Tool-Guided Domain-Specific, Systematic Requirements Management

Rainer Gerlich, Ralf Gerlich

BSSE System and Software Engineering, Auf dem Ruhbuehl 181,
88090 Immenstaad, Germany, Phone +49/7545/91.12.58, Mobile +49/171/80.20.659, +49/178/76.06.129
Fax +49/7545/91.12.40, e-mail:Rainer.Gerlich@bsse.biz, Ralf.Gerlich@bsse.biz URL: http://www.bsse.biz

ABSTRACT:

The importance of the quality of requirements for successful execution and completion of a project from a technical and contractual point of view is being recognized more and more. Many methods are targeted to improve the support for collecting requirements while still focusing on natural language. However, the ambiguities in the semantics of natural language are the biggest obstacles towards success. The approach presented in this paper focuses on the elements of a domain while keeping the expressiveness of natural names and terms and introducing clear semantics. This brings the advantage that immediate verification of the human-provided inputs is possible, immediate contributions to validation are available and inconsistencies can be detected by a tool immediately. This leads to guidance of an engineer by a tool towards consistent, complete and correct requirements – requirements of high quality – and eases maintenance for the same reasons. As most of the complexity is handled by the tool due to its good knowledge on the domain, the approach is scalable towards large specifications. Several examples of application domains are described which illustrate the universality and feasibility of the approach across domain boundaries.

1 INTRODUCTION

Historically, requirements were expressed in natural language. In the perspective of the need of higher quality and larger systems, the process of requirements elicitation and collection was improved – while still relying on natural language.

The HOOD group [1] proposes a project-specific knowledge database for syntactic parts of the natural language which shall enforce convergence of terms and give hints on problem solution. R. Melchisedech [2] applies rules to elements of natural language upon which a tool can perform verification. The method and tool from the SOPHIST group [3] introduce rules which limit the syntax of natural language and shall avoid typical problems in understanding and interpreting text.

DOORS [4] is a tool which manages textual requirements provided in natural language. Support is given to the generation of documentation, management of dependencies between requirements, collaboration of different teams at multiple sites. Polarion [5] provides a similar, but web-based solution with extended capabilities regarding administration of textual requirements. Requisite Pro [6] and CaliberRM™ are also tools for management of requirements with similar capabilities.

All these tools have overlapping and complementary capabilities regarding requirements expressed in natural language.

In order to overcome above issues of natural language, some methods try to extend the degree of formality, though replicating the approach of natural language while aiming towards "universality" as main goal.

UML [8], the Unified Modelling Language, is also applied to requirements management. In this case requirements are expressed as UML models. To some degree elements of UML language replace natural language. But major parts may be still expressed in natural language, e.g. as notes or documentation text – and this is common practice. The Object Constraint Language (being part of UML) applies higher order logic to express formal constraints and thereby inherits all the usability and decidability issues from this class of logic languages.

SysML [9], the Systems Modelling Language, addresses the specification of systems rather than software. Mostly derived from UML, it contains new constructs for modelling constraints and their interdependencies. Still, proper support for formal specification of non-functional requirements is missing, so that such requirements are mostly described using natural language. Dependencies and relations between the requirements again have to be managed and maintained manually, leading to possible inconsistency issues as the specification evolves.

Simple text processing software such as Microsoft™ Word® also needs to be mentioned in the context of collecting requirements. Here an engineer can – or is required to – define own organisation schemes.

Structural markup and semantic preloading [10] can be used to give plain text a structure allowing at least some extraction of formal information. Still, the markup needs to be manually inserted and maintained, which is not a trivial task. It therefore can also deviate and even distract from the actual formal content of the marked plain text.

From a principal point of view classical methods of requirements engineering focusing on natural language can be divided into constructive and analytical methods. Constructive methods constrain an engineer when forming the textual requirements. Analytical methods analyze textual requirements on conformance with given rules. Hence, analytical methods bring in a first attempt of verification and quality metrics.

Common to all above methods and tools is that they rely on natural language and – thereby – support a large scope of application domains. Latter intention is fully in line with UML aiming to support the full spectrum of application domains.

However, the universality of these approaches introduces or preserves problems of classical extraction, computability and decidability, not allowing efficient or even complete verification or derivation of consequences. Universality therefore is in conflict with verification goals – from the perspective of scalability and automated quality assessments.

Also, these methods still need a lot of human intervention, which leads to poor efficiency, low scalability and high costs, which are specifically visible when systems are becoming larger and more complex. From the perspective of verification and validation such classical tools do not contribute much.

The method described in this paper combines the capabilities of constructive and analytical approaches while fully supporting verification based on easily computable metrics and – in consequence – automated quality assessments which can be performed by an automaton itself. This ensures scalability up to rather large systems and high efficiency of the engineering process including maintenance as well.

2 REQUIREMENTS MANAGEMENT

2.1 Definition

In our understanding "Requirements Management" (RM) is the discipline which implies

- requirements engineering (RE) covering
  - structuring,
  - analysis,
  - collection / elicitation,
  - verification and validation of requirements
- administration of requirements
  - tracing
  - linking
- organisation of the cooperation of involved engineers by support of collaboration
  - multi-team capability
  - multi-site capability
- contractual management including
  - assignment of requirements to contractual entities,
  - tracing and control within and across contractual boundaries,
  - action item tracking.

2.2 Status

All of the methods and tools mentioned above mainly concentrate on aspects of administration and organisation. Collection of requirements is supported, only, by provision of containers for text. For all other features of RM, the engineer is left alone, especially for the quality aspects. Though the quality of requirements is – or should be – an important matter of RM, such tools badly support this aspect. It is left to an engineer to define appropriate methods on top of such a tool, though in most cases it is hard to do in practice.

Once a set of requirements has been established, it has to be maintained. This requires capabilities for monitoring changes and efficient means for repeated quality assessments.

Conventional tools well support tracing of changes per requirement. However, as linking of requirements is a manual task, links may become invalid. This has consequences on consistency and on efficiency. Every invalid link needs to be detected and updated manually to obtain consistency.

Moreover, as the contents of a requirement – natural text – cannot be checked by a tool, conflicting requirements can only be detected by manual inspection.

Several tools support collaboration. However, it is questionable whether full visibility from everybody on every requirement is really required. From an organisational point of view, especially regarding responsibilities, access should be limited depending on the role of a team member. Hence, global visibility is not a must, but dedicated access rights on higher-order elements of a specification is recommended as being more efficient due to reduction of interaction possibilities not really needed.

Rarely supported by tools is contractual management, an exception is e.g. the Polarion tool.

3 SYSTEMATIC REQUIREMENTS MANAGEMENT

The method for Systematic Requirements Engineering (SRM) is based on the following major objectives:

1. to focus on a certain domain,
2. to find and apply rules which can be checked mechanically.

Due to objective 1, background information is available which can be used to establish the rules satisfying the needs of automation. This also has positive consequences on maintenance: any change can immediately be analysed and verified.

The applied rules contribute in twofold manner to requirements engineering (RE):

- they guide an engineer during the construction or update of requirements, and
- they allow to analyse the provided requirements by an automaton.

Hence, the approach combines constructive and analytical issues and allows to assign the issues of quality assessment to a tool. Administration of requirements could be fully automated, contractual management can be supported and linked with technical requirements. Collaboration is a matter of organisation of the considered domain into proper entities.

An important general issue is the support of the full application domain chosen and of the freedom to introduce natural words and terms – while introducing a formal approach based on a meta-model.

3.1 Universality or Specialisation?

Due its importance for the quality assessment and the efficiency of the approach, the issue of specialisation is explained in more detail.

3.1.1 Natural Languages

Natural language is most easily used for requirements description, as it is pre-established and quite expressive as it is a matter of daily life.

Its main disadvantages are, however, its ambiguity and – partially a consequence thereof – its inaccessibility to mechanical extraction of information. The ambiguity is present both in syntactical and semantic structure – pronouns, ambiguous references – but also in the different meanings of words dependent on context or reader's background information.

Already flexion of words, plural and singular, irregular verbs, and other basic lexical characteristics of natural language introduce nearly insurmountable problems for parsing and lexical analysis.

Even though there are what in modern modelling terminology would be called universal meta-meta-models of natural language[11] establishing a formal theory of the logical meaning of language, no concrete and at the same time universal instantiation of such a meta-meta-model for a given language is known to the authors.

Experiments in the area of computer game development have shown that semi-natural formalised language can be used to describe facts and relationships used to generate interactive text adventures[12]. However, the complexity of such analysis tools indicates that the effort is probably not affordable, not for a specialised and certainly not for a universal approach.

The main advantage against structured models, as used in UML, SysML, AADL [13] and other formal languages, would be the ease of use due to the natural feeling, a specification language based on text has. Engineers would be able to use the specification form seemingly without training.

In order to achieve such a natural feel, highly sophisticated parsers need to be developed. Further, the presumption on the lack of necessity of training does not hold in practice. Even though the language feels natural, there still is a computer system behind which formally processes the language. Without an understanding of the formal meaning of the language and without proper self-discipline in the use and introduction of new terms, the quality of specifications cannot be raised, similar to the need for glossaries in other natural-language-based specification methods.

All methods based on natural language imply universality regarding the supported application domain: everything can – unsurprisingly – be well expressed in natural language. The problem of extracting what is expressed in a formal way, however, remains unsolved.

In fact, what was considered above as a major advantage of natural specification languages – no need to learn the language – is more than compensated in practice by the big disadvantage of needing much additional support and training on how to apply the language for a specification in order to achieve high quality.

3.1.2 Modelling Languages

There are some modelling languages, such as UML or SysML, which target formalisation and universality at the same time.

At first sight universality of a formalised language bears the advantage that a single language and toolset can support all possible application domains, leading to the expectation of lower investment for tools and training, as well as implicit standardisation of the industry.

In contrast, domain-specific languages are expected to be costly in design and maintenance of both the language itself and the toolset required to support it, and to lead to fragmentation, as every supplier is expected to have its own language.

Looking deeper, neither is the case. In order to enable a tool to analyse, verify, simulate or support to validate a model or a set of requirements, i.e. to well support quality issues, all the necessary information must

1. be automatically extractable from the model,
2. be available in such a way that correctness is decidable and relevant consequences are computable with algorithms of low runtime and implementation complexity, and
3. actually be used by a tool.

Therefore the information must be either explicitly present in the model or accessible by static analysis, which in turn is based on the semantics as defined by the meta-model.

A universal meta-model provides no specific information about any application area, so the information about the domain must be made explicit by the engineer. This increases the workload, the size and complexity of the model – as seen by the engineer – and the probability for inconsistencies or incompleteness in areas which are not governed by the meta-model and therefore cannot be found or hinted at by the tool. In summary, the level of abstraction – as representative for an application domain – is not raised, as the specification either has to be spelt out in detail, not much different from programming, or remains incomplete.

Often, profiling is considered an appropriate solution for this dilemma.

But even then, the tool is not prepared for the relevant issues of the application domain and must be adapted, leading to additional development and maintenance effort and costs.

Further, for universal languages like UML or SysML – though superior to natural language regarding formality – training costs and effort are rather high. The specifications of either language are sized at several hundreds of pages explaining complex structures and their interdependencies – a consequence of the demand for universality. For the largest part these semantics are not formal but rather presented in plain text, oriented around known formal concepts but not completely adopting them [14].

The size and complexity are consequences of the need for universality. The requirements of all relevant application domains need to be met, even if these needs are merely overlapping.

Reasonable means to simplify the use of a universal language is tailoring or customizing. However, still for each application all the language elements need to be analysed and understood before a decision can be made which aspects need to be taken care of and which can be ignored for the relevant domain.

This adds to the cost and effort required to devise and train the methods for expressing the needs of the target domain in a universal language framework not designed for such a task. Consequently, domain experts are forced to learn a “foreign” language to express their needs, increasing the overall complexity.

All this does not even take into account that the UML is known to be incomplete[14] and inconsistent [15], both intentionally – in the so-called Semantic Variability Points – and by accident. Therefore the underlying theory of correct systems is neither sound nor complete, introducing an insurmountable barrier for verification or validation by derivation of consequences (“ex contradictione quodlibet”).

Most of these inconsistencies are non-trivial and require detailed insight and research into the specification and its consequences. However, if ignored, they can lead to serious misinterpretations of the specification. Additionally, some of the semantics – such as “run-to-completion”-semantics or “zero execution time” – are difficult to implement or far from reality, thereby masking actual issues of a specification and risks inherent in or introduced for the further phases of a project based on such a specification.

The tool support for UML2 in most cases does not even reach OMG compliance Level 0 [16], the lowest level of support, and no currently known tool supports the UML2 completely. Selecting a tool supporting the portions relevant for a given project or domain is a complex task. The level of support for verification by the available tools is not known – as this feature still does not seem to be of major interest, but can be assumed to be insufficient both due to the universality of the language and practical experience [17]. The lack of support can mostly be attributed to the complexity and size of the specification, for which full support is not affordable.

In contrast, a domain-specific language does not have to contain elements which are not needed in the domain. Further, the domain experts already know the elements of their domain and will easily recognize them in the language, especially when the language is designed to meet the culture of the addressed engineers. Consequently, design, documentation and training costs can be much lower than for universal languages respectively profiles, and the complexity of such a specialised language does not even by far reach that of a universal language.

While being less complex in total, the meta-model of a domain-specific language may implicitly contain a large amount of additional information about the domain itself, which can be used by tools for verification,

simulation and support for validation, e.g. by automatic construction of different views on the specification and visualisation of hidden information, implicitly included but needed for validation. Due to the specialisation, previously non-decidable verification problems can become decidable and non-trivially computable issues can become much easier to compute.

In addition, today there is a vast set of tools available as Open Source Software specifically for designing and implementing domain-specific languages, ranging from the Eclipse Modelling Facility (EMF) via the Graph Editing Framework (GEF), the Graph Modelling Framework (GMF) or transformation utilities such as ATL to parser generators such as ANTLR and code generation engines such as Java Emitter Templates (JET), and many more.

After all, the decision for a domain-specific language does not mean that users have to bear the costs each on their own or that fragmentation of an industry ensues. The similarities between suppliers in the same domain can lead to a common domain-specific effort just the same as it can lead to the design of common UML or SysML profiles.

3.2 The Benefits of an SRM Meta-Model

A meta-model for Systematic Requirements Management (SRM) is based on the rules and characteristic elements as identified for the chosen domain from what engineers are using. In consequence, such characteristic elements are represented in first-class types of the meta-model, and the rules become statically or dynamically checked constraints on the elements of a model.

3.2.1 Organisation

An important consequence – especially regarding efficiency – is that the rules are inherently relating such types and all instances of these types automatically to each other – where the instances are introduced by an engineer in the context of requirements elicitation. This saves a lot of effort compared to classical approaches, where such dependencies need to be established and maintained manually for every new element and for each update.

The problem of constraining the allowed syntax, semantics and terms as known from use of natural language is solved by introducing a cleanly structured form of input, not involving any natural language at all, as well as specialised element types. Each such type may have attributes, some of which are optional, others are mandatory to be specified, and yet others may be optional depending on the context established by the ruleset.

Dictionaries and databases – to be maintained manually – are also no longer needed, because the meta-model allows the tool to manage and check the requirements – silently if no error is reported, and noisily if an error is introduced thereby guiding the engineer. The tool itself can establish and maintain the database in accordance with the inputs from an engineer, based on the information on relations from the meta-model.

An engineer is not constrained regarding the terms (s)he wants to apply, but (s)he is constrained regarding the rules which ensure correctness, completeness and consistency. Metrics on the quality can be defined based on the rules and their potential violation. This way an engineer is guided to requirements of high quality while not being constrained in size and complexity.

In practice, it was observed that flagging errors due to violation of such rules enforces engineers to think about what they did. In many cases their attention was drawn to weaknesses of their inputs which cannot be detected by any formal method as this is a matter of requirements validation.

For these cases detection of weaknesses in formal parts lead to an improvement of the quality in areas not assessable by a tool. In fact, this is compliant with the well-known rule: “when there is one error detected, in most cases there will be more errors”. Hence, when increasing the number of detected errors by a formal approach, the chance to detect more errors is very high.

3.2.2 User Interface

The meta-model represents a unique, unambiguous description of the functionality of a domain. In consequence, the requested information is not constrained to a certain format. As an SRM meta-model is a formal model, any input notation compliant with the meta-model can be used. To allow an engineer to express the requirements as instances of the elements types, usually templates are provided. Templates may relate several types with each other or request the attributes of a an instance of a type.

This brings a big advantage compared to the classical approaches. They urge a user to apply a format suitable for the tool, but possibly – in most cases clearly – not suitable for the user: (s)he needs to learn the notation of the tool which implies increased effort and a risk of misunderstanding what is specified.

Quite differently, an SRM tool may adapt to the user’s world at little effort. Only the templates need to be changed when adapting, and a transformation of the specific notation to the standard one is required.

The essential advantage is: not the engineer has to transform ideas from his/her world into the (internal) notation of the tool manually, but the tool does it. This

saves a lot of effort, decreases significantly the complexity, the risk to fail and the time-to-completion.

3.3 SRM Tools

For every intended application domain a meta-model and a tool-chain need to be established. This looks like big effort, but it is not in practice. As the ordering principle of a meta-model is similar for different application domains, major parts of the software can be reused when the next meta-model needs to be implemented in a tool-chain.

On a technical level, the advent of Eclipse and its modelling tools – most notably EMF and ATL – have lead to a standardisation of meta-model data structure and interchange formats.

The code for implementation of the data model, serialisation and notification are provided nearly for free after the data structures have been specified. Constraints can be declared in the data model itself and stubs for verification procedures are provided automatically, neatly integrating the verification process in a known environment. Reflection on the data model structure allows easy integration with transformation and code generation utilities like ATL and JET.

All of these mechanism remain mainly transparent to any application using the data model, thereby allowing a bus of applications working on an interwoven and integrated set of models, providing a consistent tool-chain, possibly supplied by different vendors or industry members. Specialisation towards the needs of an individual supplier can be as simple as adding another application to the bus.

4 APPLICATIONS

The approach has been applied to several application domains and will be applied to more in near future. Some examples are described below.

4.1 Domain of Communicating Systems and Processes: System Operations

As domain of "communicating systems and processes" the large domain of distributed systems is understood covering e.g. the subdomains of real-time systems, client-server systems, embedded systems, IT processing systems. The focal point for modelling is consideration of system operations.

4.1.1 Modelling Elements

The principal elements needed to express requirements in this domain are: processes, processors, communication channels and protocols, messages, states, resource utilization, user roles, actions, reliability, availability, safety and security constraints, etc.

In this domain the main part of requirements can be grouped around activities which can be expressed as Input $\Rightarrow$ Processing $\Rightarrow$ Output, which is the (general) IPO principle. It is a generic approach which explains that a wide spectrum of subdomains can be supported.

Above elements allow an engineer to spawn every system from this domain. The tool guides the engineer in which pieces of information need to be provided and gives a feedback on the achieved quality. Due to the meta-model the tool can conclude on completeness, consistency and correctness and report on any non-conformance.

4.1.2 Provided Information

According to the users' world and criteria templates were expressed in spreadsheets which also well support the needs of collaboration.

Templates are provided for

- the activities, allowing to express a sequence of activities, either at the whole or in parts, possibly spread over several sheets and sub-sheets (tables),
- data, allowing to express basic or structured data and the related types and numerical representations,
- access rights and roles,
- the other types listed above and their attributes,
- the requested deliverables like documents and their providers,
- applicable and reference entities.

4.1.3 Reporting

The inputs are checked on conformance with the rules and reports are generated.

Errors are reported according to the severity level. For each error an explanation and the location in the input sheets is given. Also, an error is marked in red in the sheet and the explanation is displayed when the cursor is in the input field.

Different views on the system are generated – in detail and in summary, as tables, text or graphics – such as (non-exhaustive list):

- a behavioural view showing the communication between processes,
- a database view showing the data structures,

- filtered information focusing on a certain aspect like de-facto process interfaces as derived from the process description,
- basic information on the attributes of an instance.

An important benefit is the derivation of "hidden" information, which is information included in the inputs but not really visible for an engineer. As an example consider performance figures: from the specified occurrence of a message the occurrence of following messages can be derived, from the occurrences, the size of data and the used channel the channel load can be calculated and so on.

This is important because constraints (limits) may be specified, which are not directly compatible with the figures specified – as it is for the channel load. The great benefit of a sound and specialised meta-model is that it allows to derive such information immediately when requested for every item needed.

4.1.4 Link to Project Management

As an extension to the technical domain, the meta-model supports a link to project management:

1. For every requirement an action, e.g. for maintenance, an actor, a deadline, the status of progress and the related work package can be given.
2. Every requirement can be
   a. classified as being part of the basic version, an extension or a certain release.
   b. marked as an assumption, not being confirmed yet.
   c. marked as needing principal clarification.

Reports on the aspects of project management are generated according to a number of criteria.

An example for a simple quality gate rules imposed on a specification are: no assumptions may be present any more and no error may be present.

4.1.5 Support of Concepts

As activities are expressed via the IPO-formalism, the organisation of requirements easily allows an implementation of SOA services (SOA = Service-Oriented Architecture): every exchanged message is a candidate for a SOA service.

Also, modularisation (componentization) and object-orientation of the requirements is inherently enforced by the applied organization principle.

4.1.6 Derivation of Test Cases

Due to the IPO-approach test cases for system level tests can be derived very easily – as this can be done automatically by the tool.

Every input initiating a sequence of actions identifies such a test case. Consequently, the tool can list all such cases together with the needed and produced data, involved processes, processors and (human) roles.

4.1.7 Examples of Application

Practical results from real projects are given below (non-exhaustive list).

4.1.7.1 Quality Assessment 1 (QA2)

The quality of requirements expressed in MS-Word had to be assessed based on the metrics provided by the SRM tool from BSSE.

About 30 documents amounting to about 1500 pages in total were provided. The requirements were expressed in text, mainly, and to a small part in tables. About 100 man-months (estimated to about 14000 man-hours) were spent.

The customer asked for a feedback on the quality because evaluation of the text was impossible in practice. The contents of the documents was analysed manually and converted into an input to the SRM tool. This yielded about

- 1000 (formal) requirements (SRM entities)
- 300 textual requirements which could not be converted due to unclear text
- 585 errors in total (according to severity: 330 low , 70 medium, 185 high)

In a cross-check of a sample of these errors the related deficiencies were also found in the original documents. The conclusion given to the customer was: the quality is very poor. Due to the high number of errors (requirements / errors ≈ 2) and open questions (300) no real conclusion on the degree of completeness could be made at all. As the project was aborted, no further data could be derived.

4.1.7.2 Quality Assessment 2 (QA2)

A customer asked for a quality assessment of a specification of an inteded SOA system expressed in a UML model. It consisted of more than 5600 elements (UML entities), an outcome of a project which consumed about 150 man-years (estimated to about 200.000 man-hours). Probably, more elements were included, but the XMI-file could not be read completely, possibly due to incompatibilities of XMI-syntax or non-conformances with UML syntax in the model.

Modelling focused on Activity Diagrams to express the functionality / behaviour of the system. About 255 activities of business processes were described.

The model contents on inspection indicated lack of understanding on the semantic of UML2 activities, as for example alternative entries into activity chains were described in such a way that UML2 semantic dictates them to be in fact parallel thereby in conflict with what was intended.

Further, no use was made to express the logic flow, so that the distribution of responsibility was unclear and communication requirements could not be derived. As a consequence, the identification of service candidates for a SOA design would have required high additional effort, as the natural separation of concerns due to inherent responsibility borders could not be considered automatically.

It is to be noted that the modellers were previously thoroughly trained in multi-week courses by the vendor of the UML2 tool, not only regarding the tool but also regarding the application of the UML2. Necessary tailoring of the modelling environment and applicable rules did not take place.

4.1.7.3 PLM Application

A PLM (Product Lifecycle Management) application was specified with the SRM tool. All processes needed to be defined for product definition (technical and commercial aspects, manufacturing), shop portal and control of the delivery (including fulfilment and complaint management).

In a first iteration, about 1000 requirements were collected at an effort of about 1000 man-hours.

4.1.7.4 Shop / Portal Application

The application covered the operations of an Internet portal from selection and configuration of a (set of) product(s), purchase order, billing, and delivery to payment at the end. About 300 requirements were collected with the SRM tool at an effort of about 100 man-hours.

4.1.7.5 Bank Transfer

All operations needed to complete a bank transfer were specified with the SRM tool: ~400 requirements at an effort of about 50 man-hours.

4.1.7.6 Embedded System

A rather complex real-time system of about 35 processors spread over 2 processors was specified by an SRM tool supporting “executable specifications”. From this specification binary code was automatically generated by the tool. The system passed ESA acceptance tests in 2003 and is now successfully operated on-board of ISS.

About 5000 requirements were expressed in the model at an estimated effort of about 1000 man-hours.

4.1.8 Efficiency Considerations

The following table Tab. 4-1 shows efficiency figures for the examples described above.

The figures should be interpreted as an indication of a trend, not as figures of high precision, because the effort estimation was not exactly tracked but was a matter of a rough estimation.

From a principal point of view the size of the project impacts the efficiency. Therefore for small sets of requirements the efficiency is higher, though the efficiency itself also depends on the type of the application and the experience and knowledge of the engineers.

Example	Tool	# RQs	Effort / m-h	Efficiency / (RQ / m-h)
QA1	Word	1000	14000	0.07
QA2	UML	~5600	200000	~ 0.028
PLM	SRM	1000	1000	1
Shop	SRM	300	100	3
Bank Trf.	SRM	400	50	8
Embedded	SRM	5000	1000	5

Tab. 4-1: Efficiency Figures

4.2 Cross-System Engineering: System Synthesis

The approach described in Sect. 4.1 applies to the operational part of a system specification. However, more properties of a system need to be specified such as its mechanical structure, (electronic/electric) hardware, thermal properties, interfaces to its environment etc.

For a mechanical structure its geometrical properties, maximum weight, surface properties, stiffness etc. need to be specified. As thermal properties the heat load, the maximum load or the heat conductance need to be given, and so on.

Moreover, dependencies, for example, between structural and thermal properties or between structural components may exist. It is of high importance to achieve consistent requirements for all these aspects, and even more important, to keep them consistent when requirements evolve and during maintenance.

Domain-specific meta-models can take care of the needs of the structural and thermal (and other) domains and can establish dependencies across domain boundaries. Another meta-model can connect the meta-models of the subsystem domains on system-level. This allows to deal with the different domains in an integrated manner

such that a tool automatically can assess on the quality of the whole set of requirements or, as far as validation is concerned, provide proper support for such an assessment.

4.3 Mission Definition

High-level requirements such as defining a mission require a different meta-model although they are addressing aspects described in Sect. 4.1 and 4.2. However, at this stage the intention is not to specify system operations or system decomposition completely, but only to some part. Hence, a check on completeness like in above cases would fail.

More likely, the requirements at this stage will implicitly address requirements of the domains mentioned above, and possibly one requirement will address several requirements of one or more sub-domains. This leads to the issue of tracking whether a high-level requirement is considered later by sub-domain requirements – in a consistent manner.

This issue can also be solved by an appropriate meta-model, relating an early phase with later phases of a project, while allowing a tool to take over tracing of requirements which are not expressed by an engineer, but are derived by the tool accordingly.

On each dedicated level of system decomposition the relevant figures can be provided immediately, such as budgets which need to be derived from sub-budgets and have to be compared with limits.

4.4 Project Management

Though project management is quite different from the (technical) applications described above, it also may be supported by a meta-model to check project planning. Instead of technical requirements work packages and their contents and dependencies are the elements of modelling, verification and validation.

Work packages and their scheduling may become complex rather soon due to explicit or implicit dependencies between work packages and on (human, technical or organisational) resources.

Implicit dependencies arise from needed inputs and produced outputs and their deadlines, allocated / estimated effort, availability of resources.

Element types of the meta-model would be the elements already mentioned, and in addition (non-exhaustive list) costs, hourly rates of personnel, rates of technical resources, internal tasks of a work package, organisational units and so on.

As a result, all the dependencies and constraints can be checked. Derived information is e.g.: costs and cost distribution, work load of personnel, statistics on complexity of planning, and an input for a conventional planning tool like Microsoft™ Project®.

As an example, the following quantities were successfully processed:

- 36 work packages including 171 activities,
- 26 employees involved,
- 178 deliverables or contributions to them, recorded as an output from a work package.

The effort to define the contents of the planning and to get it free of errors, amounts to approximately 200 man-hours which yields about 6 hours per work package to get a consistent and complex work plan.

5 EXAMPLE INPUT AND OUTPUT

To be provided in the full paper

6 CONCLUSIONS

The discussion and the presented examples and figures show that the quality of a specification and the efficiency to establish it can be significantly improved when applying a systematic approach to requirements management. The achieved improvements are a consequence of applying a domain-specific approach which allows to benefit from domain-specific knowledge and to go beyond pure administration and content management of requirements.

Though limited to a certain domain there exists an infinite number of applications which makes it reasonable to establish specific tools and not to rely on universal tools. Building a new domain-specific tool once another tool is already available is much cheaper than doing it the first time because the mechanisms to be implemented are quite similar for different domains as the examples on IT applications and project management demonstrate.

7 REFERENCES

[1] HOOD, DESIRe®,Dynamic Expert System for Improving Requirements, http://www.hood-group.com/

[2] ADMIRE, Advanced Management of Informal Requirements, http://www.melchisedech.net

[3] SOPHISTGmbH, http://www.sophist.de/

[4] DOORS, http://www-01.ibm.com/software/awdtools/doors/

[5] Polarion® Requirements™, http://www.polarion.com

[6] Requisite Pro, IBM, http://www-01.ibm.com/software/awdtools/reqpro/

[7] CaliberRM™, Borland, http://www.borland.com/us/products/caliber/index.html

[8] UML, Unified Modelling Language, OMG, http://www.uml.org/

[9] SysML, Systems Modelling Language, OMG, http://www.omgsysml.org/

[10] M. Kohlhase: Using LaTeX as a Semantic Markup Format; pp. 279–304 in Mathematics in Computer Science; Birkhäuser 2008

[11] G. Frege: Funktion - Begriff – Bedeutung. Mark Textor (ed.), Göttingen: Vandenhoeck & Ruprecht, 2002.

[12] G. Nelson: Natural Language, Semantic Analysis and Interactive Fiction, 2005. Available at http://inform7.com/learn/documents/WhitePaper.pdf, last revised April 10th, 2006.

[13] AADL, Architecture Analysis and Design Language, SAE (Society of Automotive Engineers), http://www.aadl.info/

[14] T. Schattkowsky and A. Forster: On the Pitfalls of UML 2 Activity Modeling. In Proceedings of the international Workshop on Modeling in Software Engineering (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC

[15] H. Störrle and J.H. Hausmann: Towards a Formal Semantics of UML 2.0 Activities. In Proc. Software Engineering 2005, 2005.

[16] H. Eichelberger, Y. Eldogan and K. Schmid: A Comprehensive Survey of UML Compliance in Current Modelling Tools. In Proc. Software Engineering 2009, 2009.

[17] R.Gerlich, D.Sigg, R.Gerlich: Model Transformation in Practice. In Proc. DASIA'07, Data Systems in Aerospace, managed by Eurospace, May 2007, Naples, Italy, ESA SP-638