demetrix
Process modelling system
Vision document

Dimitri Pissarenko
dimitri.pissarenko@gmx.net

Purpose of the document

To explain the architecture of the emerging Demetrix Process Modelling System

Application domain

Management of any kind of data, which can be organized as a sequence of processes and resources flowing between process, for instance

- plant scheduling
- workflow management
- optimization of business processes

Overview over intended Demetrix architecture

This section gives an overview over most important principles, upon which the philosophy of Demetrix is based.

Graph based data representation

The data model of Demetrix is built around the concept of graph. The advantage of this is the availability of a working toolkit for managing graphs (Jgraph, http://www.jgraph.org).

Only three types of graph objects

A Demetrix graph consists of three types of objects:

- task vertices (task nodes)
- resource vertices (resource nodes)
- edges

Tasks transform resources. Edges show, which input resources are being transormed in a task into which output resource.

An example of a Demetrix graph is given in illustration 1.

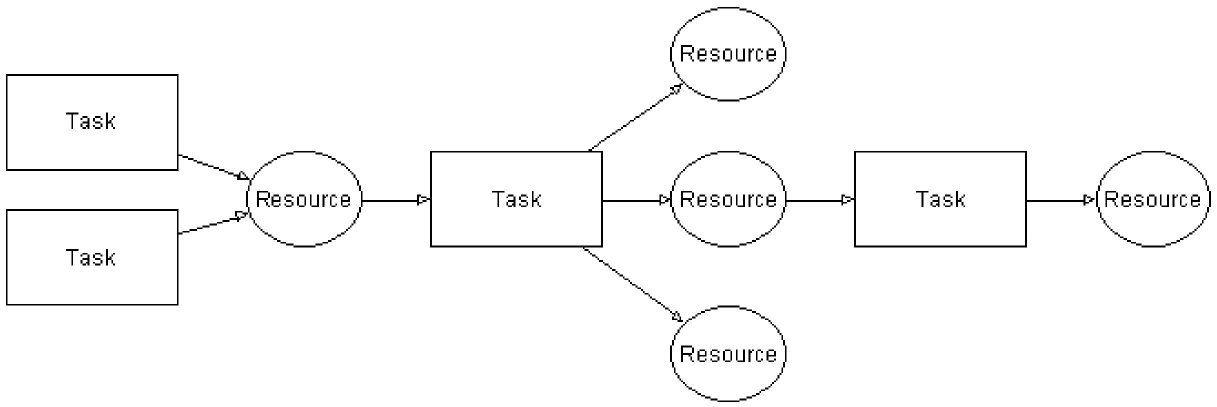


Illustration 1

Multi-level tasks

Each task node can have a subgraph, which has the same structure as shown in illustration 1. Most important advantage of this approach is the possibility of top-down modelling of process chains.

Imagine that the user knows that his/her process chain can be divided into coarse three parts. For instance, the user may know that the entire life cycle of an industrial product can be divided in *Production, Use and Disposal*.

The user first draws the top-level process chain, as shown in illustration 2.

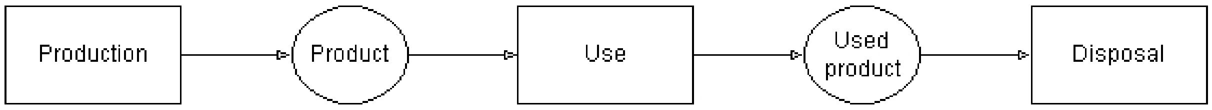


Illustration 2

In reality, all three tasks *Production, Use and Disposal* are process chains. As soon as the user gets enough data to refine the model, he/she refines the *Use* task. For this reason, the user „descends" into the *Use* process chain. At first, input resource *Product* is transformed into output resource *Used product* by magic, as shown in illustration 3.

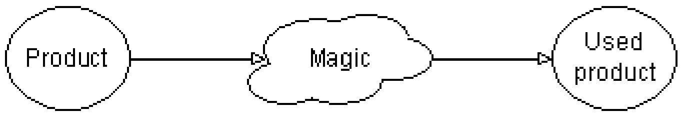


Illustration 3

The user knows, that in scope of the use of the product, sometimes the product is used without repairs and sometimes the product is repaired once. Further, the user knows that the product pollutes the environment differently if used particularly intensively.

The user draws the process chain shown in illustration 4 (this is only one of many possible ways to model the facts specified above).

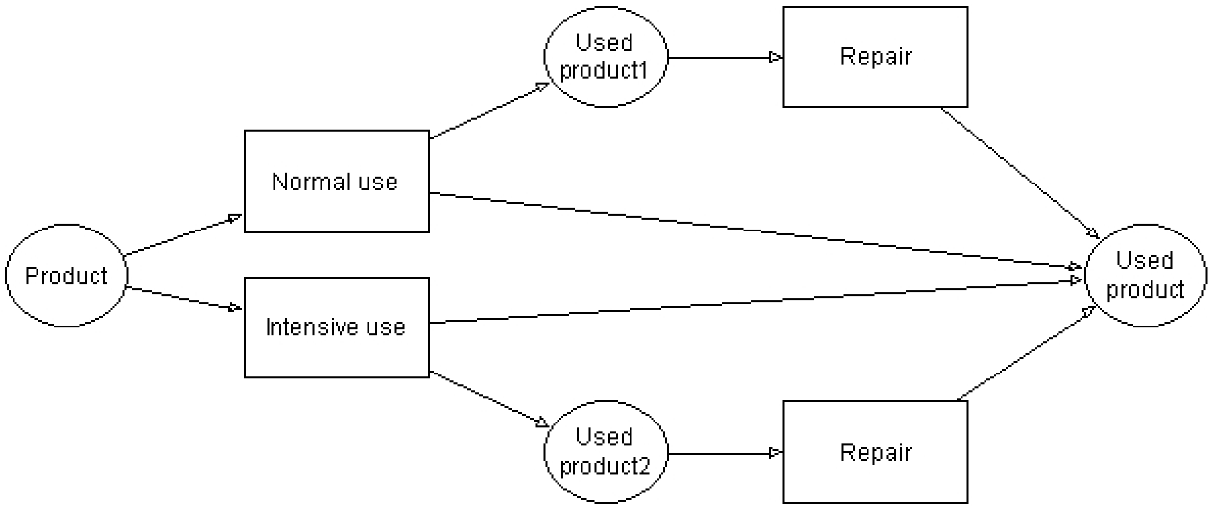


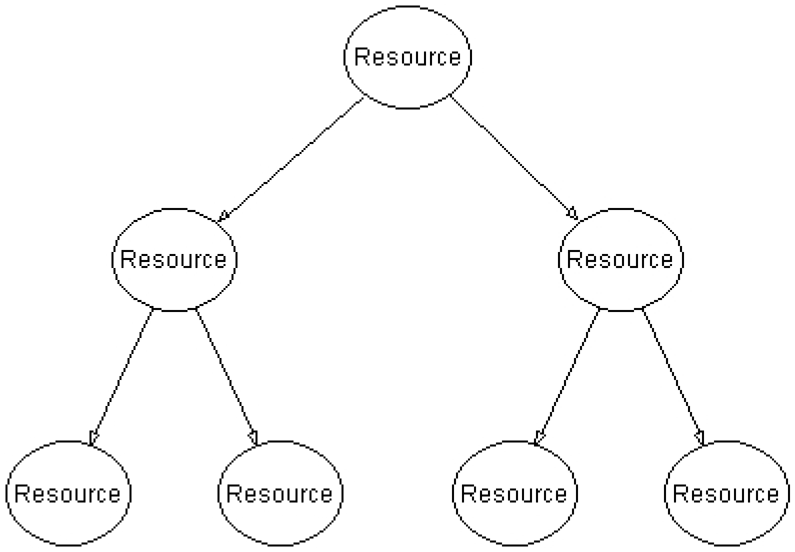
Illustration 4

If the user has additional data, he/she may also refine the *Normal use* and *Intensive Use* tasks in the analogous way.

In this way, the user is able to create process chains at arbitrary level of detail. Complex process chains can easily be subdivided into different levels, thus preserving clarity irrespective of the size of process chain.

Multi-level resources

Each resource node can also have a subgraph, which has the structure, shown in illustration 5.



Only resource nodes are allowed in the resource graph. What do the arrows mean? Well, this depends on the particular resource. There at least two possible interpretations of these edges:

- Composition relation (A consists of B)
- Attachment relation (A is attached to B)

An example of composition relation is given in illustration 6.

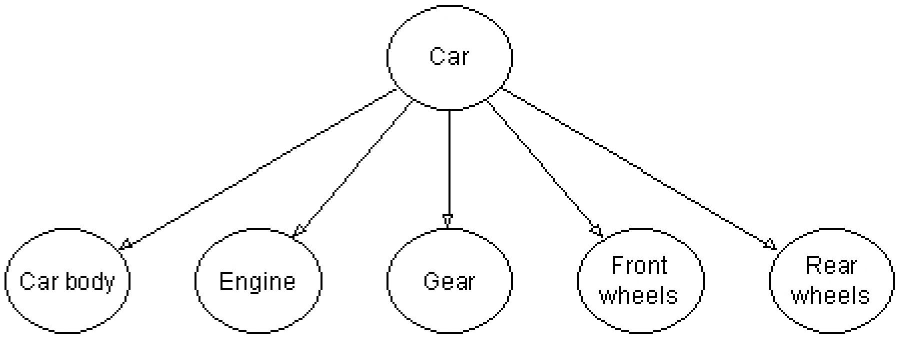


Illustration 6

Here, each arrow represents a *consists-of* relation, ie *Car body, Engine, Gear, Front wheels* and *Rear wheels* are all part of *Car*.

Illustration 7 shows a more sophisticated example. Here, *Computer* consists of *Housing, Power supply, Mainboard, Graphics card* and *Loudspeaker*. In addition, this graph tells us that all parts of the computer are attached to the component *Housing*. This information may be useful, if the user is interested in determining how much time is required to disassemble the computer system.

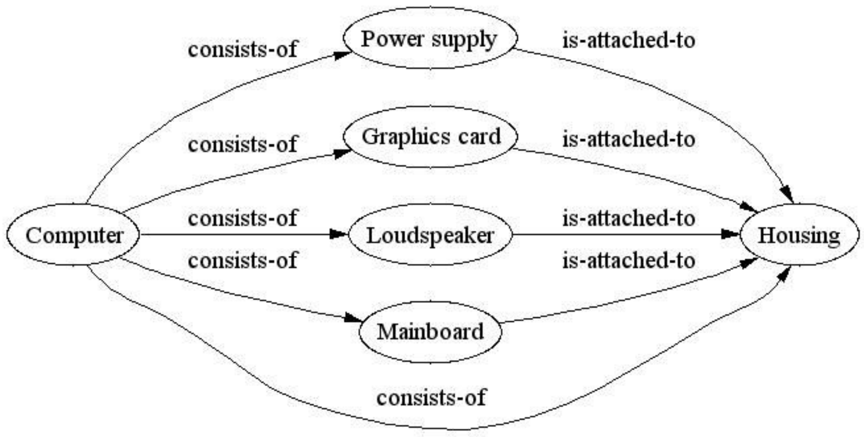


Illustration 7

Properties

Each of the graph objects (task nodes, resource nodes and edges) has several properties, which are <u>not</u> hardcoded, but can be added, edited and removed flexibly by the user.

Which properties a graph object has, is determined at runtime. Information about which properties an object has, and what their values are, is stored in the database.

Automatic UI generation

Data of each graph object can be edited via automatically generated user interface (UI) forms. The fact that all the data of an object is stored in form of properties in the database, allows the application to generate UI forms for a particular object on the fly.

This reduces the programming and maintenance effort.

Report generation mechanism

Generation of reports for the task and resource graphs is also simplified via the storage of data in form of properties (as opposed to hardcoded class attributes).

Non-Technical properties

There are also several guiding principles, which in my opinion are essential for the success of the Demetrix system.

Documentation

All more or less complex algorithms used within Demetrix should be documented in a brief form. These documents should be available via the Internet.

Where possible, the user must be able to calculate the results the application delivers, by himself/herself. Transparency may be a good USP for Demetrix.

Open-Source Development

In order to ensure that Demetrix will not die all too soon, all parts of the system, which are not customer-specific should be published under the LGPL licence.

The Application Should Be Sexy

Working with Demetrix (both as a user and as a developer) must be fun. The more time a person is willing to spend with an application, the higher the probability of success.