Towards an Automated Requirements-driven Development of Smart Cyber-Physical Systems
Position paper

Jiri Vinarek Petr Hnetynka

Charles University in Prague, Faculty of Mathematics and Physics,
Department of Distributed and Dependable Systems,
Malostranske namesti 25, Prague, Czech Republic

vinarek@d3s.mff.cuni.cz hnetynka@d3s.mff.cuni.cz

The Invariant Refinement Method for Self Adaptation (IRM-SA) is a design method targeting development of smart Cyber-Physical Systems (sCPS). It allows for a systematic translation of the system requirements into the system architecture expressed as an ensemble-based component system (EBCS). However, since the requirements are captured using natural language, there exists the danger of their misinterpretation due to natural language requirements' ambiguity, which could eventually lead to design errors. Thus, automation and validation of the design process is desirable. In this paper, we (i) analyze the translation process of natural language requirements into the IRM-SA model, (ii) identify individual steps that can be automated and/or validated using natural language processing techniques, and (iii) propose suitable methods.

1 Introduction

Smart Cyber-Physical Systems (sCPS) are complex distributed decentralized systems of cooperating mobile and stationary devices closely interacting with the physical environment. Examples of sCPS include systems like smart home, smart traffic management, etc.

Designing and developing such a system is a quite complex task with many challenges. Mobility and distribution bring the high level of dynamism to the system, which has to be aware of changes in its environment. Openness and open-endness are other challenging issues resulting in needs that the designed system has to be able to tackle unanticipated changes and participants unknown at design time.

The traditional software design and development techniques have been shown unsuitable for such systems and novel approaches [8, 13, 14] have been proposed to tackle with the challenges. One of these promising approaches is Ensemble-Based Component Systems (EBCS) [1]. Using EBCS, the system is modeled and developed as a set of *ensembles*, i.e., dynamic cooperation groups of software components. Components are specified by their *knowledge* (i.e., component's attributes) and by a set of *processes* manipulating the knowledge.

The Invariant Refinement Method for Self Adaptation (IRM-SA) [2] is a design method targeting development of sCPS using EBCS. IRM-SA allows for a systematic translation of the system requirements written in natural language into the system architecture expressed as components and ensembles. Using IRM-SA, a designer gradually refines the initial requirements and iteratively builds a model that consists of so-called *invariants*. Invariants are then hierarchically decomposed and at the lowest level, they directly correspond to an implementation (in the DEECo component model [1], with which IRM-SA is currently tied).

Submitted to:
FESCA 2016

To speedup and ease the design process with IRM-SA, the guide [1] and graphical editor [2] have been created. The editor allows for editing of the constraints and performing several basic validations of the designed IRM-SA model. Additionally, skeletons of the implementation can be generated directly from the designed model. Even though the guide and editor exist, the whole process, i.e., translation of requirements into the IRM-SA model, is manual and it can be time-consuming and laborious. Additionally, as the requirements are expressed as a text in natural language, there is a danger of ambiguity and misinterpretation of them, which can result in a suboptimal design. Even more, designers can unintentionally miss important requirements.

The goal of this paper is to analyze the IRM-SA design process and identify particular steps, which can be, fully or at least partially, automated with the help of natural language processing tools. To achieve the goal, we use our experience gained with automated processing of textual use-cases, their verification and transformation into an implementation ([15, 16, 18]).

The paper is structured as follows: Section 2 explains the IRM-SA method and its inputs and outputs. Section 3 discusses steps of the IRM-SA method from the perspective of their automation and proposes solutions for them. Finally, Section 4 discusses related work while Section 5 concludes the paper.

2 IRM-SA explained by example

In this section, we briefly describe the IRM-SA method and its usage on an example (for a detailed description, please see the IRM-SA guide).

The experiment described in [6] proved that usage of IRM-SA represents a significant help in EBCS design and development. Participants of this experiment designed using IRM-SA an EBCS architecture with less errors than participants using another design method. Even though, the resulting architectures were not completely without errors, especially thank to different understanding of the input requirements provided as a text in natural language and thus, there is still space for improvements.

This is even more important, as one of the outputs of the IRM-SA method – the IRM-SA model – can be used not only at design time, but also during development and maintenance of the developed system. In particular, the IRM-SA model allows for traceability between purpose of each invariant (requirement) and its realization and therefore it is ideal for documentation and maintenance. Plus, as stated in [10], it is a mistake to understand requirements specifications as final and unchangeable and thus keeping up-to-date traceability links to requirements is quite important.

Additionally, the IRM-SA model is in the DEECo implementation employed for controlling self-adaptation of the system, i.e., the model captures multiple alternatives of the system architecture and the appropriate one is chosen based on actual situation.

To sum up, the IRM-SA model is one of the key artifacts of the developed system and its correctness is essential. Therefore, designers/developers would benefit from a tool which not only allows for easy creation of the model (the currently available editor allows for this) but also which would be able to (semi)automatically parse the textual requirements, generate parts of the model from the requirements, and validate individual actions performed by the designer/developer. To provide such a tool, natural language processing methods and tools have to be incorporated in the process.

[1] `http://svn.pst.ifi.lmu.de/ascens/guide/irm/`

[2] `https://github.com/d3scomp/IRM-SA`

2.1 IRM-SA method and model

The IRM-SA design method is an iterative top-down design approach. A designer has to perform the following steps in order to built the IRM-SA model from the requirement specification:

1. Find the top-level goals of the system and specify the top-level (abstract) invariants.
2. Find the components of the system (and their fields) by asking “which knowledge does each invariant involve and where is this knowledge obtained from?”
3. Decompose each invariant by asking “how can this invariant be satisfied?”
4. Separate the concerns of the abstract invariants into sub-invariants that correspond to (abstract) activities that can be done in isolation.
5. Compose invariants together by asking “why do I need to satisfy these invariants?”
6. In case of situation-specific requirements, try first to accurately capture the condition of being in one situation or another. Use the assumptions to do that. Then use OR decomposition to specify which invariants to satisfy in each situation.

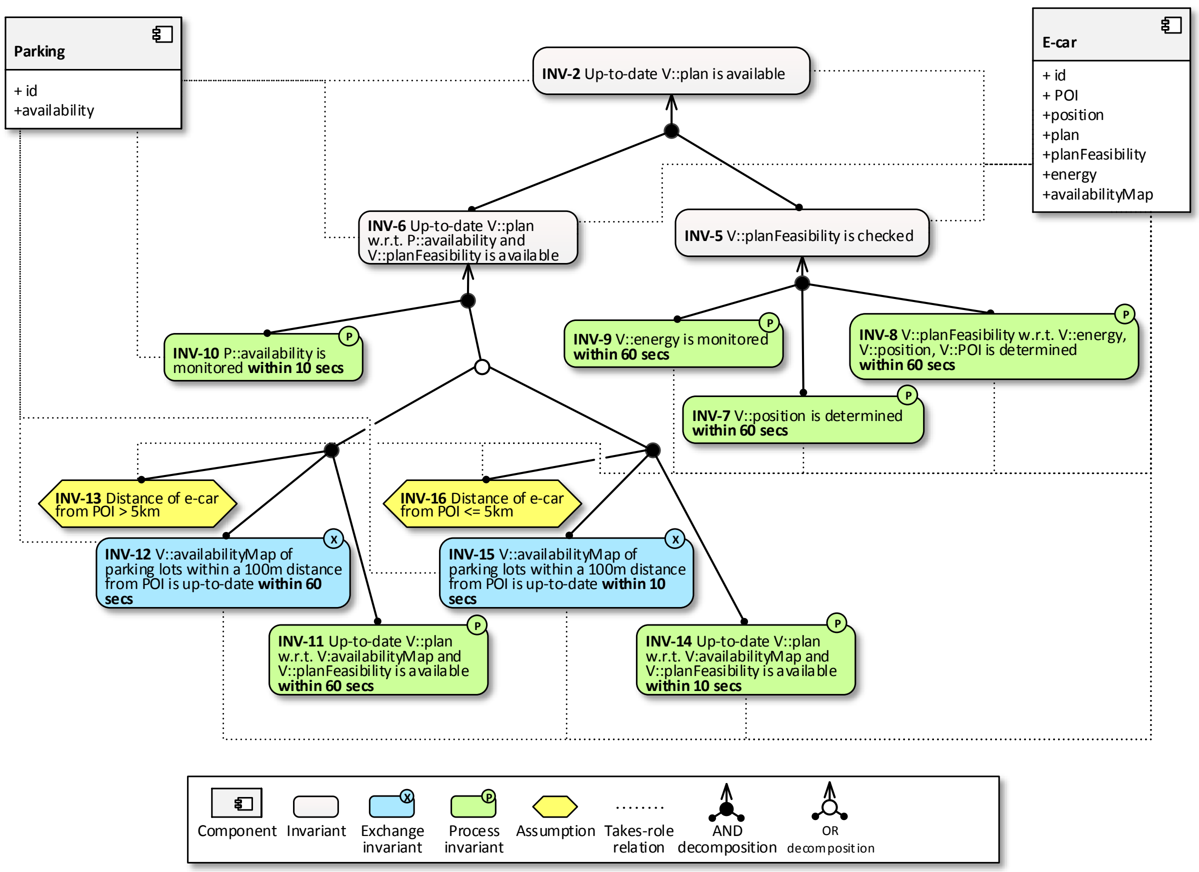
Parking
+ id
+availability
INV-2 Up-to-date V::plan is available
E-car
+ id
+ POI
+position
+plan
+planFeasibility
+energy
+availabilityMap
INV-6 Up-to-date V::plan w.r.t. P::availability and V::planFeasibility is available
INV-5 V::planFeasibility is checked
INV-10 P::availability is monitored within 10 secs
INV-9 V::energy is monitored within 60 secs
INV-8 V::planFeasibility w.r.t. V::energy, V::position, V::POI is determined within 60 secs
INV-7 V::position is determined within 60 secs
INV-13 Distance of e-car from POI > 5km
INV-16 Distance of e-car from POI <= 5km
INV-12 V::availabilityMap of parking lots within a 100m distance from POI is up-to-date within 60 secs
INV-15 V::availabilityMap of parking lots within a 100m distance from POI is up-to-date within 10 secs
INV-11 Up-to-date V::plan w.r.t. V::availabilityMap and V::planFeasibility is available within 60 secs
INV-14 Up-to-date V::plan w.r.t. V::availabilityMap and V::planFeasibility is available within 10 secs
P
X
Component
Invariant
Exchange invariant
Process invariant
Assumption
Takes-role relation
AND decomposition
OR decomposition

Figure 1: IRM-SA model for e-cars system

Figure 1 shows the IRM-SA model created for the electric car (e-car) navigation and parking system example; its requirement specification is in Figure 2. Both the specification and model are overtaken from the IRM-SA guide (and they were also employed in the experiment mentioned above). The model was created manually with the help of the IRM-SA model editor.

The highlighting and underlining in the requirements specification is included solely for the purpose of this paper and would not appear in an actual specification. Meaning of the highlighting/underlining is as follows:

- Underlined text refers to components and their attributes (double line for components, single line for attributes). Red color is used for the "E-car" component, blue one for the "Parking" component)
- Highlighted text is related to a particular invariant; colors correspond with colors in figure 1.
- Purple numbers in the requirements specification refer to corresponding invariants in the model.

Electric Car Navigation and Parking (ECNP)

Summary:
The main objective of this system is to allow e-cars to coordinate with parking stations and have an adequately up-to-date view of the availability of parking spaces at each time point. At the same time, e-cars should monitor their battery level and choose a different trip plan (e.g., which involves picking a parking place which is closer to the e-car) if the existing plan is not possible to follow any more.

Requirements:
The **general** requirements for the ECNP are:

1. Every e-car has to arrive to its place of interest (POI) and park within a radius of 100 meters. [2] In order to do that, every car needs to :
   (a) Continuously monitor its energy level (battery) ;[9]
   (b) Continuously monitor its position) ;[7]
   (c) Continuously assess whether its energy level would be enough to complete the trip based on the distance left to cover; [8]
   (d) Have a plan to follow, which is based on its energy level and on the available parking slots in the parking places near the POI .[11, 14]
2. Every parking place has to continuously monitor its availability level (e.g., in terms of available parking slots per time slot .[10]
3. The information regarding the availability of the parking slots has to be exchanged with the appropriate e-cars .[12, 15]

The **situation-specific** requirements of the ECNP are:

4. When an e-car is more than 5km far from the POI [16], it should update its plan at least once per 60 seconds .[14]
5. When an e-car is equal to or less than 5km far from the POI [13], it should update its plan at least every 10 seconds [11].

Figure 2: e-cars system requirements

3 Automation of IRM-SA

In this section, we analyze the IRM-SA method from the perspective of its automation. We identify individual steps that can be automated using natural language processing techniques, and propose suitable methods.

The individual goals of such an automation are: (i) (semi)automatically generate invariants in the IRM-SA model from the requirements document (and thus make synchronization and traceability between the requirements and IRM-SA model more robust and faster to obtain), (ii) (semi)automatically validate the resulting IRM-SA model.

3.1 Component identification

Components in EBCS design represent "smart" entities of the system. In our example, there are two types of components – `E-Car` and `Parking`. Both components and also their attributes are several times mentioned in the *requirements* as well as in the *summary*. Based on our experience with derivation of the domain model from textual specification [17], it seems possible to obtain a list of potential components in an automated fashion. Also, a similar approach is employed in [5], where authors retrieve UML class models from test cases.

Both names of the components and their attributes are in the requirement texts almost always represented as noun phrases, which in simple sentences appear as *subjects* or *objects* (either direct or indirect). To parse a sentence and identify its elements, the Stanford CoreNLP toolkit [11] is an ideal tool. For example, with the usage of the Stanford dependency parser on the sentence *"Every car needs to continuously monitor its energy level (battery)."*, we get the dependency graph showed in Figure 3. The parser returns a part-of-speech (POS) tag for each word (e.g., *VB* for verb in base form, *NN* for singular noun) and dependency relations between the words (e.g., *nsubj* for nominal subject or *dobj* for direct object). The resulting list of potential names of components and attributes contains `car` (nominal subject) and `energy level` (direct object). Unfortunately, this might not be sufficient (not all of the subjects/objects are components/attributes). A possible way to overcome this issue is to employ statistical classification techniques to learn the patterns from training data. Similarly, we have employed these techniques in [18].

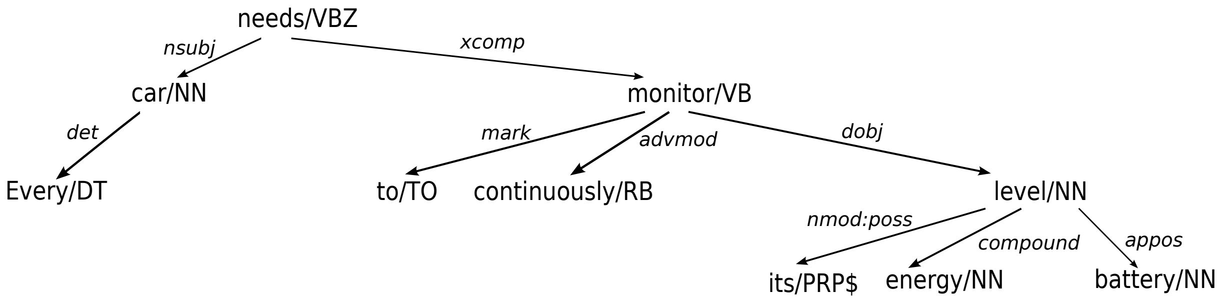

Figure 3: Dependencies and POS tags obtained from Stanford dependency parser

3.2 Component disambiguation

Another issue with the list of candidates for component/attribute names is that it may be ambiguous. Multiple noun phrases may refer to the same component, e.g., in our e-cars system specification the words "e-car" and "car" refer to the same component. Similarly, "parking station" and "parking place" or "energy level" and "battery". These ambiguities can be a sign of a poorly written specification and their replacement with the same word or phrase is advisable. On the other hand, such a situation may happen (especially if the specification is prepared by multiple authors and/or it evolves over time) and even more, in some cases, the use of different words for the same entity may be intentional, e.g., in a case of abbreviations and noun phrase shortenings ("place of interest" and "POI" or "trip plan" and "plan", etc.).

A distinction of these cases is not always clear and we expect that user will have to be involved in a decision about these ambiguities.

For example, the requirement in Figure 3 is written in a way which suggest that the phrase "energy level" and "battery" can be used interchangeably. This relation can be deduced automatically, as the word “battery” has been marked as appositional modifier (*appos*) of the word “level”. Another option for disambiguation might be employment of *string distance metrics* [4][3] to identify corresponding entities (e.g., “car” and “e-car”).

3.3 Invariant type identification

During applying the IRM-SA method, sentences in the *Requirements* section of the specification are rather directly translated into invariants of the IRM-SA model. However, the issue is to identify whether the particular sentence relates to the *abstract*, *process*, *exchange* or *assumption* invariant. It would be helpful, if the IRM-SA editor could automatically propose the invariant type.

Assumption invariants should be included only in the *situation-specific* section of the requirements specification and thus is easier to locate them (see the yellow highlighting in Figure 2 and the yellow invariants in Figure 1). Additionally, the particular sentences express a condition, which is necessary to detect and extract. To extract it, the dependency parser can be again utilized. To support it, tools for information extraction like Ollie[4] or OpenIE[5] can also be used as they are able to detect enabling conditions.

To distinguish between *process* and *exchange* invariants, it is necessary to analyze the main verb of the sentence (see the blue and green highlighting in Figure 2 and the blue and green invariants in Figure 1). With verbs such as “exchange” or “propagate”, there is a high chance that the sentence corresponds to exchange invariant, while verbs “have”, “monitor”, “assess”, “obtain”, “acquire” or “determine” usually denote a process invariant. A direct solution would be a simple comparison of the particular verb with a predefined set of verbs but it would be rather limiting. Instead, a suitable approach is to classify verbs according to their meaning, which is taken from WordNet[12]. WordNet is a large lexical database of English nouns, verbs, adjectives and adverbs. In the database, synonyms are grouped together forming so-called *synsets*. The synsets are interlinked according to their relations, forming network of related words and concepts. Multiple WordNet similarity measures were proposed and their implementation is available[6] and can be used for the *process* and *exchange* invariants identification.

Finally, sentences containing additional sub-requirements can be marked as *abstract* invariants (the gray highlighting and the gray invariants in Figures 1 and 2).

3.4 Knowledge flow recognition

One of the key IRM-SA ideas is that each invariant (with the exception of assumption ones) represents a computation that produces *output knowledge* given a particular *input knowledge* such that the invariant is satisfied (as stated in [2]).

For example, in the invariant deduced from the requirement 1(d) in Figure 2, the *energy level* and *POI* of the *vehicle* and the *available parking slots* from the *parking places* serve as the input parameters for computation of the vehicle’s *plan* (all possible parameters, i.e., component attributes, are already known as they were identified in the previous phases). Schematically, it can be written

```
V::energy, V::POI, P::availability -> V::plan.
```

[3]an implementation available at `http://secondstring.sourceforge.net/`

[4]`https://github.com/knowitall/ollie`

[5]`https://github.com/knowitall/openie`

[6]`http://search.cpan.org/dist/WordNet-Similarity/`

Such an abstraction of the requirements to input and output parameters allows for easier reasoning about the invariants and it is employed in subsequent sections.

However, an issue is how to identify which parameters are input and which output. A straightforward automatic approach for distinguishing input parameters from the output ones is an iteration starting from the simple invariants and taking into account types of parameters already distinguished from the previous iterations. The approach starts with the process invariants having only single parameter. Such a parameter must be an *output* one (otherwise the invariant would not produce any knowledge and the above mentioned IRM-SA idea would not hold). Examples of such invariants are requirements 1(a) and 1(b). Next, if a single attribute is present in multiple invariants, it can be assumed that it serves as an *input* parameter. Nevertheless, this is only an assumption and thus a fully automated approach is hard to achieve. A possible solution is to use an assisted iterative approach, in which a tool identifies input and output parameters and the human designer confirms/reverts the decisions.

In some cases, computation associated with an abstract invariant may not have all the parameters precisely specified, as the particular parameters are unknown yet. They may be specified in the child invariants and from the view of the higher-level invariant, they can be seen as an implementation detail. In such cases, we use the notation `V::?` to mark that the component `V` participates in the invariant, but the specific attribute is specified later in a child invariant. The final assignment of the parameter is up to the designer.

3.5 Invariant refinement and composition

In EBCS, communication between components is implicit via their knowledge sharing, which is conveyed via ensembles. An ensemble is thus specified via a condition determining when components are part of the particular ensemble and via knowledge that has to be interchanged.

Let us again assume the requirement 1(d) with the parameter abstraction

```
V::energy, V::POI, P::availability -> V::plan.
```

As the parameters come from different components (`V` and `P`) but the computation can be performed only in a single component, it is clear that the invariant has to be *refined* as a *composition* of several invariants, from which at least one is an *exchange* invariant (in the implementation, the exchange invariants results in the ensemble definition). Such situations can be rather easily detected automatically based on the parameters' owners.

Nevertheless, refinement and composition of the *abstract* invariants is more difficult as they represent high-level goals and can intentionally abstract from some "implementation details", i.e., omit some attributes. For example, in Figure 1, composition of the invariants 5 and 6 means that the trip plan is computed first without any knowledge about availability of the parking places (the output parameter `V::planFeasibility` in the invariant 5) and then it is made more specific with information about the availability (the invariant 6). Different composition of the lower-level invariants would lead to a completely different behavior.

Another issue in the automatic composition of invariants is a reasoning about *situation-specific* requirements, which have to be grouped together according to a requirement they belong to. The grouping is performed based on their output parameters and can result in duplication of invariant subtrees. However, identification of the right subtree to be duplicated is not straightforward. Both these issues are very hard to solve automatically and we plan to further investigate possibilities of their automation in more detail.

3.6 Model validation

With the abstraction of invariants described in 3.4, the IRM-SA model can be automatically validated according to the knowledge flow. In particular, following checks can be performed:

- Configurations with missing input parameters can be discovered (i.e., an invariant producing the particular attribute is not included in the configuration due to a missing dependency relation).
- Configurations with multiple invariants writing to the same attribute can be detected.
- Also, detection of unused output parameters or unused attributes can be performed.

All of these checks may point to flaws in the model and/or specification and discover them early.

4 Related work

As far as we know, there are no attempts to automatize requirements processing for EBCS design. Nevertheless, a related approach is described in [3], in which authors propose an approach called NPL-KAOS that can automatically obtain a KAOS model from large volume of literature (KAOS [9] is a goal-oriented requirement engineering method and it was one of inspirations for the IRM-SA method). With the use of natural language processing tools and text mining techniques they process abstracts of scientific publications. First, they detect goal-oriented keywords and then they use the Stanford parser to tag semantic structures. From obtained semantic trees they extract goals and finally organize them into taxonomies. The taxonomies are used to define relations between goals and this way they simulate the process of refinement. Similarly to our approach, the authors try to automatically derive a model from textual data which would serve for purposes of requirements engineering. However, their problem is different. Their main goal is to help requirements engineer during the early stages of goal elicitation by extraction of main concepts from the large body of research abstracts. Although the extraction process may miss some goals in the single abstract, with large number of abstracts they can count on the fact that the goal will be at the end noticed. Contrary, our method is intended to process a single specification and therefore cannot reckon on this effect.

In [5], the authors (semi-)automatically derive a UML model and OCL constraints from a specification and test cases, which are both written in natural language. They employ formal methods to verify correctness of the derived design. First, grammatical analysis is used to derive UML class diagrams from the test cases. Then, the behavior of test cases is inspected and the UML sequence diagrams are derived. In the next step, OCL constraints are deduced from the requirements and test cases. Finally, verification of static aspects (UML class diagrams and OCL invariants) and dynamic aspects (satisfaction of specified method pre-conditions and post-conditions) is checked. As in our approach, authors use the Stanford parser to get dependencies from the sentences. They also employ WordNet (in their case, to distinguish components of the system and actors). The main difference is that we directly target sCPS design and EBCS and therefore include an identification of *process* and *exchange* invariants, adaptability, etc.

5 Conclusion

In the paper, we have analyzed the IRM-SA design method with respect to its possible automation with the help of natural language processing methods and tools. We have identified steps that can be automated and sketched solutions. As the automated natural language understanding is generally still a challenging issue, the full automation is hard (and in many cases impossible) to achieve. Thus, we target a semi-automated system that guides the human designer, recommends solutions and validates the designers actions.

Currently, we plan to implement all the identified proposed solutions, to integrate them to the existing IRM-SA editor and to validate the resulting system on a real-life case-study.

Even though the proposed approaches are tailored to IRM-SA (which is currently tied with the DEECo component model), they can be reused in different contexts. The IRM-SA method itself can be without changes applied to another ensemble-based component model (e.g., Helena [7]) and the approaches proposed in this paper can be applied in tools for the KAOS method or similar ones.

6 Acknowledgement

This work was partially supported by the project no. LD15051 from COST CZ (LD) programme by the Ministry of Education, Youth and Sports of the Czech Republic and partially supported by Charles University institutional funding SVV-2016-260331.

References

[1] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit & Frantisek Plasil (2013): *DEECO: An Ensemble-based Component System*. In: *Proceedings of CBSE 2013, Vancouver, Canada*, ACM, pp. 81–90, doi:10.1145/2465449.2465462.

[2] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit & Frantisek Plasil (2015): *The Invariant Refinement Method*. In: *Software Engineering for Collective Autonomic Systems*, *LNCS* 8998, Springer, pp. 405–428, doi:10.1007/978-3-319-16310-9_12.

[3] E. Casagrande, S. Woldeamlak, W.L. Woon, H.H. Zeineldin & D. Svetinovic (2014): *NLP-KAOS for Systems Goal Elicitation: Smart Metering System Case Study*. *IEEE Transactions on Software Engineering* 40(10), pp. 941–956, doi:10.1109/TSE.2014.2339811.

[4] William W. Cohen, Pradeep Ravikumar & Stephen E. Fienberg (2003): *A comparison of string distance metrics for name-matching tasks*. In: *Proceedings of IIWeb-03, Acapulco, Mexico*, pp. 73–78.

[5] Rolf Drechsler, Mathias Soeken & Robert Wille (2012): *Formal Specification Level: Towards verification-driven design based on natural language processing*. In: *Proceedings of FDL 2012, Vienna, Austria*, IEEE, pp. 53–58.

[6] Ilias Gerostathopoulos, Tomas Bures, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, Frantisek Plasil & Noël Plozeau (2015): *Self-Adaptation in Cyber-Physical Systems: from System Goals to Architecture Configurations*. Technical Report D3S-TR-2015-02, Charles University in Prague, Faculty of Mathematics and Physics, Department of Distributed and Dependable Systems.

[7] Rolf Hennicker & Annabelle Klarl (2014): *Foundations for Ensemble Modeling — The Helena Approach*. In: *Specification, Algebra, and Software*, *LNCS* 8373, Springer, pp. 359–381, doi:10.1007/978-3-642-54624-2_18.

[8] Matthias Hölzl, Axel Rauschmayer & Martin Wirsing (2008): *Software Engineering for Ensembles*. In: *Software-Intensive Systems and New Computing Paradigms*, *LNCS* 5380, Springer Berlin Heidelberg, pp. 45–63, doi:10.1007/978-3-540-89437-7_2.

[9] Axel van Lamsweerde (2008): *Requirements Engineering: From Craft to Discipline*. In: *Proceedings of SIGSOFT'08/FSE-16, Atlanta, USA*, ACM, pp. 238–249, doi:10.1145/1453101.1453133.

[10] Craig Larman (2004): *Applying UML and patterns: an introduction to object-oriented analysis and design and the unified proces*, 3rd edition. Prentice-Hall.

[11] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard & David McClosky (2014): *The Stanford CoreNLP Natural Language Processing Toolkit*. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Baltimore, Maryland*, pp. 55–60, doi:10.3115/v1/P14-5010.

[12] George A. Miller (1995): *WordNet: A Lexical Database for English*. *Communications of the ACM* 38(11), pp. 39–41, doi:10.1145/219717.219748.

[13] Brice Morin, Franck Fleurey & Olivier Barais (2015): *Taming Heterogeneity and Distribution in sCPS*. In: *Proceedings of SEsCPS 2015, Firenze, Italy*, ACM, pp. 40–43, doi:10.1109/SEsCPS.2015.15.

[14] Ivan Ruchkin, Bradley Schmerl & David Garlan (2015): *Architectural Abstractions for Hybrid Programs*. In: *Proceedings of CBSE 2015, Montreal, Canada*, ACM, pp. 65–74, doi:10.1145/2737166.2737167.

[15] Viliam Simko, David Hauzar, Petr Hnetynka, Tomas Bures & Frantisek Plasil (2014): *Formal Verification of Annotated Textual Use-Cases*. *The Computer Journal* 58(7), pp. 1495–1529, doi:10.1093/comjnl/bxu068.

[16] Viliam Simko, Petr Hnetynka & Tomas Bures (2010): *From Textual Use-Cases to Component-Based Applications*. In: *In proceedings of SNPD 2010, London, UK*, *SCI* 295, Springer, pp. 23–37, doi:10.1007/978-3-642-13265-0_3.

[17] Viliam Simko, Petr Kroha & Petr Hnetynka (2013): *Implemented Domain Model Generation*. Technical Report D3S-TR-2013-03, Charles University in Prague, Faculty of Mathematics and Physics, Department of Distributed and Dependable Systems.

[18] Jiri Vinarek, Petr Hnetynka, Viliam Simko & Petr Kroha (2014): *Recovering Traceability Links Between Code and Specification Through Domain Model Extraction*. In: *Proceedings of EOMAS 2014, Thessaloniki, Greece*, *LNBIP* 191, Springer, pp. 187–201, doi:10.1007/978-3-662-44860-1_11.