






SRCM: A Semi Formal Requirements Representation Model Enabling System Visualisation and Quality Checking

Mohamed Osama¹, Aya Zaki-Ismail¹, Mohamed Abdelrazek¹, John Grundy² and Amani Ibrahim¹

¹Information Technology Institute, Deakin University, 3125 Burwood Hwy, VIC, Australia

²Information Technology Institute, Monash University, 3800 Wellington Rd, VIC, Australia

{amohamedzakiism, mdarweish, mohamed.abdelrazek, amani.ibrahim}@deakin.edu.au, john.grundy@monash.edu

Keywords: requirement representation, requirement modeling, requirement engineering, quality checking.

Abstract: Requirements engineering is pivotal to the successful development of any given system. The core artifact for such phase is the requirements specification document. Requirements can be specified in informal, semi-formal, and formal notations. The majority of the requirements across many fields and domains are written natural language. However, natural language is inherently ambiguous and imprecise and the requirements cannot be automatically validated. Formal notations on the other hand enable automated testing and validation but is only comprehensible by experts and requires rewriting the requirements. Semi-formal notations strikes a good balance between comprehension and checking for several systems. However, the majority of the existing representation models mandates the requirements to be (re)written to adhere to certain templates. They also do not support automated checking. In this paper, we present SRCM –a semi-formal requirements representation model based on a comprehensive requirements capturing model (RCM) that does not enforce much limitations on how the requirements can be written. We also provide an automated approach to construct SRCM from RCM. In addition to providing a unified visualisation of the system entities and relations between the requirements key components, SRCM also enables automated quality checking on the requirements.

1 INTRODUCTION


Detecting quality issues (i.e., inconsistencies, incompleteness and incorrectness) at an early stage, in system requirements specifications, improves the quality of the developed system. In addition, it minimises the overall development time and cost [Kamalrudin et al., 2011]. Formal methods and model checking are used for requirements verification in software engineering to detect such issues [Ghosh et al., 2016, Konrad and Cheng, 2005]. To benefit from these approaches, requirements should be represented in a suitable formal notation. However, requirements formalisation is a difficult task that requires experience in both mathematics and the system domain of interest [Buzhinsky, 2019, Kamalrudin et al., 2011]. Another research direction is detecting


quality issues at the semi-formal representation level. Semi-formal requirements are typically represented in graphical notations (i.e., helping users to better understand systems –specially large ones). Existing techniques are limited to detecting quality issues by comparing the semi-formal model against a reference (e.g., system domain or model repository [Kamalrudin et al., 2011]). This is primarily because they are built upon either widely known semi-formal models (e.g., UML) or on a proposed model (e.g., [Kamalrudin et al., 2011]) that provides abstract view of the system (i.e, behavior, structure, etc.).


Complementary to existing work, we propose a semi-formal model –system requirements capturing model (SRCM)– visualising the system requirements in a unified view. This model is designed to enable a better understanding of textual requirements and the relations among them, in addition to automatically detecting a wide range of quality issues.


SRCM is an extension to requirement capturing model (RCM) [Zaki-Ismail et al., 2020]. RCM is a comprehensive model developed for representing behavioural system requirements by supporting their

^a <https://orcid.org/0000-0000-0000-0000>

^b <https://orcid.org/0000-0000-0000-0000>

^c <https://orcid.org/0000-0000-0000-0000>

^d <https://orcid.org/0000-0000-0000-0000>

^e <https://orcid.org/0000-0000-0000-0000>

key properties. RCM encapsulates semi-formal and formal semantics. However, it is created to represent only one requirement and does not capture inter-requirements relations. We build SRCM to aggregate and unify the given system requirements (RCMs) in a model that shows the relations among them and enables the automated detection of quality issues.

2 RELATED WORK

The main three approaches for representing requirements in semi-formal notations are: controlled natural language (CNL), graphical representations, and mathematical representations [Marko et al., 2015].

CNL aims at writing syntactically correct requirements adhering to a given set of templates and/or boilerplates [De Gea et al., 2012]. Several types of templates and boilerplates (e.g., the EARS approach [Lúcio et al., 2017]) are present in the literature.

Farfeleder et al. [Farfeleder et al., 2011] developed an approach relying on boilerplates along with a domain ontology. A case study in [Stålthane and Wien, 2014] provides an evaluation of their work. This study showed that the use of such tools helped develop a requirement set that is consistent and unambiguous [Stålthane and Wien, 2014]. Such approaches however, require the presence of an ontology.

In [Kamalrudin et al., 2017] [Kamalrudin et al., 2011], MaramaAIC tool is proposed to support consistency management and requirements validation using semi-formal essential use cases (EUC). First, the use case models are extracted from the NL-requirements. Then, the abstract interactions of the EUC are compared to a repository of patterns. Consequently, a mismatch indicates a quality issue.

Nguyen et. al. in [Nguyen et al., 2016] proposed an ontology-based integrated framework for verifying goal use cases. They developed a tool called GUI-TAR. It takes textual requirements and transforms them into Goal use-cases for automatic reasoning and issues detection. A similar approach is presented in [Corea and Delfmann, 2017] for business processes verification, where business rules are specified as a logic program, and an ontology reasoner is used to discover model elements violating these rules.

Verification of UML Class/OCL model through (semi)formal notations has been discussed in the literature. Truong et. al. [Truong and Souquière, 2004] presented the transformation of UML class model into the B method. Then, the consistency of the class model is verified against UML well-formedness rules. In which, the rules are transformed into the invariant of B abstract machine. Similarly, Cabot et. al. [Cabot and Teniente, 2009] proposed an incremental verifi-

cation of UML Class/OCL model through (Constraint Satisfaction Problem) CSP. They illustrated that verification of constraints after every structure event (Entity Insertion, Attribute update, Entity Deletion, etc.) is costly and inefficient. They proposed the PSEs concept (Potential Structure Events). PSEs are events causing constraints violation. In this technique, PSEs for every integrity constraint are recorded and instances of entities and relationships are incrementally verified.

Model Based Systems Engineering (MBSE) like SysML [Mhenni et al., 2014] and Capella [Roques, 2016] languages are popularly used in multiple industries [Azzouzi et al., 2019]. However, they encounter considerable limitations concerning formal handling of requirements, and thus the automatic verification and validation. In addition, there is a wide gap between the early design phases (e.g., requirement capture) and the subsequent detailed design phases (i.e., when disciplinary modeling and simulation are involved) [Azzouzi et al., 2019]. A common issue within the highlighted semi-formal notations tools and approaches is that no automated checking support is provided [Azzouzi et al., 2019]

3 REQUIREMENTS CAPTURING MODEL (RCM)

RCM is a semi-formal representation model proposed in [Zaki-Ismail et al., 2020] for capturing behavioural system requirements. RCM defines the components constituting any NL-requirement specification and their breakdowns.

Figure 1 provides the compact high level architecture of RCM.

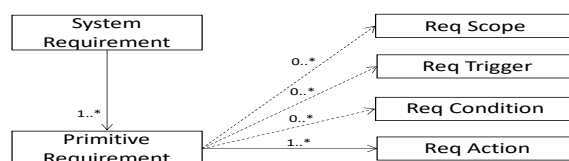


Figure 1: Compact High-level Architecture of RCM

RCM defines system requirements as a set of requirements R . Each requirement R_i , may have one or more primitive requirements PR where $\{R_i = \langle PR_n \rangle \text{ and } n > 0\}$. Each PR_j represents only one sentence, and may contain conditions, triggers, actions and scope – in the NL-requirement. RCM captures primitive requirements of the same requirement separately and stores them as one unit. A primitive requirements consists of four main components:

- Trigger: an event that automatically initiates/fires

action(s) whenever it occurs (e.g., when *emergency button is pressed*).

- **Condition:** constraints that should be checked explicitly by the system to allow action(s) to happen (e.g., if aircraft data is unavailable).
- **Scope:** provides the circumstances under which: (1) condition(s) and trigger(s) shall be valid – called “pre-conditional scope”, and (2) action(s) should occur – called action scope.
- **Action:** holds the task that should be executed in response to trigger(s) and/or constrained by condition(s) (e.g., the communication system shall sustain telephone contact with 10 callers). A primitive requirement with an action component only is a factual rule expressing system factual information (e.g., “The duration of a flashing cycle is 1 second”).

The core part of these components is the Predicate including: the operands, the operator and negation flag/property. For example, in the condition If X exceeds Y, “X” and “Y” are the operands and “exceeds” is the operator.

RCM avoids loss of information, by preserving logical relations among components of the same type in the interior nodes of a tree structure (the most suitable structure), where the components themselves are stored in the leaves.

4 System Requirements Capturing Model (SRCM)

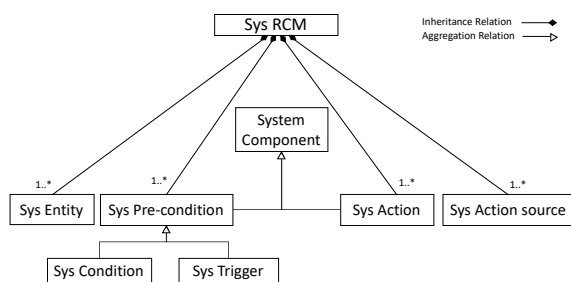


Figure 2: High-level Architecture for SRCM

SRCM is a graph-based extension of RCM that unifies the representation of all the requirements of a given system in one integrated model. It enables (1) visualising the relations among system requirements, and (2) detecting quality issues in an automated manner. SRCM consists of four main components: system entities, system pre-conditions, system actions and action sources as indicated in Figure 2.

System Entities are all the unique domain entities for

a given system. For each entity, the following properties are defined:

- **Domain name:** the name used to refer to such entity within the given system
- **Value type:** the type of the values assigned to this entity. It may be either atomic (e.g., number, boolean) or system dependent (system value).
- **Possible values:** list of the system values in case the value type is system dependent.
- **Affecting entities:** system entities that may cause a change to this entity when they change.
- **Affected entities:** in contrast to the previous list, it contains the system entities that may be changed as a result of a change in this entity.

Pre-conditions hold the prerequisite(s) required for an action to occur. A pre-condition may be a system-condition or a system-trigger. Pre-conditions are modeled as system components in SRCM as indicated in Figure 2. Figure 3 shows the architecture of different system component structures. The system component structurally visualises the relations between the contributing system entities in the component, by showing links to: (1) the primary entity (the entity upon which the system component is defined) and (2) the value source entity (the entity where the system component gets its value from). A value may have one of the following structures:

- **External value:** the primary entity is compared to the value of another entity (e.g., If X exceeds Y) as in Figure 3.(a).
- **Internal value:** the primary entity is compared to a related domain dependent system value (e.g., If the Regulator Mode equals [NORMAL]) as in Figure 3.(b).
- **Atomic value:** the primary entity is compared to atomic values as numbers, boolean, etc (e.g., If the Regulator Interface Failure is set to False) as in Figure 3.(c).
- **Aggregated value:** the primary entity is compared against the aggregation of some system entities that may include the primary entity itself (e.g., if the calculated distance is less than $(\frac{\text{current speed}}{3}) * t2$) as in Figure 3.(d).
- **Aggregated entity:** the primary entity is the aggregation of multiple entities that are compared to an atomic value (e.g., if (S1 + S2) exceeds 3). If the aggregated entities are assessed against a system value then it would match the aggregated value structure in Figure 3.(e).

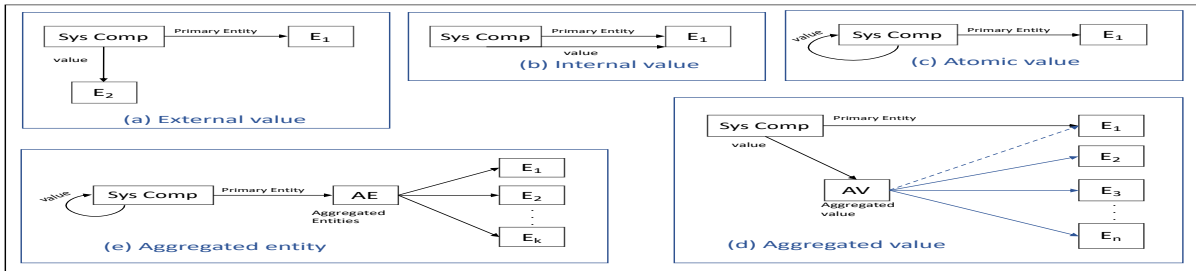


Figure 3: Different structures of a system component

Action represents the action or change that should happen in response to its precondition(s) being satisfied. It corresponds to RCM requirements action. An action can be in any of the various structures applicable to a pre-condition as in Figure 3.

Action source connects each action to its corresponding preconditions within the system as indicated in Figure 4. It also holds any relations (e.g. AND/OR) between the pre-conditions of an action.

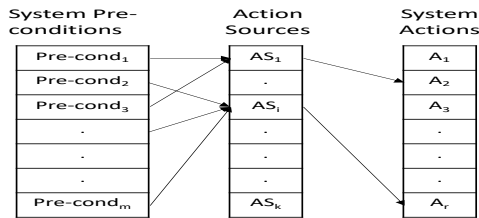


Figure 4: System action sources

5 SRCM CONSTRUCTION APPROACH

SRCM is constructed by extracting the required information from each RCM structure. The three main goals of the automatic construction are: (1) identify unique system entities, (2) identify unique system components and (3) construct system level mapping. These goals are achieved incrementally through an iterative approach.

5.1 Entities Identification

Each RCM structure is addressed apart to extract all the system entities from each component. To achieve this, the arguments of each component in an RCM structure are classified into entities and values based on domain and English characteristics as follows:

- Entities are either (1) domain variables following a special format (e.g., preceded by "RCMVAR...", "RCMCAL..." and "RCMTech..."), or (2) expressed in English noun phrases.

- Values, similarly to entities, are either (1) domain values with a special format (e.g., preceded by "RCMVAL..." in RCM), or (2) expressed in English adjective, adverb, or cardinal number.

The extracted entities are then added to the entire system entities list in case they are new (not previously extracted from another component). According to the domain variables, unique items are added to system entities. On the other hand, English entities should pass similarity checking with the existing English entities in system entities to assure uniqueness before adding them. Each unique entity, domain/English entity, is assigned a unique Id. All encountered English terms are grouped in a side structure, similar terms referencing the same entity, for similarity checking purpose.

5.2 Components Identification

The aim of this process is identifying the unique system component sets (pre-conditions and actions sets). First, the system component type of the RCM component under investigation is identified based on the RCM component type (i.e., condition \rightarrow system condition, trigger \rightarrow system trigger and action \rightarrow system action). Second, the component breakdowns, primary entity and the appropriate value structure, are identified. RCM components provide LHS (left hand side), RHS (right hand side) and relation(s) (e.g., in the component "the rain signal is ON", the LHS: "the rain signal", the RHS: "ON", and the rel: "="). To identify breakdowns of a system component from the RCM component.

The relation controlling the entities (i.e., aggregating relation) and the component (e.g., $=$, \neq , $<$, $>$, \leq , \geq) are mapped from the RCM component. After the complete construction of a system component, if the constructed component is new, it will be added to the relevant unique set and assigned a unique id.

5.3 Relational Mapping

In this process, a requirement can be defined on the system level. Since SRCM consists of unique sets

of system components, actions, and preconditions, a mapping is required to indicate which pre-conditions cause which action(s) (i.e., representing one requirement sentence done through action sources as discussed earlier). The mapping can be accomplished in a compact manner by storing the unique keys, each corresponding to a unique system component, of the preconditions and actions constituting one requirement sentence. In addition, The logical relations, AND/OR, among multiple system components of the same type (e.g., conditions, triggers, .etc) within the same requirement sentence are captured and mapped from the RCM structure.

6 SRCM AND SEMI-FORMAL CHECKING

The scope of this section is focused on illustrating how SRCM facilitates detecting requirements redundancy, ambiguity, inconsistency, and incompleteness quality issues. This is motivated by their impact and importance being ranked as the top attributes affecting requirements quality [Génova et al., 2013, Anuar et al., 2015, Kocerka et al., 2018]. Experts and studies applied in [Ott, 2012] have highlighted that ambiguity, inconsistency and incompleteness issues have the highest occurrence rates and are considered to be the most difficult requirements quality issues to resolve.

Redundancy within requirements is that, the same requirement appears more than once in the requirement specification document [Lami et al., 2004]. The problem is not in the redundancy itself, but in the consequences of having redundancy, especially in case of updating the document. For example, if a requirements document has two duplicate requirements in two places and only one of them is altered, the document will be inconsistent. Since requirements are mostly written by more than one engineer, there is a prospect of having the same requirement stated differently (e.g., "If X exceeds Y, Z shall be set to true" and "transition Z to true, if X is larger than Y"). Such case will not be easy to detect at the informal level of requirements. However, it can be easily detected in SRCM just by locating repeated action sources. The reason behind that is, storing duplicate system components of the same type only once in SRCM assigned with unique keys. Consequently, actions source of redundant requirements will map the same keys.

Ambiguity within requirements can lead to confusion, wasted efforts, and unnecessary rework. Ambiguity has a set of indicators such as vagueness, subjectivity, optionally, implicitly, .etc [Lami et al., 2004]. Currently, vagueness can be straightforwardly detected through SRCM construction. vagueness means that the requirement sentence contains word(s) with

more than one quantifiable meaning [Lami et al., 2004]. Thus, such issue can be flagged whenever a descriptive value is found in the construction process.

Inconsistency is the case where two or more requirements in the requirements document contradict each other [Lami et al., 2004]. Such type of issues can be detected through SRCM as follow:

- domain-independent approach: in such approach, inconsistencies can be detected for: (1) entities assigned/compared to values with different types. Assume the following requirement consisting of two sentences "If X exceeds Y, set X to True. Y is initialized with 2". There is an inconsistency in the assumed requirement since X is compared to an argument with a number type and assigned to value with a boolean type. Such type of inconsistency can be easily detected through SRCM construction. Since the value type of each entity is realized from the given set of requirements, in case that a contradicting type is encountered for an already type-bound entity, an inconsistency issue alerts for that entity highlighting the requirements contributing to the problem.
- domain-dependent approach: entities and values of the processed sentences are compared to the given domain ontology. In case of a mismatch (e.g., encountering an ineligible value type), the broken requirement sentence is flagged highlighting the mismatched detection. This type of checking is optional –can be achieved once the system domain is available.

Incompleteness Completeness is considered to be the most difficult of the specification attributes to define and incompleteness of specification the most difficult violation to detect [Lami et al., 2004]. SRCM can help in detecting missed information/arguments from a requirement component. Assume the following example "If X is True, Y shall be set". In such example, it is not determined to what value "Y" shall be set. In case that "Y" is a number type, then "Y" is eligible to many variations. Such type of incompleteness can be detected through SRCM construction if a component comprises at most one argument.

7 System RCM Visualization

The second goal of SRCM is automatically visualising requirements in an integrated and unified view. SRCM can provide two different views of the system.

Entities view: provides the complete dependency of either the entire requirements entities or only a selected subset. These dependencies show the effect of a change occurring in a given entities on other entities. The benefit of this view is providing a focused

interaction map of the (subset)requirements of interest. SRCM can provide this view based on the aggregated information, affecting and affected entities of each entity, for the system entities discussed before.

Requirements view: presents the SRCM components and their relations for either the entire requirements or only a selected subset of requirements. Representing textual requirements in a visual structure increases their expressiveness for all requirements stakeholders. Moreover, detected quality issues can be highlighted on the visualised view. Table 1 provides the graph notations both views.

Table 1: Graphical Representation of SRCM Views

View Type	Notation	Role
Entities View		in arrow: represents affecting entities
		out arrow: represents affected entities
		represents System Entity
Requirements View		in arrow: represents a contributing precondition in an action source
		out arrow: represents a response action of an action source
		solid line: represents the primary entity of a system component
		dashed line: represents the value source of a system component
		represents System Entity
		represents System Pre-Condition (Trigger/Condition)
		represents System Action
		represents System Action sources

8 CASE STUDY

The case study used refers to a hand crafted set of requirements specification for an elevator system. An elevator system is the service by which people can transfer from one building level to another. Usually in an elevator service a client can request an elevator, command a specific level, close/open the door and alert emergency. In order to show how SRCM can be constructed, and how it represents requirements in a unified integrated view, the results obtained with the SRCM are presented for six individual requirements presented in Table 2. Such requirements are developed with the following considerations (1) reflect the proposed cases of quality issues and (2) illustrate construction and visualisation aspects. For each, requirement, we discuss the SRCM construction process, and the quality issues detected as well. Finally, we present the corresponding requirements and entities views supported by a demonstration of the details in each view.

Table 2: Elevator Requirements

Req-Id	Requirement Text
Req1	<weight_thr> is initialized to 1000
Req2	When the direction status is [Up] or [down], the elevator motion status shall be changed
Req3	When the direction status is false, the elevator motion status shall be transitioned to [Idle].
Req4	If the weight of the elevator exceeds <weight_thr>, the elevator motion status shall be [Idle].
Req5	The elevator motion status shall be set to [Idle], if the weight of the elevator is larger than <weight_thr>.
Req6	If the weight of the elevator is heavy, the alert light shall be set to true.

The final SRCM breakdowns reflecting the elevator requirements are presented in Table 3.

Table 3: SRCM Breakdowns of Requirements in Table 2

SRCM BreakDowns		
System Entities	E1: RCMVAR_weight_thr	
	E2: the direction status	
	E3: the elevator motion status	
	E4: the weight of the elevator	
	E5: the alert light	
System Components	A1: RCMVAR_weight_thr = 1000	
	A2: the elevator motion status shall be changed	
	A3: the elevator motion status shall = [Idle]	
	A4: the alert light shall = true	
	System-Condition	C1: the weight of the elevator > RCMVAR_weight_thr
		C2: the weight of the elevator = heavy
System-Trigger	T1: the direction status = [Up]	
	T2: the direction status = [Down]	
	T3: the direction status = false	
Action Sources	AS1: [][] [A1]	
	AS2: [T1,T2] [] [A2]	
	AS3: [T3] [] [A3]	
	AS4: [] [C1] [A3]	
	AS5: [] [C1] [A3]	
	AS6: [] [C2] [A4]	

The corresponding conducted analysis, reasoning, and tracing for each requirement are as follows:

- Req1: this requirement is stored in RCM as an action component. In which, (1) <weight_thr> is replaced with "RCMVAR_weight_thr since" it is a domain variable and (2) "is initialized to" detected as "=" relation. By processing this component, one system entity is detected "RCMVAR_weight_thr", identified with the key "E1" and marked as the primary entity to the component. The type of the detected entity is identified as Number since it is assigned a value "1000" tagged as cardinal number.
- Req2: is decomposed into three components in the RCM "When the direction status is [Up]", "When the direction status is [down]" and "the elevator motion status shall be changed". The first two components classified as triggers are mapped to system triggers in SRCM and the last one is an action mapped to system action. This sentence

produced two system entities: (1) "the direction status" with the key "E2", possible values {[Up], [down]}, and type "SystemValue"; and (2) "the elevator motion status" with the key "E3". The action component is marked as incomplete since it contains only one argument in addition to the relation (i.e, the value of change for "the elevator motion status" is unknown). "E2" is stored as Affecting entity in "E3" since any change in "E2" can cause a change in "E3". In addition, "E3" is stored as affected entity at "E2" – to encapsulate information of each entity avoiding any overhead computations in any further checking processes.

- Req3: here, RCM has two component: trigger and action mapped to system trigger and action in SRCM respectively. The sentence results in two entities: (1) "the direction status" with the value "false" and type "Boolean". The entity is already identified before with the key "E2", but an inconsistency case is detected due to the types conflict "Boolean" and "SystemValue". (2) "the elevator motion status" is already recognized before with the key "E3", but it is assigned the type "SystemValue" and the value "[Idle]" here.
- Req4: this sentence has two RCM components condition mapped to system condition and action mapped to system action in SRCM. These components provide only one new entity "the weight of the elevator" and two previously detected entities "RCMVAR_weight_thr"; and the "elevator motion status". The entity "the weight of the elevator" is added to system entities with the key "E4" and type Number since it compared to an entity with the type Number. Similarly to Req2, "E1" and "E4" stored as affected entities at "E3" and "E3" stored as effected entity at "E4" and "E1".
- Req5: the sentence is paraphrasing for Req4. Thus, it does not add any new items to system entities and components of SRCM.
- Req6: the requirement has two RCM components

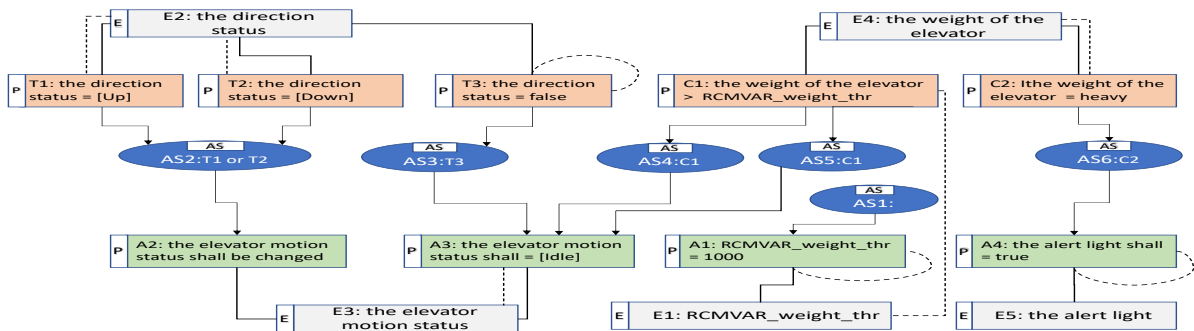


Figure 5: Requirements View of Requirements in Table 2

condition mapped to system condition and action mapped to system action. Only one entity "the alert light" is new and added to system entities with the key "E5" and the type "Boolean". In addition, the system condition "If the weight of the elevator is heavy" is flagged as ambiguous since it comprises a vague value "heavy". Finally, "E4" is stored as affecting entity at "E5" and "E5" stored as affected entity at "E4".

The relations "is initialized to", "is", "shall be transitioned to", "exceeds", "shall be", "shall be set to" and "is larger than" are transformed to "=", "≠", "≥", ">", "≤", "≠", "≠", and ">" respectively in the RCM. This help us to detect the unique system components as indicated in Table 3.

Finally, an action source for each requirement is created, including the redundant one, each is assigned a unique key and instantiated with three lists comprising the keys of the contributing conditions, triggers and action in the corresponding requirement as indicated in Table 3. After that, action sources with the same action keys for the three lists are marked as redundant (e.g., Req4 and Req5 flagged as redundant).

Requirements view and entities view of the elevator six requirements are shown in Figure 5 and Figure 6 respectively. This abstraction can help the user figure out missing requirements because any missing relation between the entities can be spotted. In Figure 6, it can be noticed that, "E3" is sensitive to "E1", "E2" and "E4"; and "E5" is sensitive to "E4".

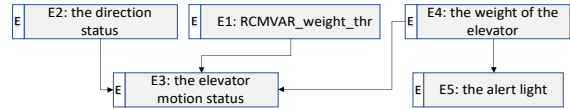


Figure 6: Entities View of Requirements in Table 2

9 CONCLUSION

In this paper, we propose graph-based semi-formal requirements representation model (SRCM)

that facilitates the visualisation of system requirements and the relations among requirements components and system entities. It also allows for quality checks to be applied automatically over the automatically constructed model views. The intrinsic structure of SRCM enables detecting wider range of quality issues (i.e., incompleteness, incorrectness and inconsistencies) compared to existing approaches – with far less effort compared to formal methods. It also keeps the requirements comprehensible by non technical stakeholders and even enhances the expressiveness of the requirements by showing a unified view of the system. We illustrated the construction and visualization of the SRCM using a case study highlighting how some quality issues can be detected.

REFERENCES

- Anuar, U., Ahmad, S., and Emran, N. A. (2015). A simplified systematic literature review: Improving software requirements specification quality with boilerplates. In *2015 9th Malaysian Software Engineering Conference (MySEC)*, pages 99–105. IEEE.
- Azzouzi, E., Jardin, A., Bouskela, D., Mhenni, F., and Choley, J.-Y. (2019). A survey on systems engineering methodologies for large multi-energy cyber-physical systems. In *2019 IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE.
- Buzhinsky, I. (2019). Formalization of natural language requirements into temporal logics: a survey. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 400–406. IEEE.
- Cabot, J. and Teniente, E. (2009). Incremental integrity checking of uml/ocl conceptual schemas. *Journal of Systems and Software*, 82(9):1459–1478.
- Corea, C. and Delfmann, P. (2017). Detecting compliance with business rules in ontology-based process modeling.
- De Gea, J. M. C., Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C., and Vizcaíno, A. (2012). Requirements engineering tools: Capabilities, survey and assessment. *Information and Software Technology*, 54(10):1142–1157.
- Farfeleder, S., Moser, T., Krall, A., Stålhane, T., Omoronyia, I., and Zojer, H. (2011). Ontology-driven guidance for requirements elicitation. In *Extended Semantic Web Conference*, pages 212–226. Springer.
- Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., and Moreno, V. (2013). A framework to measure and improve the quality of textual requirements. *Requirements engineering*, 18(1):25–41.
- Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., and Steiner, W. (2016). Arsenal: automatic requirements specification extraction from natural language. In *NASA Formal Methods Symposium*, pages 41–46. Springer.
- Kamalrudin, M., Hosking, J., and Grundy, J. (2011). Improving requirements quality using essential use case interaction patterns. In *Software engineering (ICSE), 2011 33rd international conference on*, pages 531–540. IEEE.
- Kamalrudin, M., Hosking, J., and Grundy, J. (2017). Mara-maic: tool support for consistency management and validation of requirements. *Automated software engineering*, 24(1):1–45.
- Kocerka, J., Krześlak, M., and Gałuszka, A. (2018). Analysing quality of textual requirements using natural language processing: A literature review. In *2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR)*, pages 876–880. IEEE.
- Konrad, S. and Cheng, B. H. (2005). Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering*, pages 372–381. ACM.
- Lami, G., Gnesi, S., Fabbrini, F., Fusani, M., and Trentanni, G. (2004). An automatic tool for the analysis of natural language requirements. *Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre*.
- Lúcio, L., Rahman, S., bin Abid, S., and Mavin, A. (2017). Ears-ctrl: Generating controllers for dummies. In *MODELS (Satellite Events)*, pages 566–570.
- Marko, N., Leitner, A., Herbst, B., and Wallner, A. (2015). Combining xtext and oslc for integrated model-based requirements engineering. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, pages 143–150. IEEE.
- Mhenni, F., Choley, J.-Y., Penas, O., Plateaux, R., and Hammadi, M. (2014). A sysml-based methodology for mechatronic systems architectural design. *Advanced Engineering Informatics*, 28(3):218–231.
- Nguyen, T. H., Grundy, J. C., and Almorisy, M. (2016). Ontology-based automated support for goal-use case model analysis. *Software quality journal*, 24(3):635–673.
- Ott, D. (2012). Defects in natural language requirement specifications at Mercedes-Benz: An investigation using a combination of legacy data and expert opinion. In *Proceedings of the 20th International Requirements Engineering Conference*, pages 291–296. IEEE Computer Society.
- Roques, P. (2016). Mbse with the arcadia method and the capella tool.
- Stålhane, T. and Wien, T. (2014). The dotd tool applied to sub-sea software. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 420–427. IEEE.
- Truong, N.-T. and Souquières, J. (2004). An approach for the verification of uml models using b. In *Proceedings. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004.*, pages 195–202. IEEE.
- Zaki-Ismail, A., Osama, M., Abdelrazek, M., Grundy, J., and Ibrahim, A. (2020). Rcm: Requirement capturing model for automated requirements formalisation.