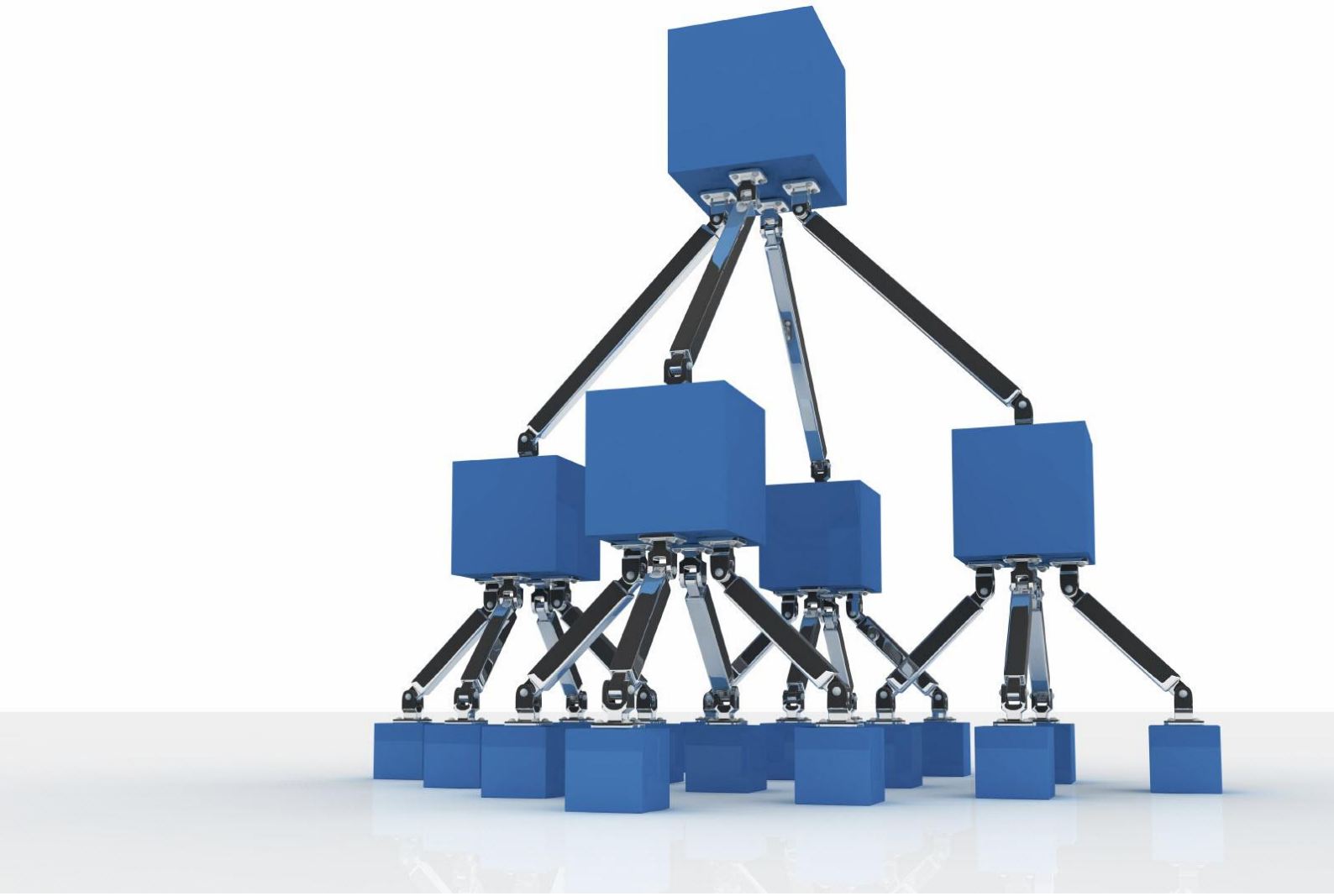

Using statement-level templates to improve the quality of requirements

An Integrate white paper

integrate
systems engineering

Using statement-level templates to improve the quality of requirements

Abstract

Much has been written on the subject of what constitutes an acceptable requirement statement, and tools are available to check the quality of a requirement in terms of grammar and vocabulary after it has been written. Using a statement-level template (or “boilerplate”) is a means of creating an acceptable requirement as it is written, rather than after the event.

There have been a number of papers published on using templates and structured language for improving requirements quality, and for relating textual requirements to formal specifications of various kinds. This paper focuses on the use of statement-level templates to aid in the expression of textual requirements in general, including stakeholder requirements at levels of abstraction above the specification. The aim of the approach is to provide grammatical structures that encapsulate the accepted rules for writing singular, unambiguous, verifiable requirements.

The paper surveys a number of tools for supporting statement-level templates, and describes some experiences of applying the use of templates to projects.

Introduction

It is received wisdom in the discipline of requirements management that textual requirements should possess a number of characteristics, such as singularity, non-ambiguity and verifiability. A synthesis of such properties can be found in [1] and in more detail in [12]. A number of tools are available, such as LEXIOR [2], and RQA (Requirements Quality Analyzer) [3], that analyze textual requirements, and identify where they fail to possess such characteristics.
This paper reviews an approach to assuring that such characteristics are present in the requirements as they are written, rather than assessing them after the event. The approach encourages the author to select a pre-determined textual template [18] specific to the type of requirement to be expressed, and to fill out place-holders to instantiate it.
There is a body of published work in the area of requirements templates (or, more precisely, patterns) associated with formal specifications, which work is briefly reviewed below. The main motivation of such work is to make the specification of a system more approachable through the use of structured text that has a precise relationship with one or more underlying formal models.
The approach described in this paper is slightly different. We do not attempt to tie the templates to any underlying formal model, and argue that there are still significant benefits to be gained, as enumerated below. What’s more, the approach can address a very broad range of kinds of requirement at all levels of abstraction, including high-level stakeholder goals, aspirations and requirements.
Early work on the use of this kind of requirement template dates from 1998. Although not formally published, the basic approach can be found in [18] and at [4]. Each template is an English language sentence with place-holders for specific words that define the particular requirement. No constraints are placed on the kinds of words that can fill the place-holders; only encouragement to use consistent vocabulary.
More recent work [15], [19] has applied the approach in a very specific domain and attempted to quantify the benefits. Later development makes use of semantic parsing with underlying domain-specific ontologies to guide the author in the selection of appropriate terms, units and values.
The benefits of using this approach to author requirements include:

- *An aid in articulation*. Being able to draw upon a palette of classified templates helps in finding effective ways of expressing particular kinds of requirement or constraint.
- *Uniformity of grammar*. By always considering the reuse of existing templates, the same kinds of requirement will always be expressed in the same kinds of way, leading to a consistent use of language in expressing requirements.
- *Uniformity of vocabulary*. Each placeholder in a template can be controlled through application of ontology. For instance, if the place-holder is expected to contain a "mode

of operation", the ontology can articulate the allowable vocabulary.

- *Ensuring essential characteristics*. The use of a template aids in assuring that the requirement statement is unambiguous, quantified, testable and other essential characteristics.
- *Easier identification of repeated and conflicting requirements*. When requirements are expressed in uniform way, automatic detection of potentially repeated or similar, requirements becomes possible. Likewise, candidates for contradictory requirements can be identified.
- *One-stop control over expression*. With effective tool support, global changes in the way requirements are expressed can be effected by just changing the template in one place. E.g. changing "will" to "shall".
- *Protection of classified information*. Often the only part of a classified requirement is the quantity associated with it – e.g. the speed of the vessel, the rate of fire. Using templates, the classification can be focused on just the restricted attributes, allowing the rest of the requirement to be viewed for non-restricted use.

Terminology

We wish here to make a distinction between “templates”, “boilerplates” and “patterns”. The term “template” is most often used in the field of requirements management to refer to the heading structure of a requirements document. The use of such templates helps ensure that the authors consider the complete range of concerns when collating a set of requirement statements.

Because of this use of the word “template”, early work [18] in the subject of this paper coined the term “boilerplate” to refer to a grammatical structure for an individual textual requirement. In this paper, we use the term “statement-level template” to refer to this concept.

“Patterns” usually refer to something broader than an individual statement. Design patterns [21], for instance, are bundles of artefacts that frequently occur together when defining a particular kind of design. One could imagine requirements patterns as sets of related requirements that together provide a complete specification of a particular feature. This paper is not about such patterns.

Related work

Existing work on the concept of requirements templates falls into three categories:

- The use of controlled English grammar to express properties that have an underlying formal semantics.
- The use of templates in tabular rather than textual form.
- The use of statement-level templates.

Approaches related to formal models

Examples of the use of statement-level templates and patterns associated with formal notations include:

- *(1999) Finite state automata and Petri Nets*. In [6], specification patterns are related to a formal semantics based on finite state automata. This work is later extended by [7] to cover real-time aspects. Here the patterns are given projections into three different formalisms.
- *(2001) Safety. Finite state automata* in the safety domain are considered by [8].
- *(2008) Probabilistic verification techniques*. The work of [5] applies the concept of specification patterns to make it easier for authors to formulate probabilistic properties of systems, improving the acceptability of formal verification techniques.
- *(2009) Requirements Documents generation*. The use of formally declared requirements patterns, as in [13] allows semi-automatic techniques to create software requirements documents.

The frequently cited objective of using such templates or patterns is to allow the specification author to specify system properties in familiar language controlled in such a way that there is an underlying formal semantics. The advantages of this are clear: automated formal verification and model-checking techniques can be brought to bear on the specification despite the requirements being expressed in natural language. The most relevant drawback has to do with the difficulty to use a strongly formal approach to write requirements.

Approaches that use tabular templates

Examples of approaches that use tabular templates or patterns without formal semantics are:

- *Planguage.* In [9], tabular templates with keywords are used to specify requirements rather than relying on grammatical sentences. One of the key motivations for this is to ensure that an appropriate set of performance requirements and constraints are considered for each functional requirement, thus ensuring completeness. The consistent use of keywords also ensures uniform and unambiguous language.
- *Volere*. In [10], a form of tabular template is proposed called a “snow card”, which ensures that certain information, such as rationale and requirement type, is associated with each requirement statement.

These approaches are not primarily textual-requirement-centric, attempting to provide semi-formal representation schemas.

Approaches based on statement-level templates

In this paper, we make no attempt to associate the statement-level templates with a formal semantics. We simply control the language used to reduce the risk of ambiguity or inconsistency, and to maintain certain important characteristics of requirements statements we focus on the expression of singular textual requirements. Nor do we address the completeness of sets or sub sets of related requirements.
Other work that is most similar to this approach includes:

- *(2002) Requirements Boilerplates*. The ideas presented in Chapter 7 of [18] are the most direct predecessor of this work.
- *(2009) Easy Approach to Requirements Syntax (EARS) and Adv-EARS*. In [15] and [16], a mapping of the grammatical constructs of English has been use to formalize requirements for a specific domain: aero-engine control systems.
- *(2010) The Requirements Specification Language (RSL) of CESAR*. The CESAR research project, funded by the European Union AREMIS program, reviewed work on the use of boilerplates [14] with a view to extending and applying the approach to a number of safety-critical domains, with discussions of how to formalise the approach using ontologies.

All three of these references will be covered in greater detail in the next section.

Project Experience

The purpose of this section is to report on a number of projects that have defined and applied statement-level templates. Some of these projects were completed with no formal publication of their results; others do have related publications, and these are cited; yet others are on-going experiments.

Future Surface Combatant

In 1998, the concept of requirement statement templates was applied to a UK defence project in which requirements were being elicited for a naval warship, Future Surface Combatant (FSC). It was observed that requirement authors had difficulty in articulating certain kinds of requirement, such as timeliness and periodicity. Once shown a good example of how to do it, however, the difficulty was largely overcome. A palette of “boilerplate” requirements was established for the project containing examples of many kinds of requirement, carefully phrased to avoid ambiguity and to ensure proper quantification. (The term “boilerplate” was used to distinguish the concept from requirements document templates, which consist purely of a heading structure.)
The palette was organised into types of property or constraint, and each boilerplate consisted of a complete sentence with place-holder words that the author could replace with specific terms. The placeholders were named consistently across all the boilerplates, so that, for instance, “<stakeholder>” and “<unit of volume>” might appear in multiple boilerplates where the same concept was referenced.
The requirements writing process was defined as follows:

1. Consider what kind of requirement or property you wish to articulate.
2. Select the most appropriate boilerplate from the palette.
3. Fill out the place-holders in the boilerplate to form the specific requirement.

As enthusiasm for the concept grew, tool support was provided that supported the requirements author in steps 2 and 3. The tool encouraged the author to select and use consistent vocabulary across requirements by offering terms already used for the same placeholder in other requirements. Beyond this, no attempt at formalising an ontology or glossary was implemented.
With tool support, a requirement statement became, in effect, a boilerplate and a set of values assigned to the placeholders. By representing the requirements in this fashion, it became possible centrally to control and evolve the precise way in which different types of requirement were expressed. Simply by changing the global boilerplate, the text of all requirements that used the boilerplate was also updated.
It also became possible to abstract from the requirements all the values that had been used for the placeholders. For instance, the placeholder “<function>” gave a list of all the functions mentioned, and “<user>” gave a list of all users mentioned.
Table 1 shows some examples of the boilerplates that were used. In the left-hand column are the boilerplates hierarchically organised into clauses that may be combined to construct complete requirement statements. The right-hand column shows two things: the classification of the combined clauses, and a complete example instantiation.

Table 1: Example boilerplates from FSC

BOILERPLATE	*CLASSIFICATION & EXAMPLE INSTANTIATION*
While <operational condition>,	**Mode** While *disconnected from a source of power, ...*
... the <user> shall able to <capability>	**Mode/Capability** While *disconnected from a source of power*, the *photographer* shall able to *capture still images*
... at least <quantity> times per <time unit>	**Mode/Capability/Rapidity (Maximise, Exceed)** While *disconnected from a source of power* the *photographer* shall able to *capture still images* at least *10* times per *minute*
... for a sustained period of at least <quantity> <time unit>	**Mode/Capability/Rapidity/Sustainability (Maximise, Exceed)** While disconnected *from a source of power* the *photographer* shall able to *capture still images* at least *10* times per *minute* for a sustained period of at least *4 hours*
... the <system function> shall able to <action> <entity>	**Mode/Function** While *in winds up to Beaufort scale 8* the *missile launcher* shall able to *fire missiles*
... within <quantity> <time unit>s from <event>	**Mode/Function/Timeliness (Minimise, Do not exceed)** While *in winds up to Beaufort scale 8* the *missile launcher* shall able to *fire missiles* within *5 seconds* from *receipt of warning signal*

The application of this approach in the project had a positive effect on the confidence of the requirements authors in expressing the requirements. At the end of the exercise, the project had developed a palette of about 120 boilerplates for different kinds of requirement. This palette represented an intellectual asset to the organisation to pass on to future projects.

Baggage Handling

A similar approach to the above was applied by the British Airports Authority (BAA) in gathering requirements for the London Heathrow Terminal 5 Baggage Handling System, but without specific tool support. Their boilerplates were referred to as “statement level templates” (as opposed to document templates), and templates were provided at the stakeholder level for capabilities, constraints, assumptions and definitions. A different style of template is used at the application-specific level, where more structure can be prescribed.

Table 2 shows an example capability template used in the baggage handling project.

Table 2: Example capability template from baggage handling

Templates

<Actor> shall be able to <Act> <Action_Subject> whilst <Qualification> <Qualification_Subject>.

<Actor> shall be able to <Dispatch_Act> <Action_Subject> from <Dispatch_Object>.

where

<Actor> ..... the user (person or external system) who interacts with the system to be developed in order to perform the Act.

<Act> ..... the act the actor needs to perform to achieve some business goal.

<Action_Subject> ..... the part of the application domain which will be affected by the Act.

<Qualification> ..... a description of the circumstances under which the Actor should be able to perform the specified Act.

<Qualification_Subject> the part of the application domain over which the Qualification is defined.

<Dispatch_Act> ..... the dispatch act the Actor needs to perform to achieve some business goal. (A dispatch act engages in a different grammatical construct, because it is expected to require an indirect object, the Dispatch_Object).

<Dispatch_Object>..... the location from which the dispatch takes place.

As an example how the capability template was used, an original, poorly expressed user requirement was re-expressed as shown in Table 3. By being constrained to use provided templates, requirements authors are guided into expressing requirements in a uniform style at the appropriate level of abstraction. Hence the emphasis is shifted from solution to capability.

Table 3: Re-expression of user requirement

Original requirement

Each check-in desk shall be equipped with weigh, label and take-away conveyors, feeding collector conveyors.

Re-expression as a set of capability requirements

Check-in clerks shall be able to *weigh std baggage items* whilst *stationed at the check-in desk*.

Check-in clerks shall be able to *label std baggage items* whilst *stationed at the check-in desk*.

Check-in clerks shall be able to *despatch to a take-away conveyor std baggage items* from *the check-in desk*.

Check-in clerks shall be able to *despatch to a feeding collector conveyor std baggage items* from *the check-in desk*.

Requirement author reaction to this approach was mixed: some complained that they felt the templates restricted their freedom of expression. However, the project managers persisted with the approach because they felt the benefits of uniformity of expression outweighed the constraints.

Easy Approach to Requirements Syntax (EARS)

Reported in [15] and [19] are some experiments in apply the statement-level template approach to the specification of properties of aero-engine control systems. Because the domain is so specific, and applied at a technical system level rather than user level, the experiment was able to use just 4 statement templates to cover the majority of requirements. These are shown in Table 4.

Table 4: Templates for control systems

Ubiquitous requirements The <system name> shall <system response>	***Example*** The *control system* shall *prevent engine overspeed*
Event-driven requirements WHEN <optional preconditions/trigger> the <system name> shall <system response>	***Example*** WHEN *continuous ignition is commanded by the aircraft*, the *control system* shall *switch on continuous ignition*
Unwanted behaviours IF <optional preconditions/trigger>, THEN the <system name> shall <system response>	***Example*** IF *the computed airspeed fault flag is set,* THEN the *control system* shall *use modelled airspeed*
State-driven requirements WHILE <in a specific state> the <system name> shall <system response>	***Example*** WHILE *the aircraft is in-flight*, the *control system* shall *maintain engine fuel flow above XXlbs/sec*

Note the use here of capitalised key words to make it clear exactly which template is in use.

Reference [15] reports an improvement in requirement quality when judged by a small set of criteria. The later reference [19] evolves the approach, adds some more templates, and uses more refined criteria for assessing the benefits accrued.

Context RDS

Context RDS is an experimental toolkit that develops the concept of statement templates strongly towards the use of domain-specific ontologies. The tool is being applied to projects within the aero-engine division of Rolls-Royce where templates are being used to frame requirements for engine control systems, a domain that lends itself well to stylised requirements.

In RDS, statement-level templates contain placeholders that may be of the following types:

- Process Defined
- Value by Reference
- User Defined Term
- User Defined Text

Placeholders that obtain their value by reference can only be populated by terms of one or more type, defined in a project-specific process model. For example, “While <Condition> is.....” has a placeholder that can be constrained to allow references to atomic condition or complex conditions; atomic conditions have further constraints. User Defined Terms have templates with similar placeholders, thus allowing terms to reference other terms.
Figure 1 shows an example template structure, and Figure 2 shows an example instantiation.

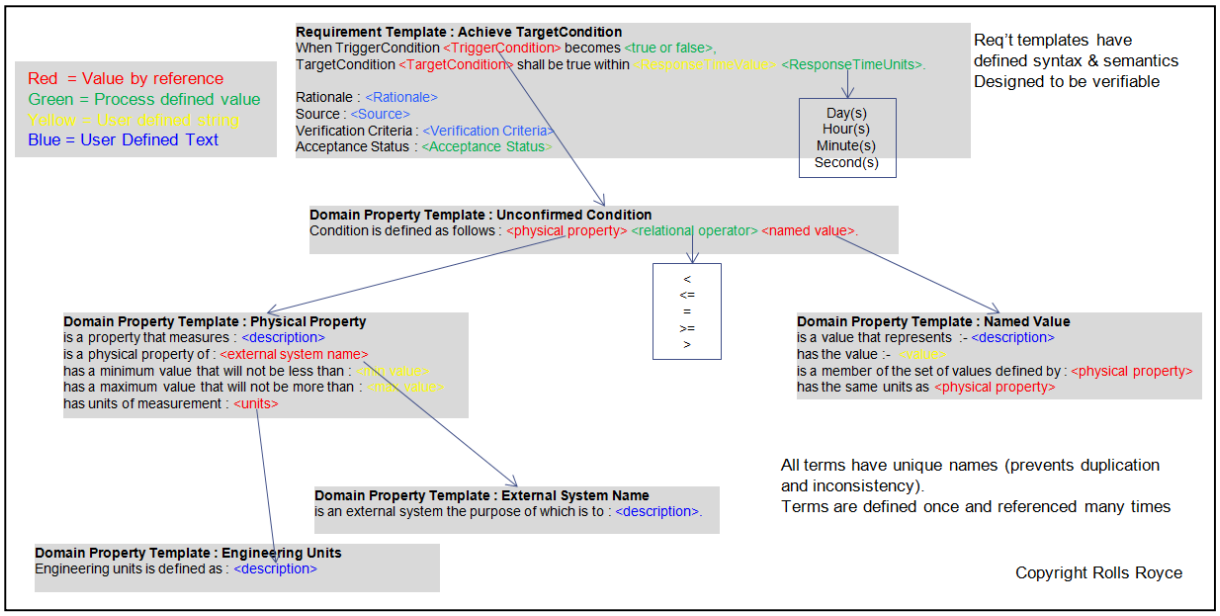

Figure 1: A Context RDS Template Structure

The configuration of the tool constrains how authors are able to instantiate the templates. This ensures, for instance, the consistent use of terms and units, and prevents nonsense requirements created by meaningless instantiation.

It is considered too early in the experience of using Context RDS to be able to report specific feedback. However, it is evident that considerably more up-front work and cost is engendered by this approach; it is hoped that the investment will return dividends later in the project.

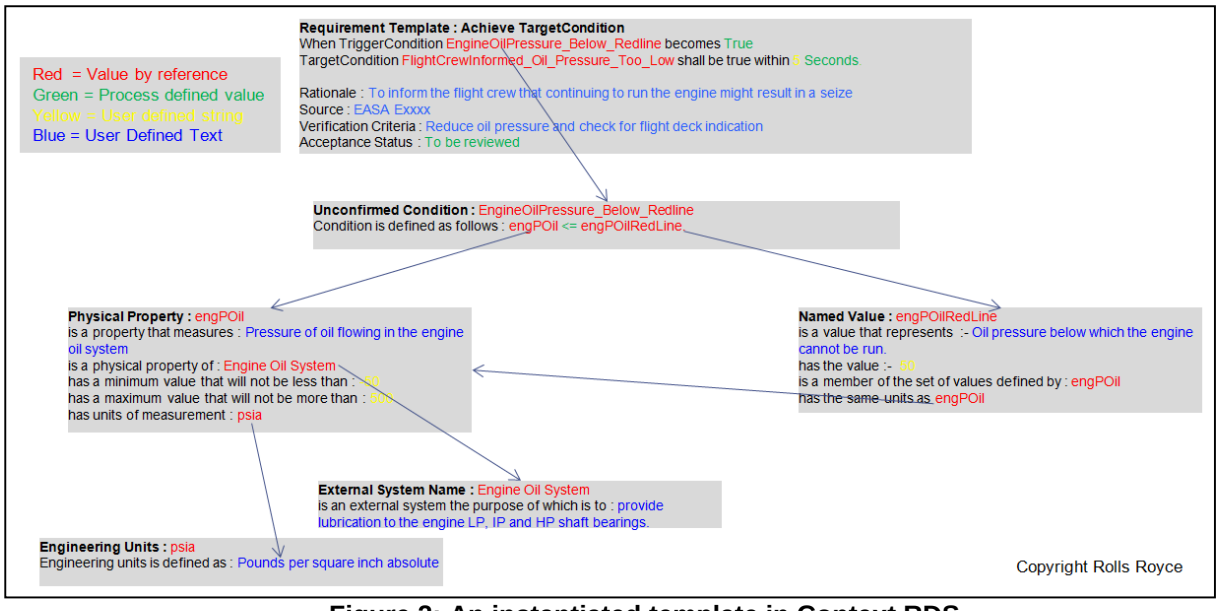

Figure 2: An instantiated template in Context RDS

The Requirements Specification Language (RSL) of the CESAR project

A main goal of the European-funded CESAR project was to consider way of improving the development process for a number of safety-critical domains, such as aerospace and rail. In their analysis of opportunities in requirements engineering, the use of boilerplates was identified as candidate for bringing about improvement.

A key CESAR report [14] considered two things together: a Requirements Specification Language (RSL) largely based on the boilerplate concept, and a Requirements Meta Model (RMM) consisting

of a general and domain-specific ontology for supporting requirements vocabulary. The report notes that such ontology provides the semantic base for which requirements can be analysed, validated and verified.

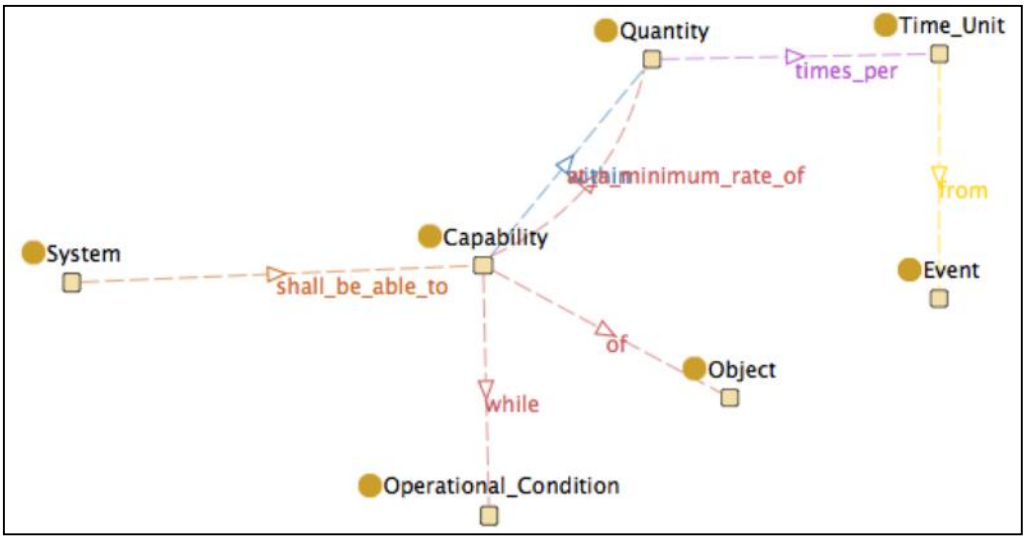

Figure 3: An example ontology underpinning requirement statements

Figure 3 shows an example ontology, from which a variety of statement-level templates could be generated by traversing parts of the structure as appropriate. The nodes are typed entities that form the placeholders, and the arcs describe the relationships between those entities.

Requirements Authoring Tool (RAT)

Another project relates to a recently released product, the Requirements Authoring Tool (RAT) from The Reuse Company [17]. RAT allows the user to start typing a requirement, and simultaneously determines which of a palette of statement-level templates are applicable, guiding the user on permissible terminology. If the requirement that the user is typing eventually matches a template, the user is informed of the fact.

To achieve this, it combines three things:

- A set of pre-determined templates.
- A semantic parsing engine.
- A strong underlying domain-specific ontology.

The combination of these properties allows the requirements author to write requirements in an assisted way, as a counterpart to the “fill the gap” approach to placeholders. The possibility of not selecting a-priori a pattern before typing a requirement leads to the tool being considered as an assistant, as well as a coach, and users may feel their creativity is less constrained.
At present, the tool comes equipped with about 70 statement-level templates that can be combined in various ways. RAT allows the evolution of the set of templates and associated ontology to be managed as part of the enterprise’s knowledge base. In doing this, it introduces the specialist role of “knowledge manager”, in addition to that of “author”. If authors find that the tool is able to match templates to the requirements they are trying to express, they can apply to the knowledge manager to introduce new ones, with associated terms. Such a role requires an understanding of the purpose of ontology, and how to use the tool to define it.

Future Directions

This section presents some ideas for the future development of the approach.

Extension to artefacts other than requirements

Works so far has focused on controlling the expression of requirements. Other artefacts closely related to requirements may benefit from the same approach. For instance:

- Descriptions of V&V activities, their criteria and expected outcomes.
- Requirements flow down structures and associated decomposition rationale. (See the concept of rich traceability [22].)

The two aspects just mentioned relate to two key relationships that are typically articulated during requirements management activities: the decomposition (or satisfaction) relationship between layers of requirements, and the association of test activities with requirements. In both cases, a particular type of requirement will typically require similar patterns of decomposition and testing. This gives rise to the idea of extending the statement level templates into design patterns

that bundle a requirement with its decomposition structure and test structure.

Conceptual representation of requirements

In the templating approach described here, the textual requirement is the primary representation, because there is no underlying formal semantics. In the other approaches described in section 2, the textual representation is merely a projection of a formal construct. The absence of a formal semantics makes the approach applicable to the authoring of requirements that are at a level of abstraction that does not lend itself to formal modelling.

However, it may be possible to abstract away from the textual representation by using conceptual graphs in the style of [11]. More general than any specific formal specification language, conceptual graphs are a candidate for a representation of requirements and system properties from which textual requirements could be projections. From a single conceptual representation of a requirement, a textual statement could be project in any chosen natural language. Projections from conceptual graphs into modelling notations, such as SysML may also be possible.

Conclusion

We have attempted to describe hitherto unpublished work on the use of templates – or boilerplates – for textual requirements statements. We have described the general approach, with several specific applications, and a number of tools that offer support. The only available quantitative analysis of the benefits of such an approach is based on small studies [15] [19]. Real practical experience does strongly suggest, however, that the use of templates offers benefits to authors of requirements and manager of projects.

References

[1] Guide for Writing Requirements, INCOSE Product INCOSE-TP-2010-006-01, V. 1, April 2012 (requires login)

[2] LEXIOR from Cortim

[3] Requirements Quality Analyser from The Reuse Company,

[4] Requirements Engineering Boilerplates

[5] Grunske, L.: Specification patterns for probabilistic quality properties. In: ICSE, New York, ACM (2008) 31-40

[6] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, New York, ACM (1999) 411-420

[7] N. Abid, S. Dal Zilio, and D. le Botlan. A Real-Time Specification Patterns Language. Technical Report 11364, LAAS, 2011.

[8] Friedemann Bitsch, Safety Patterns - The Key to Formal Specification of Safety Requirements, Proceedings of the 20th International Conference on Computer Safety, Reliability and Security, p.176-189, September 26-28, 2001

[9] T. Gilb. *A Handbook for Systems and Software Engineering using Planguage*, Addison Wesley 2001

[10] Suzanne Robertson and James C. Robertson. *Mastering the Requirements Process*, 2nd Edition, Addison Wesley 2006.

[11] John Sowa, *Conceptual structures: Information processing in mind and machine*, Addison Wesley, 1983

[12] Genova, G. Fuentes, J. Llorens, J. Hurtado, O. Moreno, V. *A framework to measure and improve the quality of textual requirements*, Requirements Engineering, Springer, Url: http://dx.doi.org/10.1007/s00766-011-0134-z, Doi: 10.1007/s00766-011-0134-z

[13] Renault, S. Mendez, O. Franch, X. Quer C. A Pattern-based Method for building Requirements Documents in Call-for-tender Processes. International Journal of Computer Science & Applications (IJCSA), 6(5): 175-202 (2009)

[14] Definition and Exemplication of RSL and RMM, CESAR Project Technical Report D_SP2_R2.1_M1:

[15] Alistair Mavin, Philip Wilkinson, Adrian Harwood, Mark Novak, Easy Approach to Requirements Syntax (EARS), 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31-September 04

[16] Dipankar Majumdar, Sabnam Sengupta, Ananya Kanjilal, Swapan Bhattacharya, “Adv-EARS: A Formal Requirements Syntax for Derivation of Use Case Models”, Proc of the First International Conference on Advances in Computing and Information Technology, July, 15-17, 2011, Chennai, India. pp 40-48.

[17] Requirements Authoring Tool (RAT) from The Reuse Company,

[18] E. Hull, K. Jackson and J. Dick: Requirements Engineering. First Edition, Springer 2002.

[19] Alistair Mavin, Philip Wilkinson: BIG EARS (The Return of “Easy Approach to Requirements Syntax”, 2010, 18th IEEE International Requirements Engineering Conference, Sydney, Australis, September 27 - October 1

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[22] "Rich Traceability", Jeremy Dick, Proc. 1st International Workshop on Requirements Traceability, Edinburgh, Sept 2002

About the authors

Dr. Jeremy Dick was educated at London University's Imperial College in Computing Science, majoring in formal methods of software development. He went on to pursue academic and industrial research in this field for a number of years. From 1996 to 2006, he worked for software tool supplier Telelogic (now IBM) in their UK professional services organization, as a principal consultant in tool-supported requirements management. This role has afforded him a broad exposure to requirements management practices and issues across many industry sectors. In this role, he developed the concept of Rich Traceability. He co-authored the Springer book entitled "Requirements Engineering," and toured internationally giving seminars in this subject. In 2006, he moved to UK-based consultancy Integrate Systems Engineering Ltd, where he has been helping to develop and promote the concept of Evidence-based Development. He is a past Chairman of INCOSE's Requirements Working Group.

Dr Juan Llorens received his MS degree in Industrial Engineering from the ICAI Polytechnic School at the UPC University in Madrid in 1986, and his PhD in Industrial Engineering and Robotics at the Carlos III University of Madrid in 1996. In 1987 he started his own company dealing with Artificial Intelligence applied to Database systems, named Knowledge Engineering SL (KE). Dr. Llorens worked as technical director in KE and as Marketing Director in Advanced Vision Technologies until 1992, when he joined the Carlos III University. Since 2003 he has been Professor of the Informatics Department of the University. His main subject is Software and Knowledge Reuse, which he teaches in the Informatics and Information Science graduate studies department at the University. Dr. Llorens is the leader of the KR Group (Knowledge Reuse Group) within the University. In 1998 he was invited to the Högskolan på Åland (HÅ) (Åland, Finland). From 1998 to 2008 he split his educational activities between Madrid's University and the HÅ, where he taught Object Oriented Software design. Since 2008, based on an agreement with his University, he shares his academic duties with the position of R+D officer in The Reuse Company (a brand of the University's spin-off organization named CISET) that commercializes systems engineering quality and reuse technology. Since 1999 he has been the board of CISET.

His current research involves the study of Knowledge technologies and System Engineering techniques integration towards Software and Information quality measurement and reuse. He is member of INCOSE's (International Council on Systems Engineering, www.incose.org) Requirements Working Group.

His CV is presented at http://www.kr.inf.uc3m.es/juanllorens.htm

About Integrate

integrate's vision is for companies to align strategy and execution, drive quality and ensure compliance through using systems engineering techniques as the bridge between the domains of engineering and management. Our consultancy practice and supporting technologies help clients deliver this vision. We use systems engineering as a vehicle to support informed management and direction, and to optimise solutions against both technical and business constraints. Our approach is intensely pragmatic, firmly focused on our clients' delivery needs, and founded on a thorough understanding of the underlying problem.

For more information go to http://www.integrate.biz or email info@integrate.biz

This is an edited version of a paper presented at ICSSEA 2012, the 24th International Conference on Software & Systems Engineering and their Applications

Find out more

If you require further information, please contact

e: sales@integrate.biz

integrate systems engineering ltd

PO Box 9171

Sherborne

Dorset

DT9 9DX

United Kingdom

www.integrate.biz