1 ?a))))
```

14.5.3 input_material

An object ?r is an input material for an activity ?a if and only if ?a is a processor activity which consumes or possibly consumes ?r.

EXAMPLE: input_material resources are defined strictly with respect to resource roles. In a typical manufacturing activity, the resources that are consumed (such as raw materials) are considered to be input materials.

```
(forall (?r ?a) (iff (input_material ?r ?a)
(and    (processor_activity ?a)
        (or     (consumable ?r ?a)
                (possibly_consumable ?r ?a))))))
```

14.5.4 output_material

An object ?r is an output material for an activity ?a iff ?a is a processor activity which produces or consumes or possibly consumes ?r.

EXAMPLE: In a typical manufacturing activity, output material is either something that is created (such as an printed circuit board) or is possibly consumable (such as an assembly onto which additional parts have been attached).

```
(forall (?r ?a) (iff (output_material ?r ?a)
(and    (processor_activity ?a)
        (or     (creates ?a ?r)
 (consumable ?r ?a)
                (possibly_consumable ?r ?a))))))
```

15 Resource paths

This clause characterizes all definitions pertaining to Resource paths.

15.1 Primitive lexicon of Resource paths

No primitive relations are introduced by the lexicon of Resource paths.

15.2 Defined lexicon of Resource paths

The following relations are defined in this clause:

— (next_processor_path ?a ?s);

— (pro_precedes ?a ?s);

— (resource_path ?a ?s)

— (initial_resource_path ?a ?s);

— (final_resource_path ?a ?s).

Each concept is described by informal semantics and a KIF axiom.

15.3 Theories required by Resource paths

This theory requires

— additive.th;

— requires.th;

— soo.th;

— act_occ.th;

— complex.th;

— subactivity.th;

— occtree.th;

— disc_state.th;

— psl_core.th.

15.4 Definitional extensions required by Resource paths

The following definitional extensions are required by Resource paths:

— processor.def;

— res_role.def.

15.5 Definitions of Resource paths

The following concepts are defined for Resource paths.

15.5.1 next_processor_path

An activity occurrence ?occ2 is the next processor subactivity occurrence after ?occ1 in an activity ?a if and only if the output material of ?a1 is the input material of ?a2, and there is no other processor

subactivity of ?a which consumes the output material from ?a1, and which occurs between ?a1 and ?a2.

```
(forall (?occ1 ?occ2 ?a) (iff (next_processor_path ?occ1 ?occ2 ?a)
(and    (next_subactivity ?occ1 ?occ2 ?a)
        (exists (?a1 ?a2 ?r)
                (and    (occurrence_of ?occ1 ?a1)
                        (occurrence_of ?occ2 ?a2)
                        (processor_activity ?a1)
                        (processor_activity ?a2)
                        (output_material ?r ?a1)
                        (input_material ?r ?a2))))))
```

15.5.2 pro_precedes

pro_precedes is a partial ordering over the processor subactivity occurrences of ?a with respect to resource flow.

```
(forall (?occ1 ?occ2 ?a) (iff (pro_precedes ?occ1 ?occ2 ?a)
(and    (soo_precedes ?occ1 ?occ2 ?a)
        (forall (?occ3)
                (implies  (and  (soo_precedes ?occ1 ?occ3 ?a)
                                (soo_precedes ?occ3 ?occ2 ?a))
                        (exists (?occ4 ?occ5)
                                (and    (next_processor_path ?occ4 ?occ3 ?a)
                                        (next_processor_path ?occ3 ?occ5 ?a))))))))
```

15.5.3 resource_path

An activity is a resource path if and only if the subactivity occurrence ordering is equivalent to the flow ordering.

```
(forall (?a) (iff (resource_path ?a)
(forall (?occ1 ?occ2)
        (iff    (soo_precedes ?occ1 ?occ2 ?a)
                (pro_precedes ?occ1 ?occ2 ?a))))
```

EXAMPLE: Resource paths capture the notions of process plans, routing, and flows in supply chains.

15.5.4 initial_resource_path

An occurrence is initial processor activity occurrence in an activity ?a iff ?a is a resource path and the occurrence is the root of the subactivity occurrence ordering.

```
(forall (?occ ?a) (iff (initial_processor_path ?occ ?a)
(and    (resource_path ?a)
```

(root_soo ?occ ?a))))

15.5.5 final_resource_path

An occurrence is a final processor activity occurrence in an activity ?a iff ?a is a resource path and the occurrence is the leaf of the subactivity occurrence ordering.

(forall (?occ ?a) (iff (final_processor_path ?occ ?a)

(and (resource_path ?a)

(leaf_soo ?occ ?a))))

Annex A
(normative)
ASN.1 Identifier of ISO 18629-44

To provide for unambiguous identification of an information object in an open system, the object identifier

```
iso standard 18629 part 44 version 1
```

is assigned to this part of ISO 18629. The meaning of this value is defined in ISO/IEC 8824-1 and is described in ISO 18629-1.

Annex B
(informative)
Example of process description using ISO 18629-44

The purpose of this annex is to provide a detailed scenario in which the ISO 18629 PSL is used in a knowledge-sharing effort which involves multiple manufacturing functions.

This scenario is an "interoperability" manufacturing scenario. This means that its goal is to show how PSL can be used to facilitate the communication of process knowledge in a manufacturing environment. Specifically, this scenario is centred around the exchange of knowledge from a process planner to a job shop scheduler.

This annex extends the test case introduced in ISO 18629-11: 2005, Annex E to illustrate the application of definitional extensions concepts in the specification of the manufacturing process of a product named GT-350.

B.1 GT-350 Manufacturing Processes

This section unites the various departmental processes into a high-level collection of activities which are enacted to create a GT-350 product. As described in the GT-350 product structure (see ISO 18629-11: 2005, Annex D table D1), subcomponents of this product are either purchased, sub-contracted, or made internally. These process descriptions address the activities performed to manufacture the internal subcomponents. This top-down view of the manufacturing process provides an overall picture from an abstract, “make GT350” activity which is expanded down to the detailed departmental levels.

As the Figure B.1 below shows, the GT-350 manufacturing process is divided into 6 main areas of work. The first five: make interior, make drive, make trim, make engine and make chassis are all unordered with respect to each other but they must all be completed before final assembly takes place.

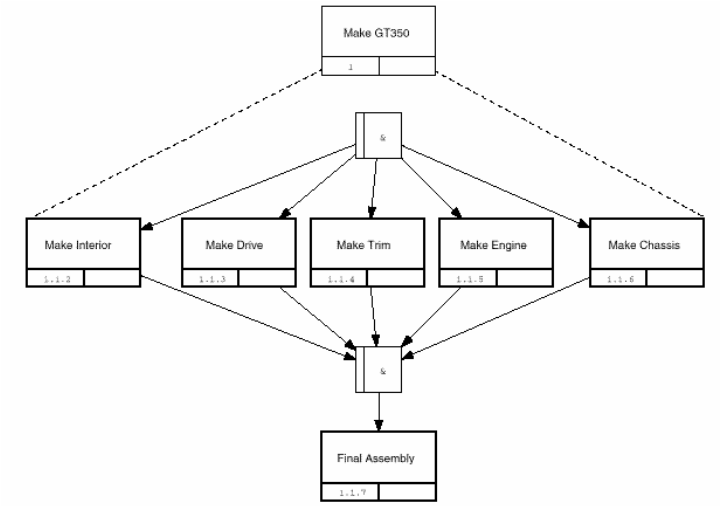

Figure B1: TOP level process for manufacturing a GT350 [4]

The PSL-Outercore-based representation of the top level process is :

```
(resource_path make_gt350)

(subactivity make-chassis make_gt350)
(subactivity make-interior make_gt350)
(subactivity make-drive make_gt350)
(subactivity make-trim make_gt350)
(subactivity make-engine make_gt350)
(subactivity final-assembly make_gt350)

(forall (?occ)
    (implies (occurrence_of ?occ make_gt350)
            (exists (?occ1 ?occ2 ?occ3 ?occ4 ?occ5 ?occ6)
                (and(occurrence_of ?occ1 make_chassis)
                                (occurrence_of ?occ2 make_interior)
                                (occurrence_of ?occ3 make_drive)
                                (occurrence_of ?occ4 make_trim)
                                (occurrence_of ?occ5 make_engine)
                                (occurrence_of ?occ6 final_assembly)
                                (subactivity_occurrence ?occ1 ?occ)
                                (subactivity_occurrence ?occ2 ?occ)
                                (subactivity_occurrence ?occ3 ?occ)
                                (subactivity_occurrence ?occ4 ?occ)
                                (subactivity_occurrence ?occ5 ?occ)
                                (subactivity_occurrence ?occ6 ?occ)
```

Each of these abstract activities can be further detailed, however for the example proposed in this annex, only some of them will be developed.

On the basis of the IDEF3 representation (in terms of process representation) of the abstract activities met during the different stages of the manufacturing process, some examples of process descriptions using the PSL-Outercore and definitional extensions ISO 18629 will be extracted.

B.2 The "make-engine" abstract activity

The 350-Engine is assembled from work performed in several CMW departments. The manufacturing process is shown in Figure B.2. The part is made up of an engine block, a harness, and wiring. The sub-processes are detailed in the sub-sections below. The 350-Engine is assembled at the A004 assembly bench and takes 5 minutes per piece.

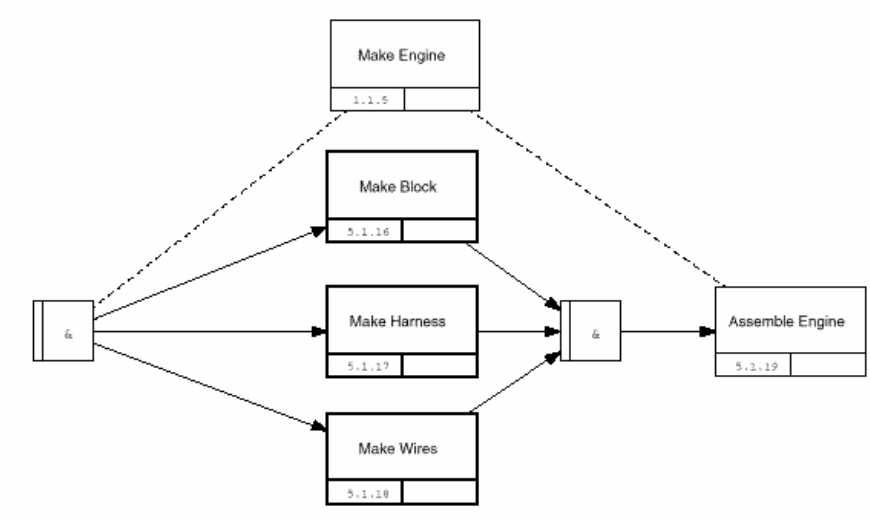

Figure B.2: PROCESS for manufacturing the 350–Engine [4]

The PSL-based representation of some activities and of the process related information at the make-engine stage is :

(subactivity make_block make_engine)

(subactivity make-harness make_engine)

(subactivity make-wires make_engine)

(subactivity assemble_engine make_engine)

(processor_activity make_engine)

(exists (?r1 ?r2 ?r3 ?r4)

(and (requires make_engine ?r1)

(requires make_engine ?r2)

(requires make_engine ?r3)

(requires make_engine ?r4)

(workcell ?r4)

(reusable ?r4 make_engine)

(processor_resource ?r4 make_engine)

(engine_block ?r1)

(input_material ?r1 make_engine)

```
        (uses_quantity make_engine ?r1 1)
        (output_material ?r1 make_engine)
        (possibly_consumable ?r1 make_engine)
        (harness ?r2)
        (input_material ?r2)
        (consumable ?r2 make_engine)
        (consumes_quantity make_engine ?r2 1)
        (wire ?r3)
        (input_material ?r3)
        (consumes_quantity make_engine ?r3 5)
        (wearable ?r3 make_engine)))
(forall (?r)
    (implies (engine_block ?r)
             (resource ?r)))

(forall (?r)
    (implies (harness ?r)
             (resource ?r))

(forall (?r ?s)
    (implies (wire ?r)
             (exists (?i)
                 (and (prior (resource_set ?i ?r) ?s)
                              (pile ?r make_engine)))))

(forall (?occ)
  (iff(occurrence_of ?occ make_engine)
    (exists (?occ1 ?occ2 ?occ3 ?occ4)
```

```
(and(occurrence_of ?occ1 make_block)
    (occurrence_of ?occ2 make_harness)
    (occurrence_of ?occ3 make_wires)
    (occurrence_of ?occ4 assemble_engine)
    (subactivity_occurrence ?occ1 ?occ)
    (subactivity_occurrence ?occ2 ?occ)
    (subactivity_occurrence ?occ3 ?occ)
    (subactivity_occurrence ?occ4 ?occ)
    (forall (?s1 ?s2 ?s3 ?s4)
        (implies  (and      (leaf_occ ?s1 ?occ1)
                    (leaf_occ ?s2 ?occ2)
                    (leaf_occ ?s3 ?occ3)
                                        (root_occ ?s4 ?occ4))
                (and     (min_precedes ?s1 ?s4 make_engine)
                                        (min_precedes ?s2 ?s4 make_engine)
                                        (min_precedes ?s3 ?s4 make_engine)))))
```

This representation formalizes the process depicted in Figure B.2.

The make_engine process requires four resources – an engine block, a harness, a set of wires, and a workcell. The workcell is a reusable resource. The engine_block is possibly_consumable because it can be reused by future activities that further modify the block, but it cannot be reused by all activities (since some changes have already been made). The harness is consumable, since no later activities can reuse the harness once it is installed. The set of wires is wearable, since it takes multiple occurrences of the make_engine process to deplete the wires in the resource set.

All of these constraints mean that make_engine is a processor activity in which the harness and the wire are input materials. The engine_block is both an input material and an output material, since it is modified during occurrences of make_engine.

The wires constitute a resource set; since any subset of wires can be consumed by make_engine, this set is a pile.

B.2.1 Make Block

The 350-Block is manufactured as part of the 350–Engine sub-assembly. This involves an integration of work from the foundry and machine shop, as shown in the Figure B.3.

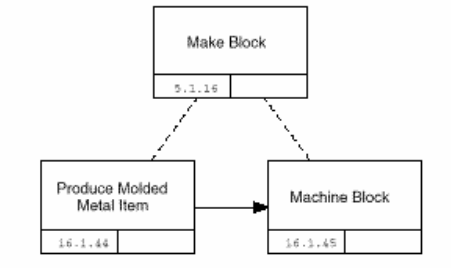

Figure B.3: PROCESS for manufacturing the 350–Block [4]

The PSL-based representation of some activities and of the process related information is :

(subactivity produce_molded_metal make_block)

(subactivity machine_block make_block)

(primitive machine_block)

(primitive produce_molded_metal)

(resource_path make_block)

(processor_activity produce_molded_metal)

(processor_activity machine_block)

(exists (?r1 ?r2 ?r3)

(and (requires produce_molded_metal ?r1)

(requires produce_molded_metal ?r2)

(cast ?r1)

(reusable ?r1produce_molded_metal)

(processor_resource ?r1 produce_molded_metal)

(metal_block ?r2)

(input_material ?r2 make_block)

(output_material ?r3 make_block)

(molded_block ?r3)

```
            (produces produce_molded_metal ?r3)
            (consumable ?r2 produce_molded_metal)
            (consumes_quantity produce_molded_metal ?r2 1)
            (unary_resource ?r1)
            (exclusive_use ?r1 produce_molded_metal)))


 (exists (?r1 ?r2)
          (and (requires machine_block ?r1)
            (requires machine_block ?r2)
            (reusable ?r2 machine_block)
            (metal_block ?r1)
            (input_material ?r1 machine_block)
            (uses_quantity machine_engine ?r1 1)
            (output_material ?r1 machine_block)
            (consumable ?r1 machine_engine)
            (milling_machine ?r2)
            (processor_resource ?r2 machine_block)
            (consumes_quantity machine_engine ?r1 1)
            (capacitated_resource ?r1 )))
(forall (?r)
     (implies (milling_machine ?r)
                (resource ?r)))


(forall (?r)
     (implies (molded_block ?r)
                (resource ?r))
```

```
(forall (?occ)
    (iff  (occurrence_of ?occ make_block)
        (exists (?occ1 ?occ2)
            (and(occurrence_of ?occ1 produce_molded_metal)
                (occurrence_of ?occ2 machine_block)
                (next_processor_path ?occ1 ?occ2 make_block)))))
```

This representation formalizes the process depicted in Figure B.3.

The make_block activity is a processor path activity composed of a sequence of two processor subactivities – produce_molded_metal and machine_block.

The subactivity produce_molded_metal consumes a metal block, uses a cast, and produces one molded block.

The subactivity machine_block consumes the molded block and uses a milling machine.

The cast can only be used by one activity at a time, so that it is exclusively used by the produce_molded_metal activity and hence is a unary resource.

The milling machine can be used by multiple concurrent activities, making it a capacitated resource.

B.2.2 Make Harness

The 350-Harness (Figure B.4) is manufactured as part of the 350–Engine sub-assembly. This involves work performed at the wire and cable department. Figure B.5 expands the harness wire production process. The 350-Harness is assembled by a bench worker at a wire and cable bench. It takes 10 minutes per set.

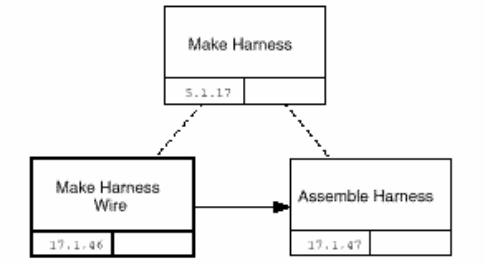

Figure B.4: PROCESS for manufacturing the 350–Harness [4]

The PSL-Outercore-based representation of some activities and of the process related information is :

```
(resource_path make_harness)
(pro_precedes make_harness_wire assemble_harness make_harness)

(subactivity make_harness_wire make_harness)
(subactivity assemble_harness make_harness)
(primitive assemble_harness)

(forall (?r)
    (implies (harness ?r)
             (wearable ?r make_harness)))

(forall (?occ)
    (implies (occurrence_of ?occ make_harness)
             (exists (?occ1 ?occ2 ?occ3 ?r)
                 (and(occurrence_of ?occ1 make_harness_wire)
                     (harness ?r)
                     (requires make_harness_wire ?r)
                     (occurrence_of ?occ2 assemble_harness)
                     (requires assemble_harness ?r)
                     (leaf_occ ?occ3 ?occ1)
                     (min_precedes ?occ3 ?occ2 make_harness)))))

(forall (?occ ?r ?q)
    (implies (and(occurrence_of ?occ assemble_harness)
                 (requires assemble_harness ?r)
                 (prior (resource_point ?r ?q) ?occ))
             (holds (resource_point ?r ?q) ?occ)))
```

Bibliography

[1] ISO 10303-1, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles.*

[2] ISO 10303-49, *Industrial automation systems and integration — Product data representation and exchange — Part 49 : Integrated generic resources : Process structure and properties*

[3] ISO 18629-13, *Industrial automation systems and integration — Process specification language — Part 13: Duration and ordering theories*

[4] *Federal Information Processing Standards Publication 184, Integration Definition for Information Modeling (IDEF3)*, FIPS PUB 184, National Institute of Standards and Technology, December 1993. IDEF3. Available from the Internet: <http://www.idef.com >.

Index

ICS 25.040.40

Price based on 48 pages