# Requirements Engineering for Control Systems Development in Small and Medium-Sized Enterprises

Dominik Schmitz, Hans W. Nissen,
Matthias Jarke, Thomas Rose
Fraunhofer FIT, Schloss Birlinghoven
53754 Sankt Augustin, Germany
{schmitz, jarke, rose}@fit.fraunhofer.de,
hans.nissen@fh-koeln.de

Peter Drews, Frank J. Hesseler
RWTH Aachen University
Institut für Regelungstechnik
52056 Aachen, Germany
{p.drews, f.hesseler}@irt.rwth-aachen.de

Michael Reke
VEMAC GmbH & Co. KG, Krantzstr. 7, 52070 Aachen, Germany
reke@vemac.de

## Abstract

*Since nowadays more and more control systems are realised within software on electronic control units, a conceptual integration of control systems engineering and software engineering must be aimed at. Within this work, we build on a recent proposal to use the software requirements formalism i* to enable a combined investigation of control systems' and software requirements. While i*'s modelling means have turned out to be sufficiently expressive, two characteristics of control systems still need to be addressed: firstly, how to incorporate domain knowledge especially about the system to be controlled in the requirements development process and secondly, how to specifically support small and medium-sized companies (SMEs) that are the main driver for innovations in this domain. Due to their innovativeness and flexibility, the SMEs usually follow a project-oriented customer-specific development approach. To be nonetheless cost-effective, especially during the offer development phase, we develop a mechanism and a tool to compare a current requirements model with requirements models of control systems from earlier projects. Altogether this reduces time and increases reliability in regard to the identification of reusable software artefacts.*

## 1. Introduction

In modern automobiles control systems play an important role. The task of a controller is to continuously compare and adapt the current value(s) of some system to some possibly changing desired value(s) [10]. For example, when the driver of a car specifies a desired velocity via the position of the accelerator, the engine controller ultimately computes the appropriate amount of fuel as well as the best point in time for injection and ignition. Control system functionality is nowadays implemented in software on electronic control units (ECUs) [1].

In the control systems development sector, requirements engineering takes place within the offer development process. Typically, a major car company contacts suppliers for an offer to develop a control system for a new engine. The supplier specifies the control system requirements based on the characteristics of the new engine and on further functional and non-functional requirements. Furthermore, the supplier needs to develop a system design already in this early phase in order to calculate the development costs.

Some characteristics of the control systems' domain let the requirements engineering task become a very specific, difficult, and interesting problem. Firstly, control system development is dominated by small and medium-sized enterprises (SMEs) from the engineering sector. Major success factors for them are their flexibility, innovativeness, and customer orientation. Accordingly, an SME will accumulate a lot of specific experiences and knowledge in a particular controller domain. In most cases their customer-oriented development process focuses on the control functions and the applied software engineering practices are quite basic but pragmatic. Secondly, the offer must be developed within a short time frame dictated by the ordering customer. Thus, support for a fast requirements capture that takes into account the SMEs knowledge must be provided. Furthermore, the SME must reliably identify reusable components from earlier projects while not being able to build

on a dedicated product family approach. If by accident a reusable component is not considered, the offer gets more expensive than necessary. On the other hand, if later design phases discover that the selected components are in fact unreusable, their new development may result in a project loss. Thirdly, control systems are realised in software but base on physical features of the controlled system. Its development therefore requires the *combined* investigation of software and control requirements. Furthermore, a detailed knowledge of the characteristics and interrelations of the different parts of the controlled system are essential for an accurate control software. Fourthly, there is a high frequency of innovations in this field, thus the knowledge changes and grows quite fast. With each new development project, new engine components and new construction styles may arise. Finally, although all engines are somehow similar on an abstract level, they greatly differ in detail. Resultingly, always a new control system is needed. In turn, this spoils long-term product planning due to the individuality of the developed solutions. Consequently, there is rarely a chance for planned product families.

After providing a brief introduction to *i** (Sect. 2), this paper investigates the two challenges, rapid requirements modelling and reliable identification of reusable components already within the offer development phase (Sect. 3). By creating a generic domain model for the example of engine control systems, we allow for a fast specification of requirements. Furthermore, the domain model enables requirements comparison as basis for the fast and reliable identification of reusable components. The paper ends with revisiting related work (Sect. 4), conclusions, and an outlook on future work (Sect. 5).

## 2. Requirements Modelling with *i**

*i** [16] is an agent- and goal-oriented, semi-formal modelling framework targeting the requirements engineering domain. By using few and simple modelling elements (agents and goals), it is intended to serve as a common language between customers and software developers: sufficiently expressive to capture all the details, the software developers need to know, but still comprehensible for non-IT-knowledgable customers. In [15], the authors successfully explored the possibility to capture control system requirements with *i**. We will briefly reconsider this approach here and thereby give a short introduction to the *i** framework.

**Strategic Dependency (SD) Diagram** In *i**, two levels of modelling are distinguished. On the higher level, the modeller can capture the various stakeholders (*actors*) and their dependencies within the so called *strategic dependency (SD) diagram*. Within controller development, the "controlled system" as well as the "controller" are the two main stakeholders. Furthermore, their combination, the

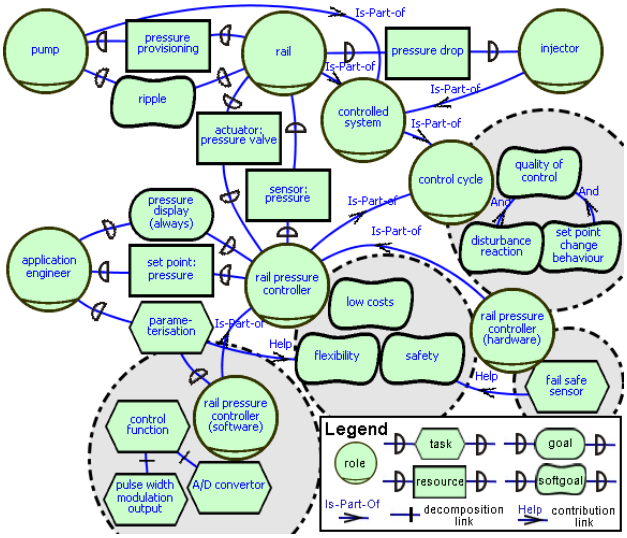


**Figure 1. Combined SD and SR Diagram for a Common Rail Injection System**

"control cycle" needs to be considered an actor of its own since some properties, e. g. stability, can only be assigned to this combination. In Fig. 1 a control cycle regarding the pressure in a common rail injection system is modelled. The controlled system is split up into three parts (using *is-part-of* links): the distributor "pump", the "rail" volume itself, and the "injector". Other stakeholders can be added, for example, the "application engineer" has requirements regarding simple means to adjust the parameters of an engine to a concrete setting.

*Dependencies* are used to capture the relationships between the modelled stakeholders. In a *task dependency* the depender specifies the details of the requested service while in a *goal dependency* it is left to the dependee how to provide a requested state. Similarly, with a *resource dependency* the dependee only has to provide the resource. A *softgoal dependency* is similar to a goal dependency but there are no clear-cut criteria when this goal is fulfilled. Instead a dependee can only make contributions. Thus, softgoals are suited to capture non-functional requirements (quality goals). In control systems engineering, all these types of dependencies apply as well. Especially, *actuators* and *sensors* are simply captured by corresponding resource dependencies. Figure 1 shows that the rail depends on the pump for pressure provisioning and that the rail is affected by the injector due to the drops in pressure that it causes. The actuator "pressure valve" indicates that the rail depends on the controller via this valve. Correspondingly, the controller depends on the rail via the "pressure sensor".

**Strategic Rationale (SR) Diagram** The individual goals and processes of stakeholders and systems as well as their relation to external dependencies are captured in the more

detailed *strategic rationale (SR) diagrams*. Regarding the modelling, the types of links (task, goal, resource, and softgoal dependencies) simply become modelling elements on this level. Additionally, the SR diagram provides new types of links to detail out a complex task (*decomposition*), to model alternatives to achieve goals (*means-ends*), or to specify qualitative contributions towards softgoals (*help, make, break, etc. contribution*). Figure 1 also shows SR details of some stakeholders involved. For example, the most important softgoals that are to be considered for the rail pressure controller are "low costs", "flexibility", and "safety". The use of "fail safe sensors" helps achieving a suitable level of "safety" while the "parameterization" enables "flexibility".

In contrast to a simple action diagram [10], the *i** model as introduced above is thus able to capture not only the functional interdependencies of controller and controlled system but also non-functional issues as well as various additional stakeholders within a single model. This enables reuse since the development of a new controller can build on an existing model. For example, the customer might do without a sophisticated parameterization since the application field of the controller is much more constrained. This in turn enables a simpler, maybe purely hardware based controller solution with much lower costs. Documenting such considerations via contribution links allows to consider very different solution ideas and relate them to the main goals and softgoals of the stakeholders. Thereby, *i** allows to explore reusability and scalability of controllers in different contexts systematically.

## 3. Supporting Offer Development

The following short scenario from our industrial practice illustrates the offer development process at SMEs. The process is typically initiated by a major car company contacting a number of suppliers to provide an offer to develop a control system for a new engine. Usually, the time frame for the supplier to respond is very short. The car company delivers a specification of the engine and the list of required control functionality. To prepare the offer, the supplier first specifies the requirements on the requested control system from a developers point of view. In a second step he prepares a system design in order to calculate the costs for the development of the control system. To keep the development costs low and to win the contract he must reuse as many software artefacts as possible from his already developed control systems. We will now focus on our support for the two main activities within this process, the requirements specification and the identification of reusable artefacts.

**Fast Requirements Capture** Since SMEs usually are experts in some particular sub domain of control systems engineering, such as, for example, engine controllers, we expect this knowledge to be captured in form of a domain model. Figure 2 shows the domain model that our interdisciplinary team of control engineers and computer scientists has developed for the combustion engine sub domain. It gives an overall, mainly SD level overview with a special focus on the components of the controlled system. More SR details had to be omitted here but are part of the complete model. The core part of the controlled system in this field is the "combustion engine block". Here many detailed characteristics like the type of engine (boxer, row, V), numbers of cylinders, type of fuel, etc. are captured. Depending on their importance, some sub components are explicitly represented as actors on their own. For example, "common rail" and "camshaft" are usually considered to be part of the engine block (see *is-part-of* links). But since they have important sensors and actuators of their own, separating them out into extra actors is required for a sufficiently detailed representation and improves thereby the practicability and readability of the model. Next to details of the so called "air path", also the "customer" (or user) that interacts with the engine via the "accelerator" is represented. This allows to capture even their preferences in regard to "comfort" or "sportivity".

While it would have been possible to build several separate models, especially for example, for a gasoline versus a diesel engine, a single good-enough model that combines all potential characteristics has evolved. This is due to the insight that despite the conceptual differences between gasoline and diesel engines regarding, for example, the need for ignition, very often innovations are shared at least conceptually between these two technologies. Thus for simplicity in usage, it is much easier to start with a more than complete model and simply eliminate the parts that are not applicable to the current setting. Furthermore, this elimination procedure is supported by *i** modelling means, for example, the modelling editor OME [9], in that connected parts can easily be automatically eliminated recursively. For our example model, if only "diesel" remains as the alternative for "fuel", also "ignition" components (even in the "ECU") can be automatically discarded from the model.

To reiterate, the domain model is suited to serve as a common starting point for a requirements model for all projects thereby accelerating the modelling process. By eliminating the parts of the model that do not apply for the current project and by adding new elements that are specific to it, rapidly a basic model of the controlled system can be established. Furthermore, the consistency of naming core components across several projects can be ensured. This becomes important when a comparison with requirements models from earlier projects is aimed at (see below). And finally, a concrete SME can specialize the domain model in regard to the particular methods, tools, experiences, and alternatives that are current at their site. Thereby, the model

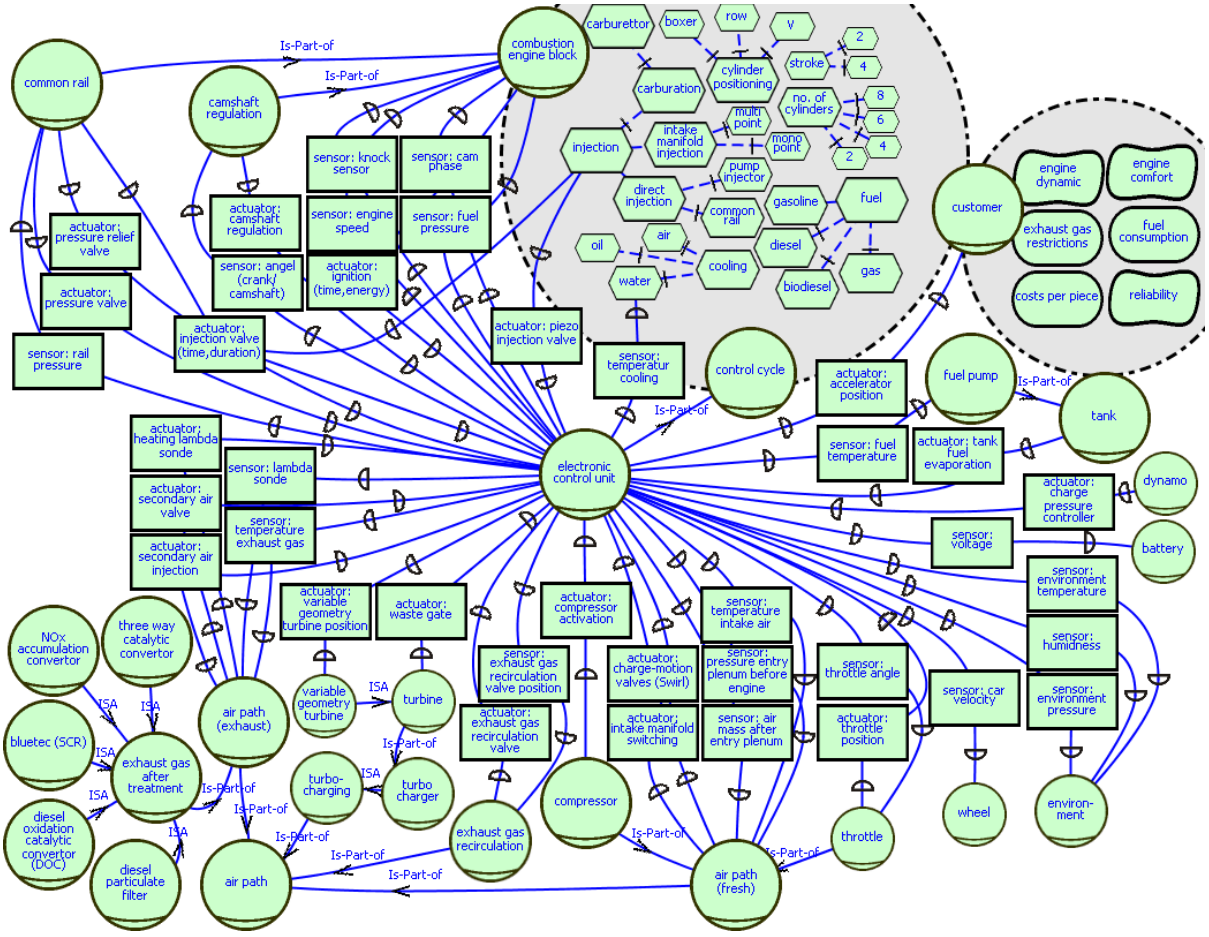


**Figure 2. Generic Domain Model for Engine Control Systems (focus on controlled system)**

also serves to manage enterprise-specific experiences and knowledge. For example, most likely a tailoring towards particular sensors and actuators will take place.

Next to sub domain knowledge, we have already pointed out in [15] that *i** is also suited to capture knowledge about the qualification of control algorithms for control problems and the incorporation of knowledge about the electronic control unit platform (or rapid control prototyping platform) that is available or limiting features thereof. Problem-solution pairs can easily be captured by means of goal dependencies (SD level) or on SR level via means-ends connections between the problem (modelled as a goal) and the potential solution (modelled as a (complex) task). Softgoal contributions on SR level can be used to indicate levels of suitability (or inadequacy). Similarly platform features can also be modelled to constrain the set of potential alternatives that are to be considered. This is important since severe restrictions are imposed on the controller by the available hardware and electronic control unit platform mainly due to the fact that the automotive industry is very cost sensitive. This influences, for example, the performance and memory available on the electronic control unit.

**Identification of Reusable Software Artefacts** The supplier must prepare a first system design already within the short offer preparation phase to calculate the costs for the whole development project. As stated above, an important task is the identification of reusable software artefacts from finalised control system development projects. Within the VEMAC company and many other companies this is done manually by a senior engineer who knows the characteristics of the finalised development projects. He selects projects most similar to the new request, inspects these in detail, and includes reusable artefacts into the system design.

The main challenge is to avoid the following situations: Firstly, existing reusable artefacts are not found due to the huge set of already finalised projects. Since the time frame for offer development is short, the engineer is not able to investigate all projects. The offer will include the re-development of these existing artefacts and therefore will become more expensive than necessary; the contract might be given to another supplier. Secondly, the offer bases on existing artefacts which the subsequent detailed design phase identifies to be actually not reusable. The neces-

sary development of new artefacts leads to more work than planned. The calculation of this unreliable offer becomes inappropriate and causes a loss within that project.

A fully automated identification of reusable software artefacts is not possible. Always a senior engineer is needed to do the technical inspection and to decide if an existing software artefact suits the new control system. But we are able to automate the identification of finalised projects containing potentially reusable artefacts. This reduces significantly the number of finalised projects the engineer has to inspect. And therefore increases the possibility that all reusable artefacts are actually found.

Our identification of *similar* projects bases on the domain model described before. We assume that the requirements of all finalised projects and the requirements of the new project are specified using that modelling approach. The requirements models therefore partly base on the domain model and partly contain project-specific extensions. Intuitively speaking, we call two projects similar, if there exist indications that parts of the control system software of one project could be reused for the other. An indication is given, if parts of the requirements models of the projects match.

For this comparison task, we employ the deductive object manager ConceptBase [4] that implements the knowledge representation language Telos [14]. The selection is based on pre-defined and ad-hoc comparison rules. The pre-defined comparison rules focus on the pre-defined domain model. The ad-hoc comparison rules can be specified by the user (i. e. the engineer) and focus on project-specific parts of the requirements model. The possibility to specify project-specific rules is very important because of the high frequency of innovations and the product diversity within our application domain. For the implementation of these rules we make use of ConceptBase's query class concept: a query is represented as a class where the instances form the answers to the query. The query itself is stated in form of a constraint of the class such that all objects of the object base fulfilling this constraint become (virtual) instances. The following example computes the kind of fuel the combustion engine uses by identifying all *task* elements that are parented by "combustion engine block" and are *or-decomposed* from the "fuel" task element.

```
QueryClass EngineFuel
   isA IStarTaskElement with
  constraint
    c : $exists i/IStarRoleElement
         (this parent i) and
         (i name "combustion engine block")
         and exists j/IStarTaskElement
         (j name "fuel") and
         exists l/IStarOrDecompositionLink
         (l from j) and (l to this)$
end
```

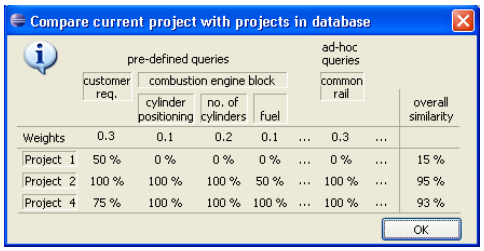


**Figure 3. The Output of a Comparison**

Other rules identify, for example, the functions of the controller. Each pre-defined comparison rule focuses on one aspect of the combustion engine control system. In the same way we allow the specification of ad-hoc comparison rules again using the query class concept.

It is quite common that the similarity in some aspects of the requirements model is more important than the similarity in others. Therefore, the user can associate a weighting factor with all comparison rules, the pre-defined ones and the ad-hoc ones. A high weighting factor indicates a high importance. The aggregation of the weighted answers of all queries results in an overall weighting for each project. The projects containing similarities within the highly weighted areas of the new project's requirements model are then ranked higher.

**Implementation Overview** The invocation and coordination of the query classes is realised by a Java-client embedded in the OME [9]/Eclipse environment that we use for modelling and that is connected to the ConceptBase server. The client evaluates the query classes representing comparison rules firstly within the requirements model of the new project, then within the models of the finalised projects. Finally, the weighting factor of a query is applied and the result is presented via the client's user interface. An example is shown in Fig. 3.

## 4. Related Work

Incorporating domain knowledge has been a major topic in requirements engineering for many years now. Maiden and Sutcliffe [13] promoted the development of so called domain specific languages that allow the customers to specify their intentions more easily but still enable a translation of the resulting document in a more formal representation that is suited as input for software engineers. Due to the high dynamic of domain models, they focused not on particular details of a domain but on generic relationships still allowing to give precise semantics to them. Similarly, the AutoRAID project [3] has developed a domain specific language for the automotive domain but mainly concerns functional aspects. Konrad and Cheng [7] investigate the use of patterns on requirements modelling level, also targeting the embedded system domain. Their multi-faceted patterns

focus on domain-specific generic information such as the representation of actuator-sensor information in UML models. Also similarities between requirements have been considered for quite a long time. For example, Maiden and Sutcliffe [12] have seen the analogy paradigm as a powerful means to exploit specification reuse. In [11] dedicated principles are given that a computational model of similarity should adhere to in order to fit with human processing. In more recent work by Kaiya et al. [5], comparison of use case diagrams of existing systems is considered as a means to identify stakeholders and their preferences about non-functional requirements. But the comparison step is left to be manually accomplished by the investigator. Also product line approaches enable requirements reuse. In [2] the authors even integrate this research direction with agent-oriented modelling. The main problem here is that product line approaches explicitly require the identification of commonalities and variabilities. As Knauber et al. [6] already pointed out, this is highly problematic for SMEs due to their flexibility and customer-orientation. Within the control system domain, this becomes even more problematic as presented here. The latter is also supported by much older work by Lam et al. [8]. They proposed "ten steps towards systematic requirements reuse" for the domain of aero-engine control systems and identified several steps to be "nonconformist", i. e. to be deviating from usual software development. Out of them, the following three have been addressed here: "requirements patterns often emerge after working in a particular domain", "make explicit the context of reuse to prevent misuse", and "parts of the requirements engineering process is also reusable". One difference is that nowadays these steps are supported by models.

## 5. Conclusions and Future Work

We have investigated two challenges in the requirements engineering phase of an integrated approach to control system and software development. A real world domain model for the combustion engine sub domain has been developed using the modelling language *i**. It alleviates the requirements modelling, ensures a consistent naming, and is capable of integrating specific knowledge of a particular team. We developed a tool that compares a new requirements model against already finalised projects. A user-adaptable similarity measure helps the system engineer to identify a small number of projects that need an in-depth investigation regarding the reuse of software artefacts. This helps creating a low-priced and reliable offer.

For future work we will evaluate the suitability of the approach using the project partner VEMAC as a pilot user. We are interested in seeing how the domain model is adapted to their specific setting and how much experience it is able to incorporate. Furthermore, we expect more sophisticated similarity measures to arise. The evolution of the domain model is a challenging field of its own. We expect schema evolution techniques to be required in order to ensure that new projects remain comparable to old ones.

**Acknowledgment.** This work was supported by the BMBF in the project ZAMOMO (01 IS E04). Thanks to the coordinator Stefan Kowalewski, RWTH Aachen University.

## References

[1] M. Broy. Challenges in automotive software engineering. In *Int. Conf. on Software Engineering*, pages 33–42, Shanghai, China, 2006. ACM.

[2] J. Dehlinger and R. R. Lutz. A product-line approach to promote asset reuse in multi-agent systems. In *Software Engineering for Multi-Agent Systems IV (SELMAS)*, LNCS 3914, pages 161–178. Springer, 2006.

[3] E. Geisberger and B. Schätz. Modellbasierte Anforderungsanalyse mit AutoRAID. *Infor. Forsch. Entw.*, 2007.

[4] M. Jarke, S. Eherer, R. Gallersdörfer, M. A. Jeusfeld, and M. Staudt. ConceptBase - a deductive object base for meta data management. *Journal of Intelligent Information Systems*, 4(2):167–192, 1995.

[5] H. Kaiya, A. Osada, and K. Kaijiri. Identifying stakeholders and their preferences about NFR by comparing use case diagrams of several existing systems. In *12th Requirements Engineering Conf.*, pages 112–121, Kyoto, Japan, 2004. IEEE.

[6] P. Knauber, D. Muthig, K. Schmid, and T. Widen. Applying product line concepts in small and medium-sized companies. *IEEE Software*, 17(5), 2000.

[7] S. Konrad and B. H. C. Cheng. Requirements patterns for embedded systems. In *10th Requirements Engineering Conf.*, pages 127–136, Essen, Germany, 2002. IEEE.

[8] W. Lam, J. A. McDermid, and A. Vickers. Ten steps towards systematic requirements reuse. *Requir. Eng.*, 2(2):102–113, 1997.

[9] L. Liu and E. Yu. Organization Modeling Environment OME. http://www.cs.toronto.edu/km/ome, [June 10 2008].

[10] J. Lunze. *Automatisierungstechnik*. Oldenbourg, 2003.

[11] M. Jarke et al. Requirements engineering: An integrated view of representation, process, and domain. In *4th Eur. Software Engineering Conf. (ESEC)*, LNCS 717, pages 100–114, Garmisch-Partenkirchen, Germany, 1993. Springer.

[12] N. A. M. Maiden and A. G. Sutcliffe. Exploiting reusable specifications through analogy. *CACM*, 35(4):55–64, 1992.

[13] N. A. M. Maiden and A. G. Sutcliffe. Requirements engineering by example: an empirical study. In *1st Symposium on Requirements Engineering*, pages 104–111. IEEE, 1993.

[14] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos - representing knowledge about information systems. *ACM Trans. on Information Systems*, 8(4):325–362, 1990.

[15] D. Schmitz, P. Drews, F. J. Hesseler, M. Jarke, S. Kowalewski, J. Palczynski, A. Polzer, M. Reke, and T. Rose. Modellbasierte Anforderungserfassung für softwarebasierte Regelungen. In *Software Engineering*, LNI P-121, pages 257–271, Munich, Germany, 2008.

[16] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, 1995.