Integrated Design and Process Technology, **IDPT-2003**
Printed in the United States of America, June, 2003
©2003 Society for Design and Process Science

UML-Based Definition of Integration Models for Incremental Development Processes in Chemical Engineering

Simon M. Becker, Bernhard Westfechtel
Computer Science III, Aachen University of Technology, D-52056 Aachen
(sbecker—bernhard)@i3.informatik.rwth-aachen.de

ABSTRACT:
In development processes multiple tools are used to describe different aspects of the developed product. The resulting information is stored in heterogeneous documents that are technically independent but whose contents are closely related on the semantic level. Thus, if one document is changed, these changes have to be propagated to dependent documents in order to restore mutual consistency. Therefore, there is a need for incremental integration tools which assist developers in consistency maintenance. Driven by this need, we realized a framework for building incremental integration tools which is currently being used in the chemical engineering domain. Integration tools are based on models of the related documents and their mutual relationships. These models are defined in the Unified Modeling Language (UML).

I. Introduction

Development processes in different engineering disciplines such as e.g. mechanical, chemical, or software engineering are highly complex. The product to be developed is described from multiple inter-dependent perspectives. The results of development activities are stored in *documents* such as e.g. requirements definitions, software architectures, or module implementations in software engineering or various kinds of flow diagrams and simulation models in chemical engineering. These documents are connected by mutual dependencies and have to be kept consistent with each other. Thus, if one document is changed, these changes have to be propagated to dependent documents in order to restore mutual consistency.

Tool support for maintaining inter-document consistency is urgently needed. However, conventional approaches suffer from severe limitations. For example, batch converters are frequently used to transform one design representation into another. Unfortunately, such a transformation cannot proceed automatically if human design decisions are required. Moreover, batch converters cannot be applied to propagate changes incrementally. Furthermore, hypertext-like tools provide base mechanisms for establishing inter-document links which provide for traceability of the development process. However, usually such tools do not incorporate semantic knowledge on the relationships between inter-dependent documents. Therefore, support for consistency control and change propagation is severely limited.

In [5], we presented a framework for building integration tools that offer more sophisticated support for maintaining inter-document consistency. Our approach is characterized by the following features:

1. Relationships between inter-dependent documents are stored in separate data structures, which are called *integration documents*. An integration document consists of a set of links connecting patterns of the related documents.
2. Integration tools are driven by *rules* which specify the relationships between source and target patterns. When a rule is applied, a corresponding link is inserted into the integration document.
3. Integration tools operate *incrementally* inasmuch as they may be used to propagate changes between inter-dependent documents. That is, when one document is changed, only the effects of this change are propagated to the related document; unaffected parts of the document are retained.
4. In general, integration tools are *bidirectional*, i.e., they can be used to propagate changes from the source to the target document and vice versa.
5. In general, it cannot be assumed that changes can be propagated automatically. Rather, integration tools operate *interactively*, i.e., they rely on human decisions concerning the selection of which rule to apply in ambiguous situations.
6. *Change propagation* is only one of multiple functions of integration tools. In addition, they can be used for *consistency analysis* and *browsing* (traversing of links).
7. Finally, integration tools are *model-based*, i.e., they are based on models of the related documents and their mutual relationships.

In this paper, we focus on the last feature mentioned above. We will demonstrate how the *Unified Modeling Language* (UML [7]) can be used to define the relationships between different document types. First class diagrams are used to define associations between related classes contained in the documents' models. Next, in collaboration diagrams corresponding patterns are identified and related to each other. Finally, executable rules based on graph transformations are derived from the corresponding patterns.

The rest of this paper is structured as follows: In Section II, we present a motivating example from the chemical engineering domain. In Section III, we give an overview of our modeling approach. After these preparations, the core part of the paper follows. Section IV presents the meta model, which is used in Section V to define actual integra-

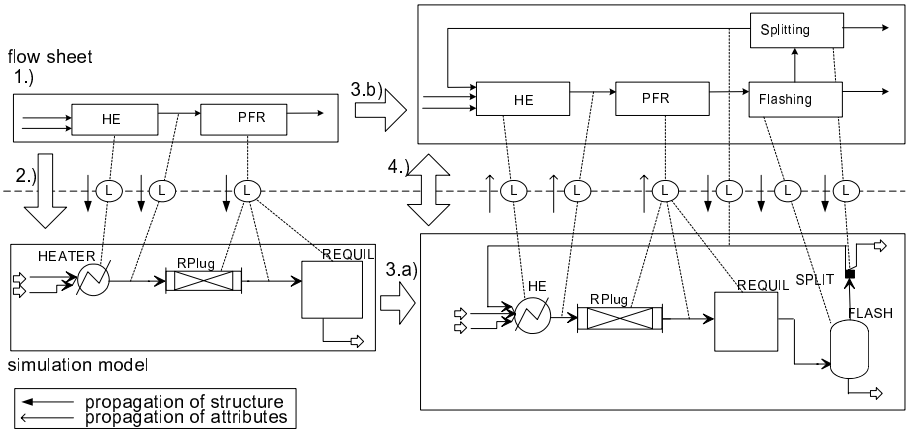

Fig. 1. Integration between flow sheet and simulation model

tion models. Section VI discusses related work, and Section VII concludes the paper.

II. Motivating Example

While the concepts underlying incremental integration tools are fairly general and domain-independent, we have focused in particular on the chemical engineering domain. Therefore, we take the motivating example from this domain. More specifically, we are interested in the relationships between flow sheets and simulation models. A *flow sheet* describes the chemical process to be designed, while a *simulation model* serves as input to a tool for performing steady-state or dynamic simulations. Heterogeneous tools may be used for creating flow sheets and simulation models, respectively. In the following, we assume that the flow sheet is maintained by COMOS PT [22] and simulations are performed in Aspen Plus [23], both of which are commercial tools for chemical engineering.

In chemical engineering, the flow sheet acts as a central document for describing the chemical process. The flow sheet is refined iteratively so that it eventually describes the chemical plant to be built. Simulations are performed in order to evaluate design alternatives. Simulation results are fed back to the flow sheet designer, who annotates the flow sheet with flow rates, temperatures, pressures, etc. Thus, information is propagated back and forth between flow sheets and simulation models. Unfortunately, the relationships between them are not always straightforward. To use a simulator such as Aspen Plus, the simulation model has to be composed from pre-defined blocks. Therefore, the composition of the simulation model is specific to the respective simulator and may deviate structurally from the flow sheet.

Figure 1 illustrates how an incremental integration tool assists in maintaining consistency between flow sheets and simulation models. The chemical process taken as example produces ethanol from ethen and water. Flow sheet and simulation model are shown above and below the dashed line, respectively. The integration document for connecting them contains links which are drawn on the dashed line[1]. The figure illustrates a design process consisting of four steps:

1. An initial flow sheet is created in COMOS PT. This flow sheet is still incomplete, i.e., it describes only a part of the chemical process (heating of substances and reaction in a plug flow reactor, PFR).
2. The integration tool is used to transform the initial flow sheet into a simulation model for Aspen Plus. Here, the user has to perform two decisions. While the heating step can be mapped structurally 1:1 into the simulation model, the user has to select the most appropriate block for the simulation to be performed. Second, there are multiple alternatives to map the PFR. Since the most straightforward 1:1 mapping is not considered sufficient, the user decides to map the PFR into a cascade of two blocks. These decisions are made by selecting among the different possibilities of rule applications the tool presents to the user.
3. The simulation is performed in Aspen Plus, resulting in a simulation model which is augmented with simulation results. In parallel, the flow sheet is extended with the chemical process steps that have not been specified so far (flashing and splitting).
4. Finally, the integration tool is used to synchronize the parallel work performed in the previous step. This involves information flow in both directions. First, the simulation results are propagated from the simulation model back to the flow sheet. Second, the extensions are propagated from the flow sheet to the simulation model. After these propagations have been performed, mutual consistency is reestablished.

The example presented above demonstrates the functionality of the integration tool, but it does not show how this is achieved. As mentioned, integration tools are model- and

[1] This is a simplified notation. Some details of the document and integration model introduced later are omitted.

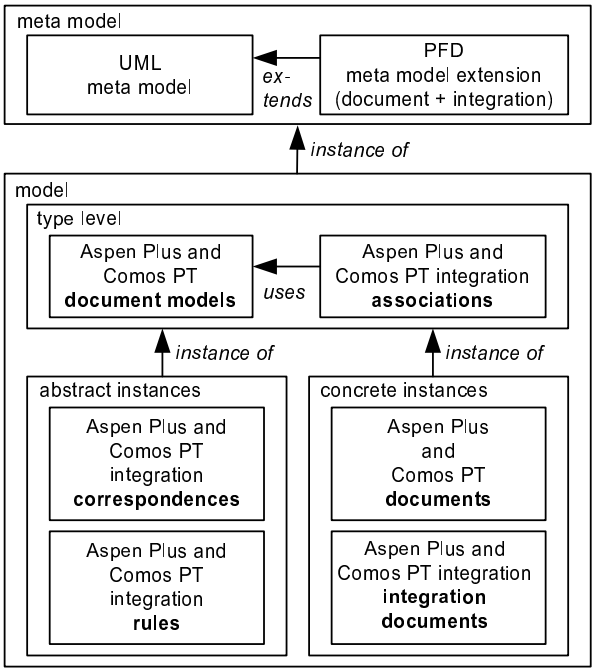

Fig. 2. Levels of modeling

rule-based. In the next sections, we will explain the modeling framework underlying incremental integration tools.

III. Overview

A. Levels of Modeling

In order to define the documents that are to be integrated, their relations, and the rules that control the integration in a structured way, a multi-layered modeling approach based on UML is used. It follows the paradigm of meta modeling, i.e., each layer defines the constructs to be used on the next lower layer. To express this paradigm in UML, we applied the approach introduced in [18], which is based on the extension of the UML meta model.

Figure 2 shows the different layers and their interdependencies. On the meta model level, the UML meta model is enriched with modeling constructs that can be used on the model level to define process flow diagram-like document types and to build models related to the integration of such documents. The process flow diagram (PFD) specific meta model is a specialization of an even more abstract meta model for the integration of arbitrary documents which is omitted here.

Like in standard UML, the model level consists of the type level and the instance level. On the type level *document models* for specific types of documents are defined. They are expressed as class hierarchies describing the documents' underlying type systems. In our example, documents containing simulation models for Aspen Plus and flow sheets for Comos PT are defined. To be able to perform an integration of these documents, *associations*[2] between the classes contained in the documents' class hierarchies are drawn.

On the instance level, we distinguish between concrete instances and abstract instances. Correspondences and rules are abstract instances. A *correspondence* relates a pattern out of one document to a pattern of another document. A pattern describes a template which can be matched against an actual document. Here, Aspen Plus patterns are related to Comos PT patterns. Correspondences can be enriched to become executable integration *rules*. These rules are used to perform the integration of two documents.

Concrete instances are used to describe actual document instances like Aspen Plus simulation models, Comos PT flow sheets, and integration documents. An *integration document* is created for each integration of two documents. It contains *links* between related parts of the documents.

The different layers of modeling are explained in detail in Sections IV and V.

B. Modeling Process

Figure 3 shows the interrelations between the different parts of the model from a more practical point of view. The meta model serves as basis both for the implementation of integration tools and the rule modeling process. It is defined according to domain specific knowledge like, in our case, the information model CLiP [4] for chemical engineering and the requirements concerning integration functionality.

Basically, there are two ways to define integration rules: top down, before the integration tool is applied, or bottom up, based on situations occuring during the usage of the tool. It is most likely that in practice first a basic set of rules is defined top down by a modeling expert and then the rule base is extended bottom up by the engineer using the integration tool. Before any rules can be defined, the documents to be integrated have to be modeled on type level, which is not shown in the figure. Next, associations have to be defined on type level that declare types for possible correspondences on the abstract instance level. Again, for both tasks domain specific knowledge has to be used.

Following a top down approach, now correspondences on the abstract instance level are modeled based on the associations on type level. These are then interactively refined to integration rules. The resulting set of rules is used by the integration tool to find corresponding parts of source and target document and to propagate changes between these two documents. The correponding document parts are related by links stored in the integration document. If no appropriate rule can be found in a given situation, the chemical engineer performing the integration can manually modify source and target document and add links to the integration document.

[2] These associations are different from the associations used in the UML standard. In the following, always the prefix 'UML' is used if it is referred to UML associations.

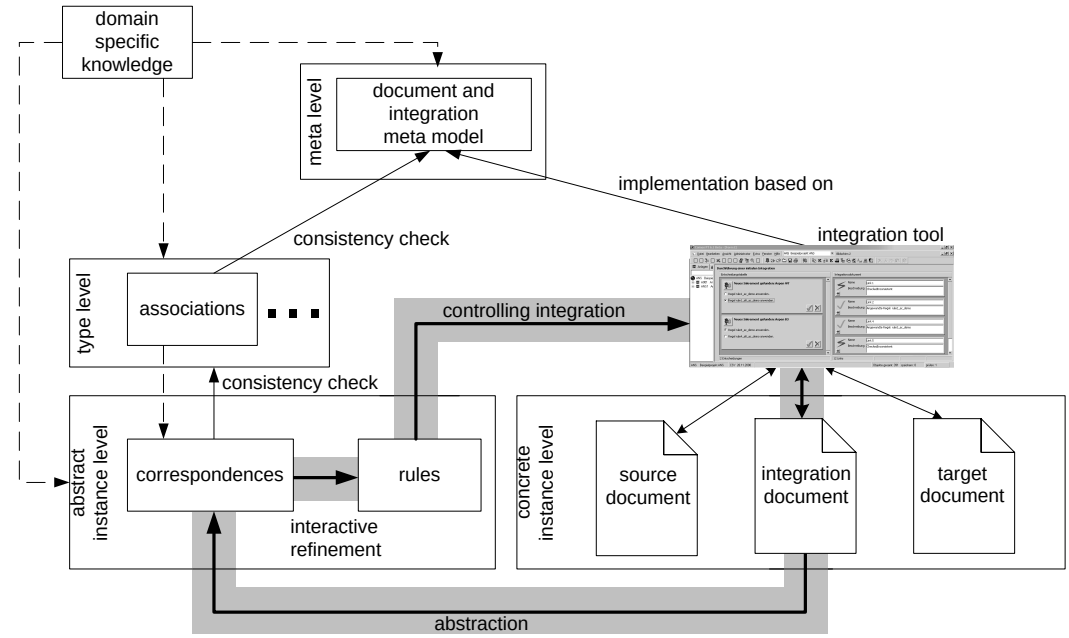

Fig. 3. Modeling process

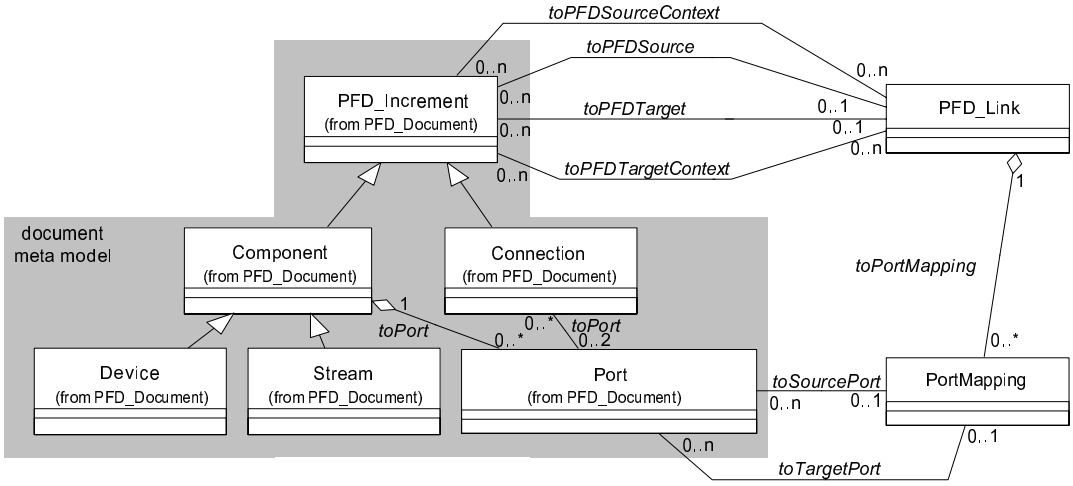

Fig. 4. Meta model

To extend the rule base bottom up, the links entered manually in the integration document can be automatically abstracted to correspondences between patterns. Next, a consistency check against the associations on type level is performed. If the correspondences are valid, the engineer is now guided through the interactive refinement of the correspondence to integration rules by a simplified modeling tool. The rules are added to the rule base and can be used for the following integrations.

IV. Meta Model

The meta model extends the UML meta model with constructs that can be used to define PFD document types and associations between them. Figure 4 shows a UML class diagram which depicts the meta model[3]. The document related part of the meta model is marked grey, the rest is integration specific. The document meta model defines meta classes that are instantiated on the type level to define the documents' type hierarchies. The class PFD_Increment is the root of the type hierarchy. Only instances of this class can be related by an association on type level. A PFD_Increment can be either a Component or a Connection.

[3]This is an informal presentation of the meta model. The new classes and UML associations have to be defined as in [18]. The cardinalities defined in the meta model do not refer to the next lower level, the type level, but to the instance level. In a formal notation they would have to be defined as additional constraints on the meta level.

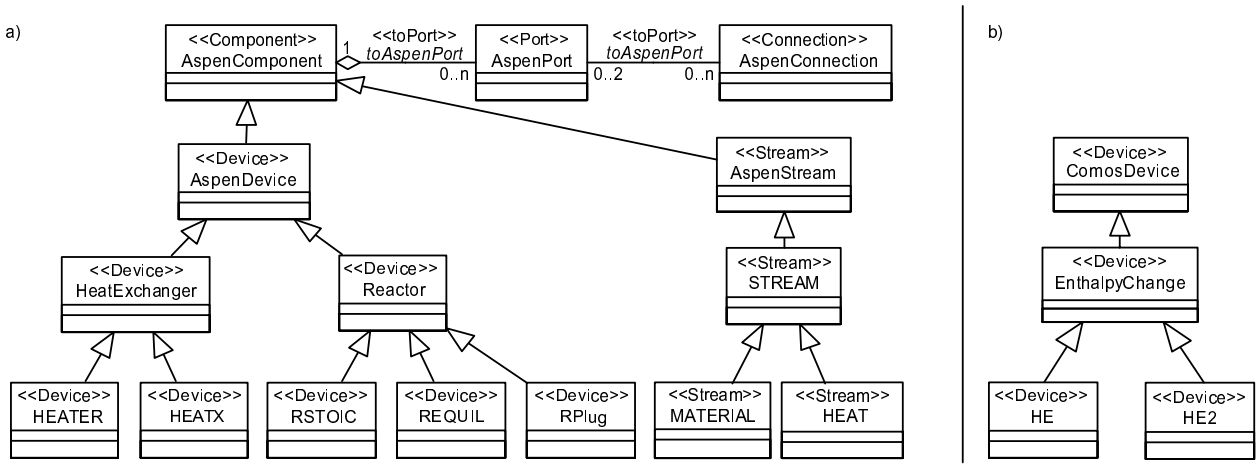

Fig. 5. Document models of Aspen Plus (a) and Comos PT (b) (excerpt)

A Component is a chemical Device that processes chemical substances or a Stream that transports them. Each Component can aggregate zero to many objects of class Port. An additional constraint which is not depicted here assures that streams can have only two ports. A Connection connects two ports, which serves to represent the topological structure of the flow sheet.

Instances of the integration specific meta class PFD_Link are used on type level to express associations between increment types. Each increment type can be referenced by at most one link as target increment (toPFDTarget) or source increment (toPFDSource). This is ensured by the relations' cardinalities and an additional constraint which is omitted here. An increment type can be referenced by an arbitrary number of links as context increment (toPFDTargetContext, toPFDSourceContext). One of the two documents that are to be integrated is chosen to be the source document containing source increments and the other to be the target document containing target increments. This has no influence on the direction in which information is propagated between both documents. Increments referenced by a link as context increments cannot be modified by the resulting rule. Each link can be extended by relating ports of the source document to corresponding ports in the target document via instances of the meta class PortMapping. The knowledge of corresponding ports is needed by integration rules that automatically reconnect components that are created in one document by other rules according to the topological structure in the other document.

V. Model

A. Type Level

On the type level the type hierarchies of the source and the target documents are modeled and associations between classes of the two hierarchies are defined. The instance-of relationship between classes on the type level and their meta classes on the meta level is expressed with stereotypes. A class with the stereotype <<mclass>> is an instance of the metaclass mclass.

In Figure 5 a) an excerpt of the type hierarchy of Aspen Plus is depicted. The class AspenComponent which is an instance of the meta class Component is the common super class for AspenDevice (instance of Device) and AspenStream (instance of Stream). Beneath these classes the simulation components available in Aspen Plus for the simulation of devices and streams are modeled as further subclasses. For instance, the class HeatExchanger is a superclass for all classes of heat exchangers like HEATER and HEATX. The document model of Comos PT is structured similarly. Figure 5 b) shows an excerpt containing some of the heat exchangers available for the definition of flow sheets in Comos PT. They are grouped by the superclass EnthalpyChange.

Beside the documents' type hierarchies associations between their types are modeled on type level. For the top down modeling approach these associations are used to explicitly express domain knowledge. For the bottom up modeling approach they are used to check the validity of corresponding patterns on the abstract instance level. Associations are expressed with link classes that are related to the associated classes via UML associations.

Type level associations can be defined on different levels of abstraction. This can be illustrated with the help of the examples in Figure 6. Part a) shows an association between the classes HEATER of the Aspen Plus type hierarchy and HE of the Comos PT type hierarchy. The class HeaterLink is an instance of the meta class PFD_Link and expresses the association. A class HeaterPortMapping is defined that can be used on the abstract instance level to specify which ports correspond to each other. The cardinalities of the toHeater and toHE UML associations are one-to-one and the connected heater classes both are leaves of the type hierarchies (see Figure 5).

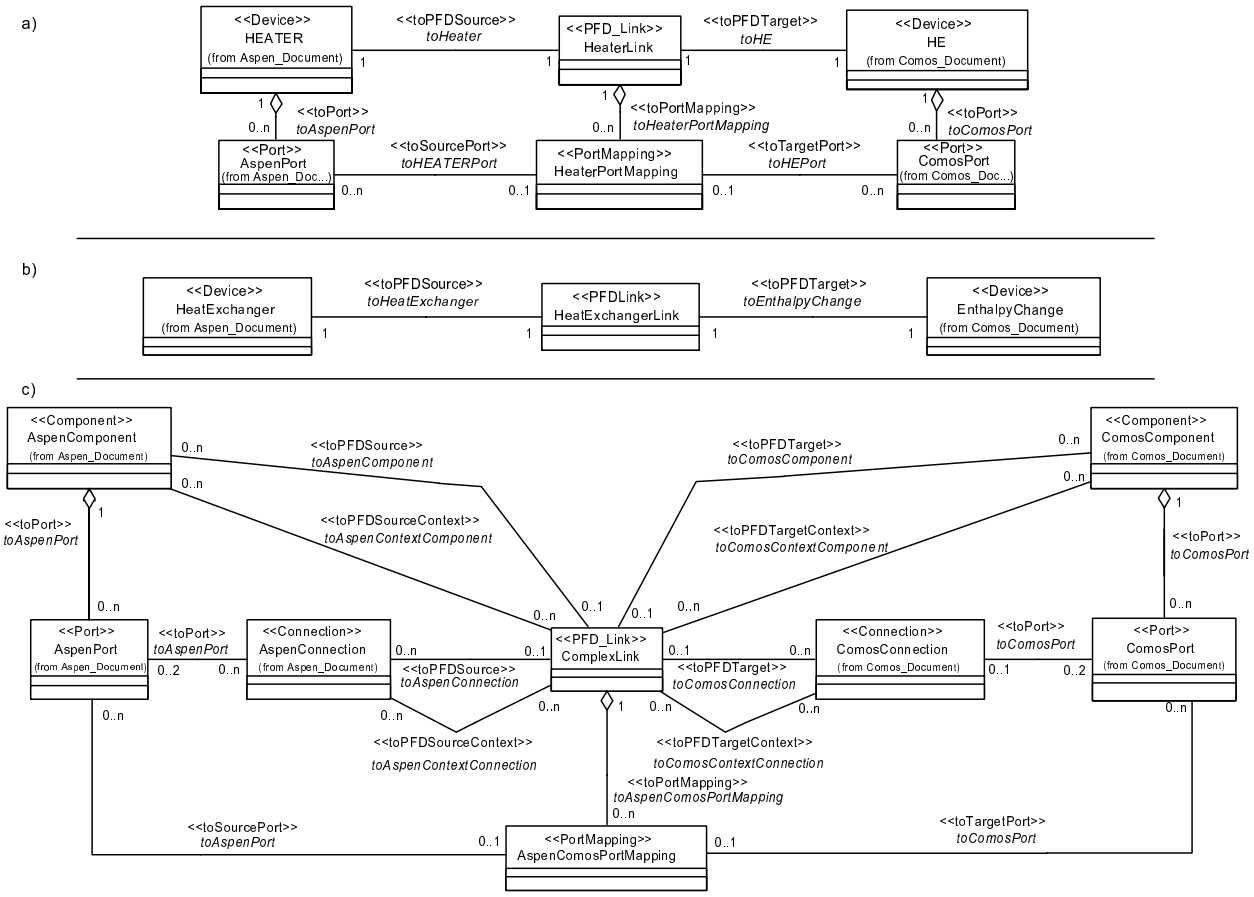

Fig. 6. Type level correspondences on different levels of abstraction

The association in Figure 6 b) is similar to the one explained before, but in this case the related classes are not leaves of the type hierarchies. The port mapping is omitted in this figure.

Associations like the two explained so far can be directly derived from domain specific knowledge. For example, in our scenario they are gained from UML associations between partial models of the chemical engineering data model CLiP [4].

The most generic association is depicted in Figure 6 c). This association allows the definition of arbitrary corresponding patterns on the abstract instance level. Thus, all corresponding patterns that are syntactically correct are an instance of this association. Of course, there are more associations defined for our running example which are omitted here.

B. Abstract Instance Level

On the abstract instance level correspondences are defined as instances of the associations on the type level. Then, the correspondences can be refined to become executable integration rules.

B.1 Correspondences

Each correspondence relates one pattern of the source document to one pattern of the target document. A pattern describes a template consisting of components, ports, connections and their relations which can be matched against the contents of concrete document instances.

The (static) UML collaboration diagram in Figure 7 shows the correspondence of an Aspen Plus heater component (S1) and a Comos PT heater component (T1). This correspondence is an instance of the association in Figure 6 a). Because the association relates only one increment type with cardinality one-to-one to another increment type and both types are leaves of their type hierarchies, the main part of the correspondence can be unambiguously derived from the association. This is done by instantiating the link class and both heater components of the association. Only the ports and their mappings have to be defined manually. In this example, the Aspen Plus heater has only one input (S1.P1) and one output port (S1.P2) for the substances that have to be heated. They are related via portmappings (L.M1, L.M2) to the corresponding ports of the Comos PT heater (T1.P1, T1.P2). The additional ports (T1.P3, T1.P4) of the Comos PT heater that are used for the cooling/heating fluid

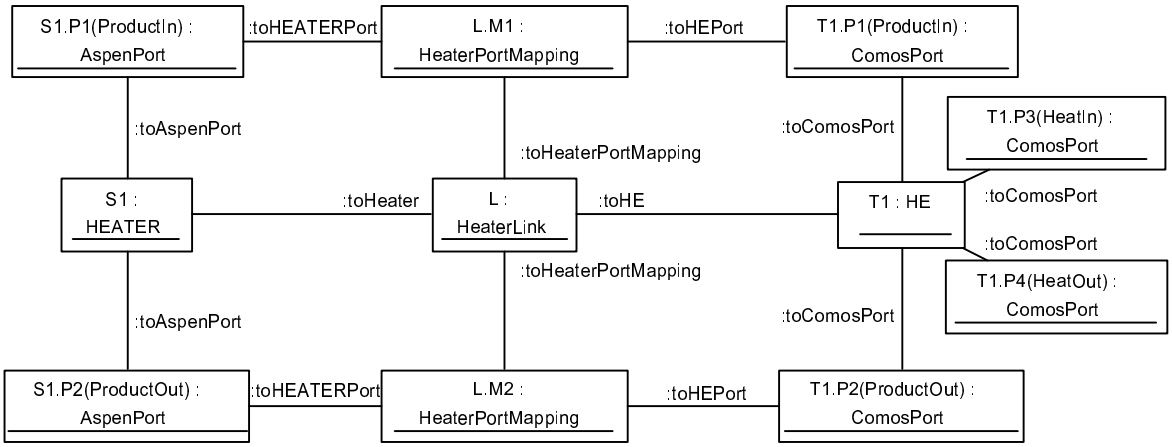

Fig. 7. Correspondence between heaters based on the association in Figure 6 a

have no counterpart in Aspen Plus. The meaning of this correspondence is that a HEATER component in an Aspen Plus simulation model can be related to a HE component in a Comos PT flow sheet through a link in an integration document. This is the case in the example in Section II. The integration rules that can be gained from this correspondence through refinement will be presented later.

To ensure the validity of the correspondences, they have to be instances of associations. If a correspondence is modeled by hand following the top down approach, the modeler should select the link type from the parent association he intends to use. Then it can be ensured by the modeling environment (e. g. Rational Rose) that the correspondence is consistent to this association. In this example, the type HeaterLink was chosen from the association in Figure 6.a.

If the correspondence is gained from an existing integration document following the bottom up approach, it has to be matched against all available associations. The correspondence explained so far could be an instance of all associations in Figure 6. All associations can be ordered following a heuristic metric that is based on the cardinalities of the UML associations and the distance of the types used to the leaves of the documents' type hierarchies. The less abstract an association is, the higher is its value according to the metric. Following this metric, the association in Figure 6 a) would be assigned a high value while the association in Figure 6 c) would get a low value. In general, the higher the sum of the values of all matching associations is for a correspondence, the stronger it is backed by the domain knowledge contained in the model on type level. The correspondence's type should be set to the association with the highest value.

A more complex correspondence that is an instance of the association in Figure 6 c) only is depicted in Figure 8. Here, a complex pattern in an Aspen Plus simulation model is related to a simple pattern in Comos PT. The reaction performed in the reactor in Comos PT (T1) is too complex to be simulated in only one reactor component in Aspen Plus. Therefore, two reactors (S1, S5) are connected with a stream (S3) via the appropriate ports and connections. Only the ports that are used to connect the complex structure to other components (S1.P1, S5.P2) are mapped (PM1, PM2) to the corresponding ports in the flow sheet (T1.P1, T1.P2). One of the rules that can be gained from this correspondence is used in the example in Section II to create the simulation blocks in Aspen Plus for the PFR in Comos PT (cf. Figure 1).

B.2 Rules

Correspondences rather precisely specify which patterns of source and target documents can be related to each other. Nevertheless, information is still missing to obtain rules that can be executed by an integration tool. Therefore, each correspondence can be enriched to gain several executable integration rules. This is done with the help of constraints that are added to the UML collaboration diagrams describing the correspondences. The constraints are used to extend the corresponding patterns into simple graph rewriting rules that are used to find a given situation in the target, source and integration document and alter the three documents according to the rule. For a more detailed description of the rule execution process please refer to [5].

There are two classes of rules: Rules that establish links and rules that deal with inconsistencies of existing links that occur after a modification of previously integrated documents.

The following constraints are defined:

- Unmarked increments have to be present in the document.
- {not}: The marked increment must not exist.
- {new}: The marked increment must not exist and is created if the rest of the pattern was matched.
- {delete}: The marked increment must exist and is deleted if the rest of the pattern was matched.

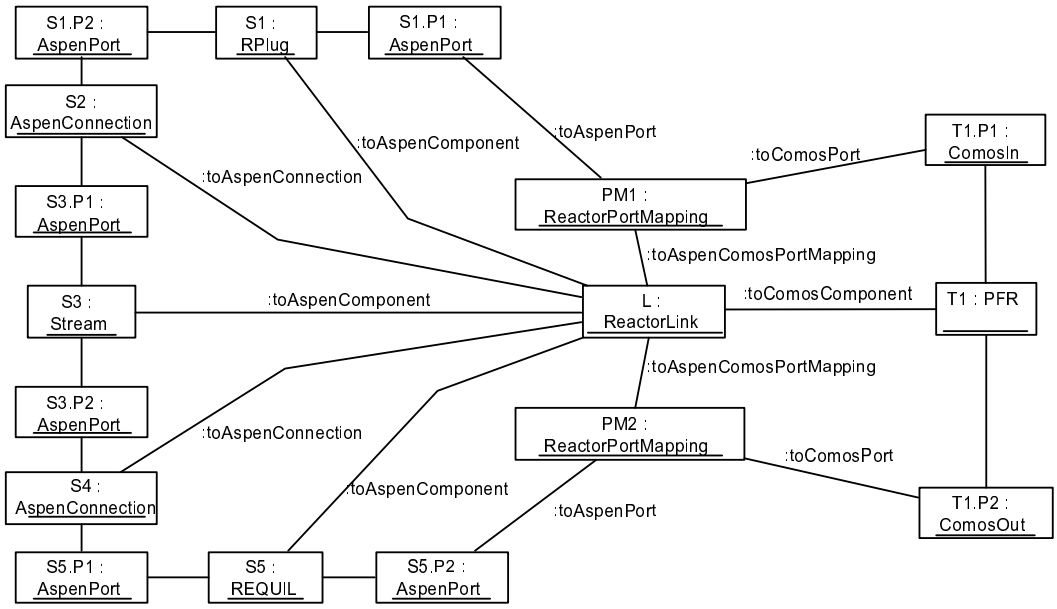

Fig. 8. Complex correspondence

• Additional constraints can be defined, e. g. to restrict the search to increments that have specific attribute values.

Some of the rules that can be gained by enriching the correspondence in Figure 7 are illustrated in Figure 9. The ports and their mapping are omitted for brevity. The rules a) and b) are link establishing rules, c) and d) deal with inconsistencies. Rule a) is a forward transformation, i. e. information is propagated from the source to the target document. Here, if an increment of type HEATER is found in the simulation model, a corresponding increment of type HE is created in the flow sheet, and a new link is created in the integration document that references both increments. Rule b) performs the same operation in the opposite direction; this rule was applied in the example in Section II. Please note, that despite the destinction between source and target document both forward and backward transformations can be performed by one integration tool in one integration cycle. This is needed if both documents were modified simultaneously and are to be made consistent by propagating the changes made in each document into the other.

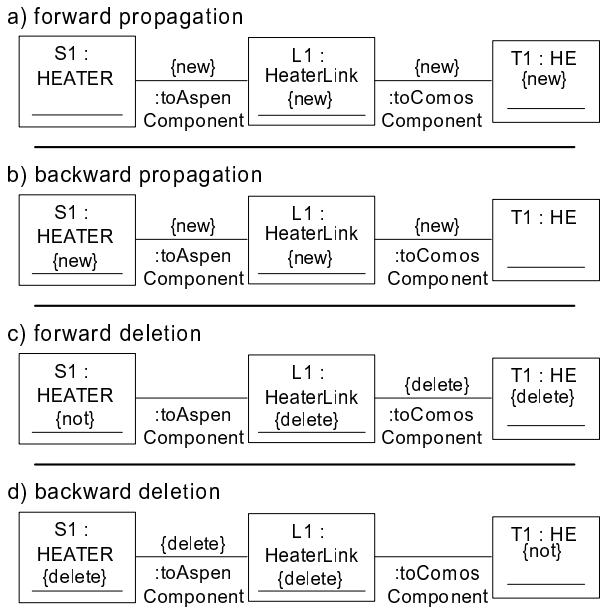

Fig. 9. Link establishing (a, b) and inconsistency resolving (c, d) rules

The inconsistence resolving rules (c, d) are executed similarly. Here, rules are depicted that deal with restoring the consistency after the deletion of an increment that was referenced by a link previously. For instance, if a HeaterLink was established by rule a) or b), and the heater component in the simulation model was deleted by the user, rule c) is executed with the result that both the link and the heater component in Aspen Plus are deleted.

Of course, there can be more complex rules than the ones explained so far. For instance, from the correspondence in Figure 6 c) several rules can be derived through the usage of the constraints defined above.

Another example for a complex rule is the one used to propagate connections between the documents (Figure 10). This rule is rather important because it is needed to reconnect patterns that were created in a document by other rules, according to the topological structure of the document where they originated. It can be used if port mappings are restricted to cardinality one-to-one. If two AspenPorts (SP1, SP2) in the source documment are connected (S1), the ports in the target document (TP1, TP2) mapped to them are connected as well (T1) and a link between the two connections is created (L1).

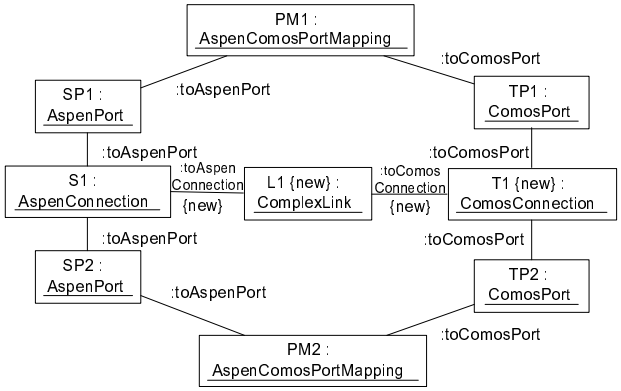

Fig. 10. Forward transformation rule for connections (1:1-port mapping)

B.3 Attribute Assignment

So far, only the structural aspects of correspondences and rules where adressed. In practice, each component is further defined by a large number of attributes and their values. To deal with the consistency of these attributes, each correspondence can be enriched by different attribute assignment statements. Attribute assignments can be expressed using the OCL language. In our prototypic implementation Visual Basic Script is used to spare the translation of OCL to an executable secification. An attribute assignment can access all attributes of the increments referenced by a link; those of context increments can only be read, the others can be written as well. There are different situations in development processes in which an integration is performed. Depending on the situation an appropriate attribute assignment is chosen. For instance, for each correspondence (i.e., for the set of resulting rules) there is one attribute assignment for the initial generation of the simulation model, one to propagate the simulation results back into the flow sheet, etc.

C. Concrete Instance Level

On the concrete instance level instances of source, target, and integration documents can be modeled mirroring the structure of 'real' documents. Here, the constructs introduced on type level can be used like on the abstract instance level. Concrete instances of links found in actual integration documents can easily be abstracted to correspondences on abstract instance level. This is done by simply replacing the identifiers of all objects by placeholders.

VI. Related Work

At our department incremental integration tools were built for different areas of application, at first for the domain of software engineering [14]. Recently, in our project being a part of the Collaborative Research Council IMPROVE (CRC 476) the domain of chemical engineering has been the main focus [8], [5], [3]. Our approach is based on graph transformations and coupled graph grammars [20].

We use UML [7] to express the multi-layered integration model. Therefore, extensions to the UML meta model have to be made [18], [6], [1].

In another project of the CRC 476 the data model CLiP for chemical engineering is developed which consists of several partial models [4]. This data model is integrated in our document models and the relations between the partial models defined in CLiP are used as a basis for the top-down rule definition.

In current practice in chemical engineering [21], the dependencies between documents are often managed by hand. This task is error-prone and time consuming. There are different approaches to improve tool support: The definition and usage of standardized data models, the usage of central databases or document management systems where all product information is stored, and batch converters that generate a target document from a source document. While batch transformators lack user interaction and incremental operation, the other approaches only deal with coarse grained integration or rely on one-to-one relations between the documents' contents.

There are different research areas in computer sciences that deal with the problem of fine grained dependencies between documents, mainly with software engineering as domain of application: Traceability, model transformation, and inconsistency management.

Traceability is applied to track the requirements of a software system in products of later phases of the development process [17]. The main interest is to provide documentation of dependencies that often have to be defined manually but not to give tool support for keeping dependent documents consistent.

Model transformation deals with consistent translations between heterogeneous models. For instance, this is of high importance for software development methods like model driven architecture [11]. There are a lot of projects where graph grammars are used to specify the translation [12], [2]. In [13] UML is used. Most resulting translations between models operate batch-like without user interaction and do not support incrementality.

In the area of inconsistency management it is dealt with the detection of inconsistencies between existing documents [19], [16]. Methods are developed to interactively or automatically resolve such inconsistencies.

In [10] a framework is proposed that identifies different view points of a product that is to be developed and provides the basis for their integration. One application of this framework with emphasis on the integration between the different view points is presented in [9]. It is based on distributed graph grammars. This approach is similar to our work but it focuses on the consistency check and the resolving of inconsistencies of existing documents. Operational rules are defined as graph transformations from scratch, there is no prior definition of domain knowledge.

Xlinkit [15] is another project dealing with dependent

documents. XML technology is used to assure the documents' consistency. Because of the structure of the documents in our domain, we believe that UML and graph grammars are better suited to model and execute integration functionality.

VII. Conclusion

We have reported on recent work on a framework for building incremental integration tools. The framework has been developed in close cooperation with an industrial partner (innotec, which develops and markets COMOS PT) under the umbrella of a long-term German research project (the Collaborative Research Council IMPROVE) which is concerned with design processes in chemical engineering. We have demonstrated how the UML may be used to define integration models for incremental integration tools. For creating models, we make use of a commercial CASE tool (Rational Rose). Current work is concerned with analyzing UML models for consistency and with generating code for the integration rules. Future work will address generalization of the framework such that it can handle other domains as well (so far, there are some domain-specific parts which assume PFD documents).

Acknowledgements

This work was partially funded by the CRC 476 of the Deutsche Forschungsgemeinschaft (DFG).

References

[1] C. Atkinson, T. Kühne: Processes and Products in a Multi-Level Metamodeling Architecture, in: *Intl. Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 6, pp. 761–783, World Scientific, 2001.

[2] L. Baresi, M. Mauri, M. Pezzè: PLCTools: Graph Transformation Meets PLC Design, in: Proc. Workshop on Graph-Based Tools (GraBaTs 2002), *Electronic Notes in Theoretical Computer Science*, Vol. 72, No. 2, 11 pp., 2002.

[3] B. Bayer, S. Becker, M. Nagl: Model- and Rule-Based Integration Tools for Supporting Incremental Change Processes in Chemical Engineering, *Proc. 8th Intl. Symposium on Process Systems Engineering*, accepted for publication, 2003.

[4] B. Bayer, C. Krobb, W. Marquardt: A data model for design data in chemical engineering – information models, Techn. Rep. LPT-2001-15, Lehrst. f. Prozesstechnik, RWTH Aachen, 2001.

[5] S. Becker, T. Haase, B. Westfechtel, J. Wilhelms: Integration Tools Supporting Cooperative Development Processes in Chemical Engineering, *Proc. Intl. Conf. Integrated Design and Process Technology (IDPT-2002)*, 10 pp., SDPS, 2002.

[6] S. Berner, M. Glinz, S. Joos: A Classification of Stereotypes for Object-Oriented Modeling Languages in: France, R. and Rumpe, B.: *Proceedings UML '99 — The Unified Modeling Language*, LNCS 1723, pp. 249–264, Springer, 1999.

[7] G. Booch, I. Jacobson, J. Rumbaugh: *The Unified Modeling Language User Guide*, Adison Wesley, 1999.

[8] K. Cremer, S. Gruner, M. Nagl: Graph Transformation based Integration Tools: Application to Chemical Process Engineering, in: Ehrig, H. et al. (Eds.): *Handbook of Graph Grammars and Computing by Graph Transformation*, Vol. 2, pp. 369–394, World Scientific Publisher, 1999.

[9] B. Enders, T. Heverhagen, M. Goedicke, P. Tröpfner, R. Tracht: Towards an Integration of Different Specification Methods by Using the Viewpoint Framework, *Transactions of the SDPS*, Vol. 6, No. 2, pp 1–23, 2002.

[10] A. Finkelstein, J. Kramer, M. Goedicke: ViewPoint Oriented Software Development, in: *Proc. of 3rd Intl. Workshop on Software Engineering and its Applications*, pp. 337–351, 1990.

[11] A. Gerber, M. Lawley, K. Raymond, J. Stell, A. Wood: Transformation: The Missing Link of MDA, in: *Proc. of 1st Int. Conf. on Graph Transformations (ICGT 2002)*, LNCS 2505, pp. 90–105, Springer, 2002.

[12] J. de Lara, H. Vangheluwe: Computer Aided Multi-Paradigm Modelling to Process Petri-Nets and Statecharts, in: *Proc. of 1st Int. Conf. on Graph Transformations (ICGT 2002)*, LNCS 2505, pp. 239–253, Springer, 2002.

[13] D. Milicev: Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments, in: *IEEE TOSE*, Vol. 28, No. 4, pp. 413–430, 2002.

[14] M. Nagl: *Building Tightly Integrated Software Development Environments: The IPSEN Approach.*, LNCS 1170, 709 pp., Springer, 1996.

[15] C. Nentwich, W. Emmerich, A. Finkelstein: Static Consistency Checking for Distributed Specifications, in: *Proc. of Intl. Conf. on Automated Software Engineering (ASE 2001)*, pp. 115–124, IEEE CS Press, 2001.

[16] T. Olsson, J. Grundy: Supporting Traceability and Inconsistency Management between Software Artifacts, in: *Proc. of Software Engineering and Applications (SEA 2002)*, 6 pp., ACTA Press, 2002.

[17] B. Ramesh, M. Jarke: Toward reference models for requirements traceability, in: *IEEE Transactions on Software Engineering*, Vol. 27, No. 1, pp. 58–93, 2001.

[18] A. Schleicher, B. Westfechtel: Beyond Stereotyping: Metamodeling Approaches for the UML, *Hawaii International Conference on System Sciences HICSS-34*, Minitrack "Unified Modeling Language: A Critical Review and Suggested Future", 10 pp., 2001.

[19] G. Spanoudakis, A. Zisman: Inconsistency Management in Software Engineering: Survey and Open Research Issues, in: Chang, S.K. (Ed.): *Handbook of Softw. Eng. and Knowledge Eng.*, Vol. 1, pp. 329–380, World Scientific, 2001.

[20] A. Schürr: Specification of Graph Translators with Triple Graph Grammars, in: Tinhofer, G. (Ed.): *Proc. WG '94 Intl. Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 903, pp. 151–163, Springer, 1994.

[21] VDI: *Engineering in der Prozessindustrie* (in German), VDI-Berichte 1648, VDI, 2002.

[22] innotec GmbH. http://www.innotec.de, 2002.

[23] Aspen Technology. http://www.aspentech.com, 2002.