Integration Tools for Supporting Incremental Modifications within Design Processes in Chemical Engineering[*]

Birgit Bayer[a], Simon Becker[b], Manfred Nagl[b]

[a]Process Systems Engineering, Turmstr. 46
[b]Computer Science III, Ahornstr. 55
Aachen University of Technology, D-52056 Aachen, Germany

Keeping the vast amount of inter-document relations consistent, which relate entities of different documents, is important for the quality and efficiency of design processes. Especially, these relations are eminent for master documents, as flowsheets (PFD) in chemical engineering, because management decisions are based on statements derived from PFDs. This paper introduces novel tools for an incremental and interactive update or check of such relations.

1. INTRODUCTION

In the process industries, there is a growing demand for the improvement of *design processes* in order to obtain better design results in shorter development cycles. The design results are of major interest for the top *management* of a chemical company, as they are the basis for *decisisons* if, when, and how to build a new plant or to modify an existing one.

Current *tool support* is mainly characterized by numerous software tools for *specific purposes* or *isolated parts* of the design process. However, a sustainable improvement of the design process can only be achieved by the integration of application tools into a design environment [1]. During the last years, commercial environments like Aspen Zyqad or intergraph's SmartPlant have been developed. They are mainly restricted to the tools of the corresponding vendor. The adaption to specific work processes of developers within a company or the integration of arbitrary tools, especially from other vendors, are not generally solved.

One key aspect of tool integration is the *integration* of the *data* created and used within different tools. These data are kept in separate heterogeneous documents, stored in independent files or databases. Nevertheless, their contents have to be consistent to each other. E. g., the process structure in a simulation model used to describe and predict the behaviour of a chemical process has to be consistent to the structure of the process represented in a flowsheet. Therefore, elements like reactors or streams in one document are related to corresponding elements in the other. This results in many *fine-grained, inter-document dependencies*, which have to be managed and which are the focus of this paper. Furthermore, coarse-grained data integration (version and configuration control) has to be managed too [2], which is not presented here.

Within design processes, especially if concurrent and simultaneous engineering are applied, *incremental change processes* are performed, where changes have to be *propagated* precisely through the network of dependent documents. Therefore, tools are needed to support developers in keeping different documents in a consistent state. Currently, these fine-grained dependencies are managed manually by developers without appropriate tool support.

[*]Financial support by Deutsche Forschungsgemeinschaft for Collaborative Research Center 476 IMPROVE.

In this paper, *tools* will be presented that *model* the *dependencies* between different documents types and documents explicitly. On this basis, integration tools are derived supporting developers within change processes and for consistency management. Therefore, integration tools also integrate those tools by which source and target documents are elaborated. These integration tools are one of four novel approaches investigated within the Collaborative Research Center IMPROVE [3, 4], to support design processes in chemical engineering on top of existing design tools (a-posteriori integration).

Integration *tools* can only be built, if there is *structure* within corresponding documents. Fortunately, this is often the case, e. g. between a flowsheet and a simulation model. The flowsheet plays the role of a master document in chemical engineering [5]. These tools have to work *incrementally* to serve for change processes, they consider design decisions *interactively*, as there are different possibilities to relate elements of different documents to each other, and they are responsible for *managing* the fine-grained *dependencies* between different documents. Integration tools have been built in the domain of software development for some time [6].

For the *realization* of integration tools a *methodology* has been developed. It consists of a top-down part, which is described in this paper, and a bottom-up part to build wrappers on top of existing tools, to get homogeneous interfaces. The methodology consists of three steps: within the information model CLiP the different types of entities of the chemical engineering domain are ordered into partial models and clearly described in a conceptual multi-level framework (sect. 2). Dependencies of this description are further refined (sect. 3): Firstly, the potential dependencies are modeled on type level. Then, patterns of object sturctures together with their links are defined and tool behaviour is specified, e. g. for a forward transformation. Finally, integration tools are realized a component framework. A short outlook concludes the paper.

2. INFORMATION MODELS FOR SPECIFYING INTEGRATION TOOLS

In order to be able to implement integration tools that allow to interrelate data of different application tools, the *data* handled and *stored* in these *tools* and their mutual *dependencies* have to be *understood*. One possibility to obtain such an understanding are information models, in which data structures and dependencies are described in an abstract and formal manner.

The *information model CLiP* (*C*onceptual *Li*fecycle *P*rocess model) has been developed to describe the information and work processes of the chemical engineering domain in a conceptual manner, i. e. independently from a specific implementation [7]. CLiP covers *three* model *levels*, on which concepts are given on different degrees of abstraction.

On the *meta level*, the `technical system` is introduced representing all kinds of technical artifacts, that are built to fulfill some functionality (cf. Fig. 1.a). Examples for `technical systems` are chemical plants, computer systems, or mathematical models. Within a `technical system`, two types of subsystems can be distinguished: `devices` which hold the major functionality and `connections` that link the different devices together. The connectivity of `devices` and `connections` is represented by their `ports`, which are connected via `couplings`.

On the *type level*, CLiP is refined for the representation of information needed during chemical process design by introducing *types* (classes) of *entities* (cf. Fig. 1.b). The chemical process itself and mathematical models used to describe the process' behavior are of major importance here. Both are instances of `technical system`; their main *elements* and *relations* are given in Fig. 1.b. `Process steps` like separations, reactions, or unit operations are the devices *of*

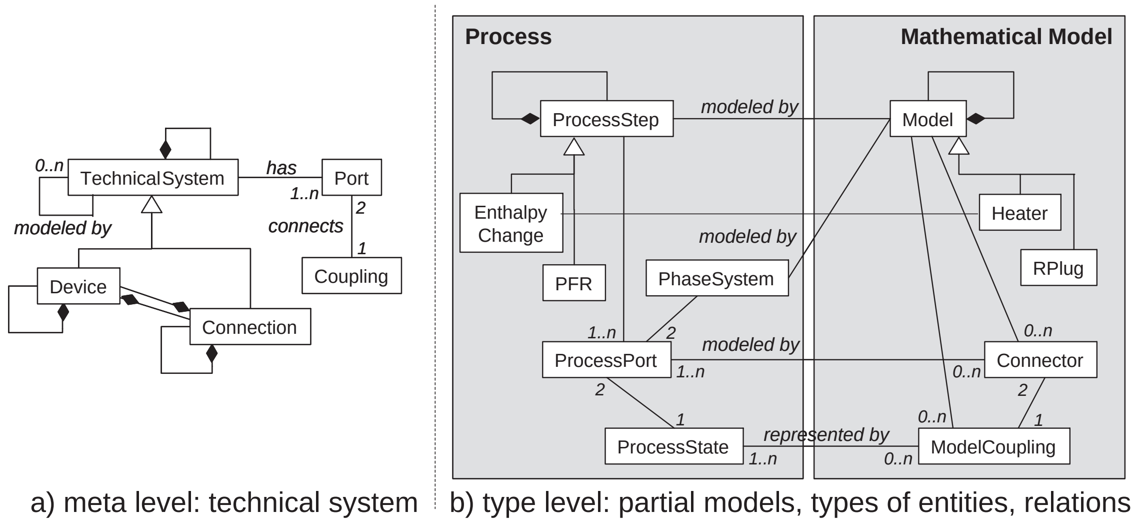

Figure 1. Meta and type level of the CLiP information model

the *chemical process*. Their connections are `phase systems` representing an amount of material at a certain `process state` which is exchanged between two process steps via their `process ports`. A `phase system` can be a material stream with a distinct flow rate (e. g. kmol/hr) as well as a material amount or hold-up with a distinct amount (e. g. kmol). The `process state` can characterize the exchanged material in an unambiguous manner.

Mathematical `models`, as part of the information model's type level, can be developed with an internal structure and specified inputs and outputs that represent the *structure* of the *modeled system*. `Models` have `connectors`, which can be used for connecting two models with another via `model couplings`. The left side of Fig. 1.b belongs to the *partial model* `Process` the right side to `Mathematical Model`.

From a conceptual point of view, `process steps` and `phase systems` can be related to the `models` which are used for their representation (see Fig. 1.b), in this case defining *dependencies between entities* of two different partial models. In principle, the relationship is many-to-many, because a process step might be represented by an arbitrary number of models, or one model can be used to represent different (similar) process steps. Furthermore, for the representation of a `process step` within one specific `model` more than one model building block might be needed, e. g. for the description of different phenomena occurring in parallel.

The structure of the chemical process given by the connectivity of the `process ports` is reflected on the model side by `connectors`. `Process steps` and `models` can be decomposed in a hierarchical manner; the structure of these decompositions are interdependent. So, both sides of Fig. 1.b can be refined by giving *specific types* of the entities for a specific domain within chemical engineering via specialization of basic entity types. For example, `PFR` is a specific `process step`. Thereby, the *dependencies* shown in Fig. 1.b are instantiated more *specifically*.

The type level of the information model defines the basic *knowledge* of an application *domain*. It is grouped into partial models and defines the entity types in a specialization hierarchy. This hierarchy forms an ontology of this domain. The relationships of this model are the basis for the development of an integration tool between Comos PT [8], a design data base where information about the chemical process and the single process steps is kept, and Aspen Plus [9], whose model building blocks are special types of mathematical models.

3. INCREMENTAL INTEGRATION TOOLS

Integration tools manage the fine-grained relationships between structured documents containing entities, the types of which are described in CLiP. The relationships are used to incrementally propagate changes between documents, to check such relations etc. Integration tools are used to detect these relationships and to perform the changes, checks, etc. In this section we describe the *methodology part below* of *CLiP* to realize such tools. As example, we take a tool that manages the consistency of process flow diagrams (PFD) in Comos PT and simulation models in Aspen Plus.

For simple cases, the *relations* between classes defined *on type level* of CLiP can be directly transformed into integration rules. E. g., the `EnthalpyChange` process step is related to the `Heater` model (cf. Fig. 1.b). A rule that can be derived from this relation can informally be described as follows: "If a step `EnthalpyChange` is contained in the PFD, insert a `heater` in the simulation model and memorize the dependency between both."

For each such relation there can be a *forward rule*, which transforms the PFD to a simulation model, a *backward* rule for the opposite direction, a *consistency checking* rule, etc. The dependencies are stored in an additional document, that is called integration document.

In general, *relations* have to be defined in terms of *corresponding object patterns*. As example, a correspondence can be defined between a pattern in Comos PT and a pattern in Aspen Plus, see Fig. 2.a. The pattern in Comos PT consists of a reactor (`PFR`) with its input and output ports. This pattern is related via a `link` to a pattern in the simulation model which consists of two reactors (`RPlug` and `REquil`), connected by a stream via appropriate ports. This can be necessary in some situations if the reaction in the PFR is too complex to be modeled by only one reactor in Aspen Plus. Again, this relation can be transformed to *different rules*. In Fig. 2.a, a forward rule is depicted. The applications of the UML-constraint {`new`} indicate that if a `PFR` is found in the PFD, the corresponding simulation model structure is created and the dependency is stored in the integration document.

Each structural entity like a stream or a chemical device is described in detail by a large set of attributes. These *attributes* have to be kept *consistent* as well. Therefore, each rule can be enriched with pieces of scripting code that contain attribute assignments. At rule enactment, the user can choose which piece of code to apply.

Fig. 2.b shows an excerpt from a sample *integration document* of the Comos PT-Aspen Plus integrator. The dependencies are stored as links referencing the dependent entities of the corresponding documents. The application of the rule depicted in Fig. 2.a results in the link `L3`. The `PFR` contained in the rule was mapped to the `PFR` in the sample PFD, the corresponding structure of the simulation model and the link in the integration document were created. Other rules applied transformed the other entities in the same way.

Furthermore, the transformed entities of the simulation model have to be connected to eachother as well. Therefore, a special rule is defined to transform the couplings from Comos PT to Aspen Plus. For instance, the link `L4` in the sample integration document results from the execution of that rule. To determine the source port of the new coupling `t2`, via the component `s1` the tool can find the corresponding link `L1` which contains a mapping of the ports of `s1` to the ports of `t1` (target port analog). This results in dependencies of the link `L4` to link `L1` and

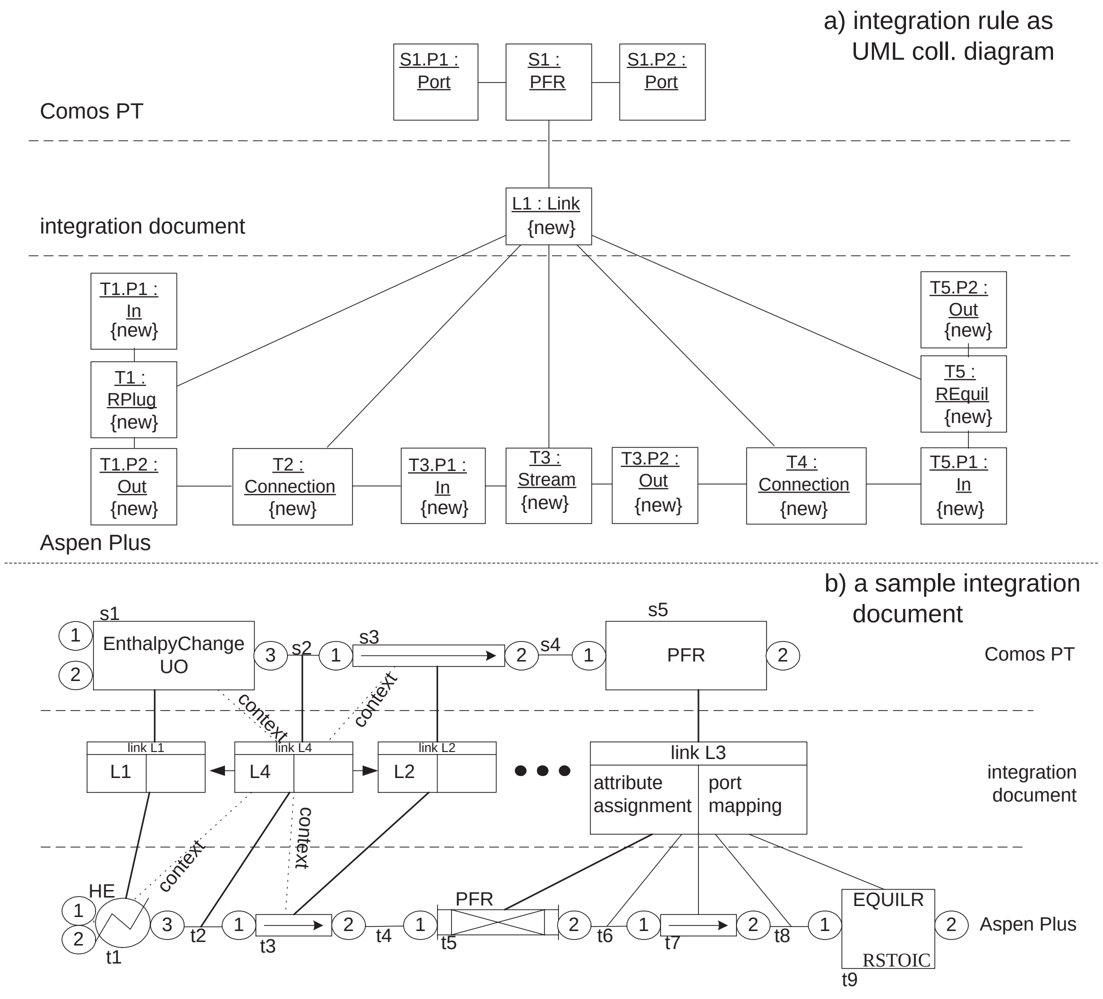

Figure 2. Integration rule and integration document

`L2` (arrows in Fig. 2.b). The components that can be read but not be modified by the rule are referenced as context of the resulting link.

Integration *tools work as follows*: In each run, the integrator first checks the links contained in the integration document, which is empty in the first run. A link is in a consistent state if all referenced entities are still available and unchanged. Then, the documents are searched for new entity patterns which can be matched against existing rules. If more than one rule can be applied alternatively, the user is asked for his decision. If the available rules do not fit the user's needs, links can be edited manually. When manually editing links, a user may detect that an entity correspondence is a general object pattern, which can be enriched to form rules as described above.

The described Comos PT-Aspen Plus *integrator* [10] has been realized. The implementation uses a multi-layered framework of reusable software components for integration tools. Beside these general components, an integrator consists of some specific components, as e. g. tool wrappers to easily access the tools and their data. Additionally, the specified integration rules are used to define the integrator's behaviour. The rules can either be interpreted at runtime (which is important for rules defined on the fly), or translated into program code and compiled as part of the integrator. The latter is used for efficient implementation of predefined rules that will not change.

4. CONCLUSION AND OUTLOOK

In this paper *integration tools* were introduced which especially support designers for handling change processes and consistency management, and which are of major importance for quality and efficiency of design processes and also for implied management decisions. A *methodology* was presented for the *realization* of such tools ranging from the definition of entities and dependencies within an information model to the implementation of such tools using reuse techniques. As example, a Comos PT-Aspen Plus integrator was explained.

From the information model side one step is still missing: Entities of different types within an application domain belong to/occur within different forms of documents. The specification of these documents, together with the definition of their internal structure should be a part of the model. From the tool building side two steps are still missing: For defining the dependencies of different documents a rule editor is currently under development. Furthermore, important in the a-posteriori context, semantical wrappers are developed in order to equal the interfaces of tools to be integrated w. r. t. their semantical levels. The methodology for integrators is applied at various places of the scenario of IMPROVE. Thereby, also simpler forms of integration tools are developed.

REFERENCES

[1] B. Beßling, B. Lohe, H. Schoenmakers, et al. Cape in process design – potential and limitations. Computers & Chemical Engineering, 21(Suppl.), pp. 17–21, 1997.

[2] M. Nagl, R. Schneider, B. Westfechtel. Tool support for the management of design processes in chemical engineering, to appear in Computers & Chemical Engineering.

[3] W. Marquardt, M. Nagl. Tool integration via interface standardization? In DECHEMA Monographie. 36. Tutzing Symposium der DECHEMA e.V. "Informationsverarbeitung in der Prozeß- und Anlagentechnik", pp. 95–126, Wiley-VCH, 1999.

[4] M. Nagl, W. Marquardt. Tool integration via cooperation functionality. In ECCE 3, 3rd European Congress of Chemical Engineering, paper 6–5. DECHEMA, 2001.

[5] B. Bayer, K. Weidenhaupt, M. Jarke, W. Marquardt. A flowsheet centered architecture for conceptual design. In R. Gani, S.-B. Jørgensen (eds.) Europ. Symp. on Computer Aided Process Engineering 11, pp. 345–350, Elsevier, 2001.

[6] M. Nagl (ed.). Building Tightly Integrated Software Development Environments: The IPSEN Approach. LNCS 1170, 709 pp., Springer, 1996.

[7] 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.

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

[9] Aspen Technology. http://www.aspentech.com/brochures/aspenplus.pdf, 2002.

[10] S. Becker, T. Haase, B. Westfechtel, J. Wilhelms. Integration Tools Supporting Cooperative Development Processes in Chemical Engineering. In Proc. of the 6th Conference on Integrated Design & Process Technology. 10 pp., 2002.