Ontology-Based Process Modeling and Execution using STEP/EXPRESS[*]

Arndt Mühlenfeld, Wolfgang Mayer, Franz Maier, Markus Stumptner
Advanced Computing Research Centre
University of South Australia
5095 Mawson Lakes, SA, Australia
E-mail: {muehlenfeld|wolfgang.mayer|franz.maier|mst}@cs.unisa.edu.au

Abstract

A common data format as provided by the STEP/EXPRESS initiative is an important step toward interoperability in heterogeneous design and manufacturing environments. Ontologies further support integration by providing an explicit formalism of process and design knowledge, thereby enabling semantic integration and re-use of process-information. By formalizing the process-model in EXPRESS, we gain access to the domain knowledge in the STEP application protocols. We present an approach to process modeling using different models for abstract process knowledge and implementation details. The abstract process model supports re-use and is independent of the implementation. As a result, we translate the process model in combination with the implementation model to an executable workflow.

1. Introduction

Modern industrial design manufacturing processes allow for collaboration among organizations and organizational units within large companies. Design knowledge that is spread over several design teams and systems is difficult to integrate. The lack of interoperability in heterogeneous information systems results from incompatible data formats and differences between domain models. Data exchange between design stages requires definition of mappings between data representations or the use of a common format. STEP/EXPRESS is an established standard for product data representation and solves the problem of incompatible data formats. Differences in domain models are addressed by modeling semantic knowledge in ontologies. Semantic interoperability between design disciplines is usually achieved by using a common upper ontology [12] or mappings between domain ontologies [16]. N.Guarino [4] proposes a model for information integration that uses separate ontologies for task and domain knowledge with a common upper ontology.

In this paper we present a meta-model for task ontologies of industrial processes that integrates process knowledge with artifact representation. The meta-model provides us with the means to express knowledge over artifacts and reconstruct provenance. We utilize established workflow execution engines for enacting the process model by building a meta-model for the execution environment and defining a mapping between the process and the enactment meta-models. The process meta-model has two parts, an abstract *process model* and an *implementation model*. Keeping the implementation details separate from the process model makes the conceptual model clearer and better suited for re-use. We use EXPRESS to specify our ontological model, which gives us direct access to the information models of the STEP Standard. Furthermore, we can use the same language for (a) process models and (b) artifacts represented in STEP. In order to close the gap between specification and implementation we present a mapping of process specifications to a specific workflow engine. Most workflow engines provide an execution trace of the enacted workflow and support data provenance. We present an example mapping for a specific workflow execution engine to demonstrate that our process model contains the necessary information. The process model is independent of the actual workflow engine and can be mapped to several different engines. Specification of mappings between our process meta-model and the meta-model of a specific workflow engine enables automatic translation of process models. Hence, we are free to use the workflow execution engine that best suits the target environment.

In Section 2 we introduce our meta-model and the benefits arising from formalizing it using STEP/EXPRESS. Section 3 is dedicated to the enactment of the process model. We use an implementation model and transformations to create a specific workflow that can be fed to a workflow

[*]This work was funded by the CRC for Advanced Automotive Technology under project C4-801 Process Modelling in the Automotive Industry.

execution engine. An example of a process-model transformation is given in Section 4. In Section 5 we present related work. Our contribution and future work is summarized in Section 6.

2. Process Modeling with EXPRESS

2.1. The EXPRESS Language

The STEP standard (ISO 10303) defines a collection of application protocols representing data models for different domains. EXPRESS is the modeling language used for the data models and is specified in Part 11 of the standard [6]. The language is able to represent entity-relationship concepts in an object-oriented way. Its powerful representation of constraints on data has shown to be suitable for formal specifications and meta programming [1]. An application model in EXPRESS comprises types, functions and data objects called *Entities*. Entities consist of attributes and constraints related to the attributes. Entities are the central elements of the language and represent classes of objects. As in object-oriented languages, classes are structured hierarchically by inheritance. The elements of a model are grouped into a *Schema*. Schemata are like name spaces and can be referenced by other schemata.

The language is powerful enough to express the structure of any meta-model within an EXPRESS Schema and the standardized access interface in several languages bindings allows for the generation of a meta-data management systems suited to the target systems [14].

2.2. The Process Model

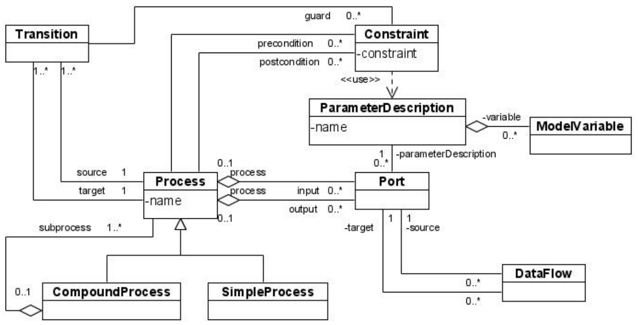

Figure 1. An overview of the meta-model in UML.

In the meta-model depicted in Figure 1 an abstract *Process* is either a *CompoundProcess* comprising one or more processes or an indivisible *SimpleProcess*. A Process has input and output *Ports* for input and output data. The data expected on a port is specified by a *ParameterDescription* which in turn includes *ModelVariables*, if it represents a complex structure like a parametric model. *Constraints* specify the behavior of the process in the role of preconditions and postconditions. A *DataFlow* connects output ports with input ports and is the basic building block for data flow in the process model. Control flow is modeled independently of the data flow by *Transitions*. A process can have many ingoing and outgoing transitions.

```
ENTITY ParameterDescription;
    description : STRING;
    variable : SET [ 0 : ? ] OF ModelVariable;
    is_similar_to : SET [ 0 : ? ] OF ParameterDescription;
END_ENTITY;

ENTITY Port
   SUBTYPE OF ( NamedEntity );
    descr : ParameterDescription;
END_ENTITY;

ENTITY DataFlow;
    source : Port;
    target : Port;
WHERE
    data_is_compatible :
        ( source.descr = target.descr ) OR
        ( source.descr IN target.descr.is_similar_to ) OR
        ( target.descr IN source.descr.is_similar_to );
END_ENTITY;
```

Listing 1. Specification of entities ParameterDescription and DataFlow in EXPRESS

The process model presented above does not specify the properties of the data exchanged between processes. The data model makes use of the application protocols of the STEP Standard and is domain dependent. We use AP 214 to model the data of the design optimization process, because it contains the domain knowledge for automotive design processes. The full data model is beyond the scope of this document. However, as an example for the expressiveness of EXPRESS, we have formulated a rule that ensures that *Dataflows* connect only *Ports* with ”related” parameter descriptions. Therefore, we define the attribute *is_similar_to* in *ParameterDescription*, which represents the association to related objects (see Listing 1). Entity *DataFlow* has a rule stating that only ports that have the same parameter description, or parameter descriptions that are similar to each other, are allowed as source and target objects.

2.3. The Implementation Model

The process meta-model describes abstract properties, process components and their relationships, but not how a process can be executed or where the data for its ports are stored. This information is part of the implementation model. The implementation meta-model in Figure 2 defines two types of process instances, *WebService* and *Executable*. For the sake of brevity, details on Web Services are omitted. An *Executable* has at least two possibilities for its input. In

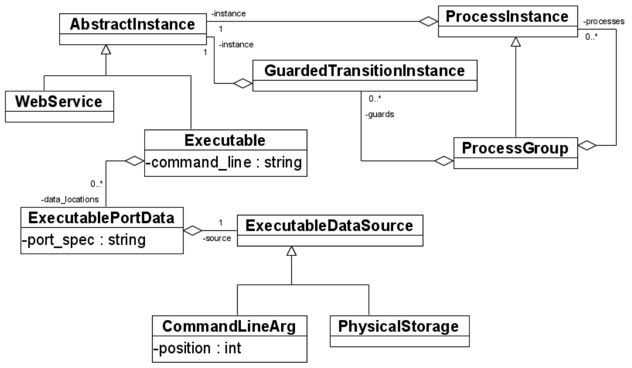

Figure 2. An overview of the implementation model in UML.

```
FUNCTION get_port_specs_from_portdata ( port_list : AGGREGATE OF PortData )
        : SET OF EntityName;
    LOCAL
        result : SET OF EntityName;
    END_LOCAL;

    REPEAT i := LOINDEX ( port_list ) TO HIINDEX ( port_list );
        result [ i] := port_list.port_spec;
    END_REPEAT;

    RETURN ( result );
END_FUNCTION;

ENTITY AbstractInstance;
    description : STRING;
    spec : EntityName;
DERIVE
    port_list : SET OF EntityName := [ ];
END_ENTITY;

ENTITY Executable SUBTYPE OF ( AbstractInstance );
    commandline : STRING;
    data_locations : LIST OF ExecutablePortData;
DERIVE
    SELF\AbstractInstance.port_list : SET OF EntityName :=
                        get_port_specs_from_portdata ( data_locations );
END_ENTITY;

ENTITY GuardedTransitionInstance;
    instance : AbstractInstance;
DERIVE
    guardedtransition_spec : EntityName := instance.spec;
WHERE
    has_result :
        SIZEOF ( QUERY (p <* instance.port_list | p = 'result')) = 1;
END_ENTITY;
```

Listing 2. Specification of process implementations in EXPRESS

our model, it can receive input from physical storage (*PhysicalStorage*), e.g., a File, or as a command-line parameter (*CommandLineArg*). Physical processes are part of a *ProcessInstance* or a *GuardedTransitionInstance*. The former is the realization of a *Process* in the process-model; the latter is a process that evaluates the guards of a *Transition* in the process-model. Realizing the evaluation of guards for conditional execution as a physical process keeps the model independent of the constraint language. Not visible in the UML-diagram, but specified in EXPRESS is the constraint *has_result*. It expresses the invariant that an entity of class *GuardedTransitionInstance* has to provide a port named 'result' for the evaluation result (see Listing 2). This is possible, because *AbstractInstance* provides an attribute containing a set of instantiated ports which is overridden by its derived concrete entities. For example, in *Executable* the port list is extracted from the list of *ExecutablePortData*.

3. Workflow Execution

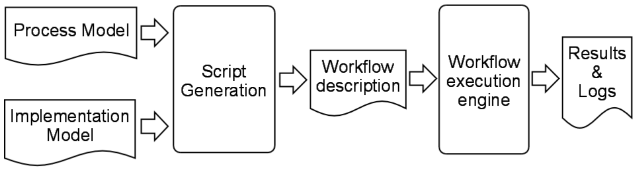

Figure 3. Process execution.

Our process model in combination with the implementation model contains all necessary data for enactment. A couple of workflow engines exist that have already reached the required maturity for production environment (e.g. Kepler [9], MyGrid/Taverna [13]). We do not want to tie our model to a specific workflow system with our model but prefer to have the choice to use the best system for particular requirements. We keep the model independent of the workflow system by defining transformations that generate the workflow description for the target system from the model. We chose the workflow engine Taverna to demonstrate the approach, because it is intuitive and simple to use, but powerful enough to support sophisticated workflows.

3.1. The Workflow Execution Engine Taverna

A workflow in Taverna [13] consists of inputs, outputs, one or more processors and the data flows between them (see Figure 4). Processors have an interface for inputs and outputs. The outputs of processors can be connected to other inputs or the workflow outputs. The whole workflow is data flow oriented and the order of execution is defined by the data dependencies between processors. A processor is executed as soon as it has got all of its inputs and processors may execute concurrently. The data flow can lead from one output to inputs of more than one processor. If an input is connected to more than one output, the first output to deliver the data "wins". In addition to the data dependencies it is possible to restrict the execution order of processors by defining temporal constraints called "Coordinate from". By defining a "Coordinate from" association between processors A and B, A will only execute when B has completed. The usual method of creating a workflow in Taverna is by using its graphical user interface (GUI). However, all workflows created by using the GUI are passed to the execution engine in the workflow description language Scufl

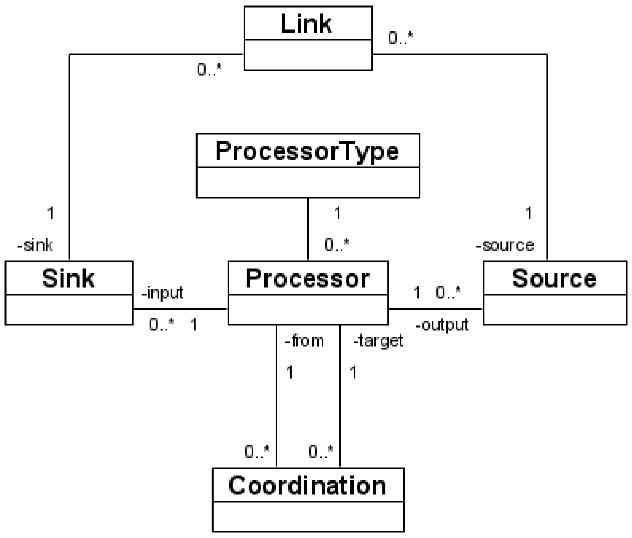

Figure 4. Meta-model of workflows in Taverna.

(Simple Conceptual Unified Flow Language). It is a simple XML-based format representing the elements and links of the workflow and can be used to execute the workflow without the GUI. It contains the workflow description, a couple of processors connected by data flow (link) and control flow edges (coordination) and the inputs (source) and outputs (sink) of the workflow. For our current project, only two types of processors are of interest: the local process FailIfFalse for conditional transitions and the Beanshell Scripting Host for program execution.

3.2. Mapping the Model to a Workflow

The data provided in the process model is sufficient to define the nodes and links in the workflow. The only missing information is the definition of the implementation of the processor nodes. This definition is provided by the implementation model. The four main elements of our process that need to be represented in the workflow description are *Process*, *DataFlow*, *Transition* and *GuardedTransition*. A process maps to a processor, where the interface of the processor is defined by the ports of the process. Data flow objects have a straight-forward equivalent in the workflow description; they are represented by links between processor interfaces. A transition has no direct equivalent in Scufl. The closest representation is a coordinating link, but coordination is more restrictive than a transition. Consider a process A with two transitions coming from process B and C. In our model, the precondition of process A specifies, if the process waits until both or only one of the processes B and C have finished execution. In Scufl, both processes have to finish execution successfully if A is coordinated from B

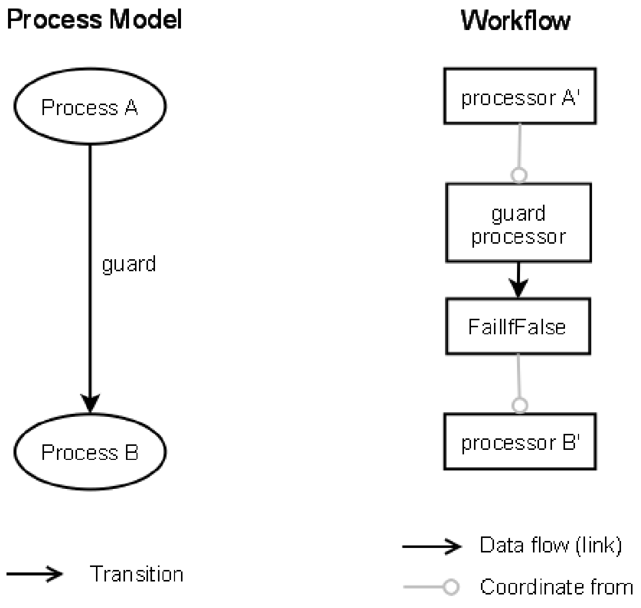

Figure 5. Implementation of a guarded transition in the target workflow.

and C. We keep it simple and stick to the behavior of Scufl's coordination element and define that a process has to be reached by all transitions in order to start its execution. The most sophisticated part of the workflow generation is the implementation of a transition with guards, i.e., a GuardedTransition. We implement it in the workflow description by using a processor for the evaluation of the constraint followed by a conditional node (FailIfFalse) and a "coordinate from"-edge to the conditional node (see Figure 5). This rough sketch of how to implement the workflow in Scufl gives a first impression on how to achieve our goal. In order to formalize the transformation we first start by defining the source and target models.

Definition 1
Let $X(P_X, I_X, O_X, D_X, C_X)$ represent a workflow, where

P_X *... set of processors $\{p_i(i_{i,1} \dots i_{i,M}, o_{i,1} \dots o_{i,N}) | 0 < i \leq N_X\}$, $i_{i,j}$ .. input j of processor i, $0 < j \leq M_i$, $o_{i,j}$ .. output j of processor i, $0 < j \leq N_i$,*

I_X *... set of inputs of the workflow, represented as processor outputs $\{o_{0,j} | 0 < j \leq N_0\}$,*

O_X *... set of outputs of the workflow, represented as processor inputs $\{i_{0,j} | 0 < j \leq M_0\}$,*

D_X *... data flow between processors represented by a set of links $\{d_\nu = o_{i,j} \rightarrow i_{k,l}\}$.*

C_X *... set of coordinations $c(p_i, p_j)$, coordinate processor p_i from p_j.*

Definition 2
Let $M(\Pi_M, R_M, D_M, T_M, G_M)$ denote a process model, where

Π_M *... set of processes* $\{\pi_i | 1 \leq i \leq N_\Pi\}$,

R_M *... set of ports* $\{r_i | 1 \leq i \leq N_R\}$, $R_i \in R_M$ *... set of ports of process* π_i,

D_M *... set of data flow links* $\{l_\nu(r_i \to r_j)\}$,

T_M *... set of transitions* $\{t_\nu(\pi_i \to \pi_j)\}$,

G_M *... set of guarded transition* $\{g_\nu(\gamma_\nu, \pi_i \to \pi_j)\}$, γ_ν *... guard*

Using the two definitions we can write the algorithm to obtain a workflow X from the process model M as follows:

1. $\forall \pi \in \Pi_M$: create processor p_i with inputs I_i and outputs O_i corresponding to the ports in R_π.
2. $\forall t(\pi_i \to \pi_j) \in T_M$: create "coordinate from" $c(p_i, p_j)$.
3. $\forall g(\gamma, \pi_i \to \pi_j) \in G_M$:
   (a) create processor p_k with inputs I_γ and output o_γ.
   (b) create processor "FailIfFalse" (p_f) with input i_f.
   (c) create data flow $d_\gamma = o_\gamma \to i_f$.
   (d) create coordinations $c_{\gamma,1}(p_i, p_k)$ and $c_{\gamma,2}(p_f, p_j)$.
4. $\forall l_\nu(r_i \to r_j) \in D_M$: create data flow $d_\nu = (o(r_i) \to i(r_j))$.

4. Example workflow

We tested the workflow generation on a process model for multi-disciplinary design optimization in the automotive industry [15]. Part of the process is the generation of an instance mesh from a geometric model, which is later used by the finite element analysis. The process model of the instance mesh generation contains conditional transitions that select between two possible paths in the process. Depending on the value of a flag (*run_geometry_flag*), either a new mesh is generated or the resulting mesh of a previous run is fetched.

The implementation model contains four executables for the two sub-processes and the evaluation of the constraints on the two guarded transitions, respectively. After applying the transformations presented in the previous section, we get a workflow that can be visualized in the Taverna GUI (see Figure 6) and executed inside the GUI or with the standalone workflow execution engine.

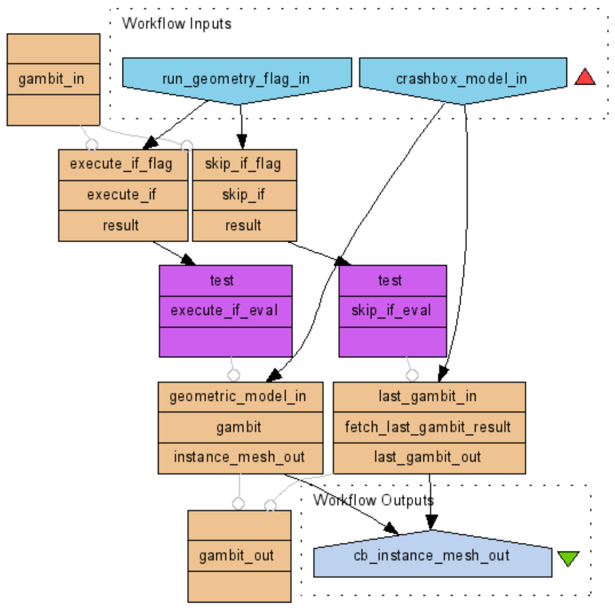

Figure 6. The example workflow in Taverna.

5. Related Work

A number of other process ontologies exist [2, 3, 8]. But to our knowledge, no other work uses EXPRESS to formalize a process model with workflow enactment. An extensive evaluation of other ontologies in the context of industrial design processes can be found in [10].

Our approach consolidates process and artifact ontologies under a common STEP/EXPRESS meta-model. We chose EXPRESS because it comes with huge artifact ontologies and is well-suited for meta-modeling. It can be argued that other standard languages like the *process specification language* PSL are better suited for process modeling. PSL offers a rigorous basis for verifiable semantic definitions, but lacks support for context relationships and needs better definitions of process artifacts [5]. STEP and its application protocols provide in contrast sophisticated domain models for artifact and process modeling.

Mimoune et al. [11] exchange data between heterogeneous database systems using a generic meta-schema formalized in the EXPRESS language to overcome the difficulties arising from different conceptual models for the same implementation and structural differences between implementations of the same conceptual model. Their approach focuses on the definition of mappings between data base schemata.

Work has been done to map from business processes to workflows including ontological mapping between the output of one process and the input of another process. However, a general mapping from control flow oriented metamodels to data flow oriented systems is hard to achieve [7].

6. Conclusion

Data exchange between different stages of an industrial process is difficult because of the heterogeneity of the involved systems. The common data format standard STEP/EXPRESS helps to overcome the structural differences. The standard defines different encodings and language bindings for its specification language EXPRESS. These "implementation methods" comprise a clear-text and an XML representation and bindings to the programming languages C,C++ and Java. STEP is not intended to provide a common conceptual model, but provides specialized models for domain knowledge. We use process modeling to capture process knowledge explicitly. In addition to allowing easier re-use of process components, explicit process knowledge supports execution tracking so that the provenance of the results is retained.

We propose EXPRESS as the specification language for the ontologies, because thereby we can directly use the domain knowledge specified in the application protocols of the STEP standard. STEP is already used as a common data format for many of the process artifacts in the automotive industry and the data is accessible by our process model without additional structural conversion overhead. Furthermore, writing our models in STEP/EXPRESS allows us to use the same tools as already used for data modeling.

Enactment is another important aspect of process modeling. We have shown that it is possible to use an abstract process model, which is independent of the target platform and build a workflow description for a specific workflow execution engine from this model. We have successfully demonstrated the necessary transformations for the workflow engine Taverna. However, because the process model is independent of the workflow engine, we can use any other workflow execution engine that provides the necessary functionality. In the future, we plan to define transformations for at least one other workflow execution environment.

References

[1] Y. Ait-Ameur, F. Besnard, P. Girard, G. Pierra, and J. C. Potier. Formal specification and metaprogramming in the EXPRESS language. In *Intl.Conf. on Software Engineering and Knowledge Engineering (SEKE)*, pages 181–188, 1995.

[2] B. Chandrasekaran, J. Josephson, and R. Benjamins. The ontology of tasks and methods. In *Proceedings of the 11th Knowledge Acquisition Modeling and Management Workshop, KAW'98*, Banff, Canada, Apr. 1998.

[3] J. Gero and U. Kannengiesser. A function-behavior-structure ontology of processes. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, 21:379–391, 2007.

[4] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98*, pages 3–15, Trento, Italy, 1998. IOS Press.

[5] A. Gunendran, R. Young, A. Cutting-Decelle, and J. Bourey. Organising manufacturing information for engineering interoperability. In R. Goncalves, J. Muller, K. Mertins, and M. Zelm, editors, *Proceedings of IESA 2007: Enterprise Interoperability II New challenges and Approaches*, pages 587–598. Springer-Verlag London, 2007.

[6] International Organization for Standardization. *ISO 10303-11:1994: Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.* International Organization for Standardization, Geneva, Switzerland, 1994.

[7] S. Jablonski. On the complementarity of workflow management and business process modeling. *SIGOIS Bull.*, 16(1):33–38, 1995.

[8] Y. Kitamura, Y. Koji, and R. Mizoguchi. An ontological model of device function: industrial deployment and lessons learned. *Applied Ontology*, 1(3–4):237–262, 2006.

[9] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system, 2005.

[10] F. Maier, W. Mayer, M. Stumptner, and A. Mühlenfeld. Ontology-based process modelling for design optimisation support. In *Third International Conference on Design Computing and Cognition (DCC'08)*, Atlanta, USA, June 2008. To appear.

[11] M. E.-H. Mimoune, G. Pierra, and Y. A. Ameur. An ontology-based approach for exchanging data between heterogeneous database systems. In *ICEIS (1)*, pages 512–524, 2003.

[12] P. H. P. Nguyen and D. Corbett. Building corporate knowledge through ontology integration. *Advances in Knowledge Acquisition and Management*, 4303/2006:223–229, 2006.

[13] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18:1067–1100, 2006.

[14] P. Saliou, A. Plantec, and V. Ribaud. Metaprogramming with EXPRESS and SQL. In *International Workshop Declarative Meta Programming, DMP02. University of Edinburgh*, Dec 2002.

[15] C. Seeling. User manual for the crash-box MDO reference problem. Internal publication, VPAC Ltd, Melbourne, 2007.

[16] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information — a survey of existing approaches. In H. Stuckenschmidt, editor, *IJCAI–01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.