Automatic Model Generation to Diagnose Autonomous Systems

Jorge Santos Simón[1] and **Clemens Mühlbacher**[1] and **Gerald Steinbauer**[1]
[1]Institute for Software Technology
e-mail: {jsantos, cmuehlba, gstein}@ist.tugraz.at

Abstract

Autonomous systems' dependability can be improved by performing diagnosis during run-time. This can be achieved through model-based diagnosis (MBD) techniques. The required models of the system are for the most part handcrafted. This task is time consuming and error prone. To overcome this issue, we propose a framework to generate formal models out of natural language documents, such as technical requirements or FMEA, using natural language processing (NLP) tools and techniques from the knowledge representation and reasoning (KRR) domain. Therefore, we aim to enable the usage of MBD in autonomous systems with few extra burden. So doing, we expect a significant increase in the usage of MBD techniques on real-world systems.

1 Introduction

Dependability is a key feature of modern autonomous systems. It can be achieved by sound design and implementation, thorough testing and runtime diagnosing. To date, all these processes are still not completely automated and need substantial manual work. However, all these fields can greatly benefit from the use of model-based techniques. Design and implementation can be greatly improved through model-driven engineering, as stated in [1]. Model-based testing (MBT) has been demonstrated [2] to outperform traditional testing techniques in both invested time and number of errors found. Model-based diagnosis (MBD) is the main target of this work. It has been successfully used in industrial settings [3], reducing the need for human intervention. Although it has being increasingly adopted in recent years, we believe that its full potential is still to be developed.

All model-based techniques require appropriate models of the system. As stated in [4; 5], creating these models is the most prevalent limiting factor for their adoption. To overcome this barrier, we propose a method that automates models creation from the documents used during the system design. These comprise requirements documents, architectural designs, FMEA and FTA, among others. The content of these documents is often given in natural language and in semi-structured form and lacks a common semantics. Thus, the contained information is not accessible for a computer. However, advances in natural language processing (NLP) and the availability of common sense and domain-specific knowledge bases (e.g. Cyc [6], RoboEarth [7]) make semi-automated derivation of models possible. Despite recent advances on this area [8; 9; 10], most techniques focus on very specific applications of the generated formal models. Thus, we pose the problem of generating a common knowledge base as an intermediate representation with a well defined semantics out of documents used during the system design process. From this central repository, different algorithms can extract different formal models for particular needs. We believe that this work can increase the acceptance of model-based techniques and broaden their use.

The motivation for this work came during the development of a model-based diagnosis and repair (MBDR) system for an industrial application. The aim is to improve the dependability of a fleet of robots that automatically deliver goods in a warehouse. As stated in [11], even minor failures often prevent a robot from accomplishing its task, decreasing the overall performance of the system. Moreover, the frequent need of human intervention increases costs and customer dissatisfaction. Using MBDR techniques, many of these failures can be automatically handled, allowing the robot to remain on service, perhaps with its capabilities gracefully degraded [12; 13]. In extreme cases, diagnosing a failure on time can prevent robot behaviors harmful for humans, itself or other elements in the environment.

Confronted with the lack of any formal model of the system, we were forced to manually code the models we need. However, this is both a time-consuming and error prone task, and also impose a maintaining burden as the system evolves. Accordingly, we believe that a mostly automated approach is not only convenient for the intended project but can also help extending the use of MBDR techniques to other projects and domains. Following this idea, we propose a framework that, in a first step, gathers the information from the project together with domain and common-sense knowledge in a machine-understandable knowledge base. Then, a suit of algorithms can extract formal models from this knowledge base for particular purposes. Though our aim is to automate the process as much as possible, human assistance will be requested whenever some pieces of information are missing or contradictory [14; 15].

The novelty of our proposal is two-fold: first, we emphasizes the usability of the resulting models for MBD. Second, we aim to integrate all the sources of information typically available in an industrial development process, such as requirements, architecture, and failure modes. As a result, we expect to boost the range and applicability of the automat-

ically generated models. To better illustrate the proposed framework, we will use a small running example extracted from a real-world application. It comes to the robot's box loading operation, performed by the robot's load handling device (LHD).

The remainder of the paper is organized as follows: Related research on model generation is discussed in Section 2. Section 3 provides an overview of the proposed process. Section 4 describes the inputs used, while Section 5 describes the proposed NLP and KRR tool-chain to interpret them. Section 6 provides an example of an output model and its use for MBD. Finally, Section 7 summarizes the presented framework and discusses future work.

2 Related research

We start the brief discussion of related research with the work using NLP methods to derive models. The work of [9] uses NLP methods to derive a formal model out of requirements. This formal model can afterwards be transformed into different representations to test or synthesize the system. The method proposed in [10] uses NLP methods to derive design documents (class diagrams, etc.) out of requirements. These design documents can afterwards be used to implement the system. The authors of [8] proposes a method to extract action receipts from websites. These action receipts comprises the desired behavior in order to achieve a given goal. The method use how-to instructions and NLP tools to derive an action receipt which can be executed by a robot. Missing parts are inferred with the help of common sense knowledge about actions. In contrast to all these approaches, we propose a framework which incorporates different information sources to get a better understanding of the system. Furthermore, our framework generates different models out of an internal formal description depending on the needs of the intended diagnosis and testing tasks.

Beside NLP methods, machine learning can also be used to generate a model of the system. The work in [4] presented a method to statistically learn the model of the system under nominal conditions. The model describes the static interaction of the system components. In contrast, the method proposed in [5] learns the behavior of a system. The method infers from observed events similar/different states and merges similar ones. Furthermore, the variables in the system for each state are estimated. Both methods are only applicable if the system is already built. Instead, we create a model during the design phase, and so the model can be used right at the first stages of the life-cycle.

Missing or contradicting information must be detected and handled when generating models. The method in [15] tries to avoid faults in the requirements document. This is done through the transformation of the requirements into so called boilerplates. Through this semi-structured text, ambiguities are removed and a consistent naming is enforced. A different approach was proposed in [14] to diagnose a knowledge base for consistency. If the knowledge base is inconsistent, the user is asked as an oracle to pinpoint the problem. Afterwards, the user needs to fix this issue. In our framework, we will use ideas from both methods to derive a consistent knowledge base of the system.

3 Framework overview

We propose the framework depicted in Figure 1 to transform informal documents and knowledge into models suitable for MBD. The informal inputs (white squares with solid lines) are processed into intermediate representations (light gray squares with dashed lines) using techniques from NLP and KRR, as well as ontologies (e.g. Cyc). We condense them into a knowledge base together with all our knowledge about the system and its domain. Finally, a variety of algorithms can produce formal models suitable for MBD (gray squares with dot-dash lines).

4 Sources of information

The proposed framework takes artifacts from the design phase as inputs. We propose the use of the following four inputs, though additional sources can be incorporated if available:

1. Requirements document: The technical requirements document describes the expected system behavior. Therefore, it is a mandatory input. The models' quality and so the resulting MBD will heavily depend on the quality of the requirements. Thus, iterative improvement of the requirements and models is used, as proposed in [15]. For our running example, we have taken four requirements that describe the box loading process of a robot:
   (a) When the robot is docked, it lowers the barrier.
   (b) When the robot is ready to load, the load handling device starts rotating backward.
   (c) The load handling device stops rotating backwards when the laser beam is triggered.
   (d) After stopping the load handling device the barrier is raised.
2. Domain knowledge: This is the most fuzzy input, as it is available not as an artifact but as the knowledge and experience of the engineers involved. We distinguish three kinds of knowledge. Common sense knowledge can be provided by existing ontologies as Cyc [16]. Generic knowledge about the autonomous systems domain can be provided by dedicated ontologies as KnowRob [17]. Particular knowledge about the targeted system itself can be partially inferred from the system architecture, though other parts must be provided by the project engineers. The use of ontologies range from providing meaning to natural language concepts to inferring missing pieces of information.
3. Architecture: The architecture of the system defines its composing elements plus the relations between them. It is typically described as a set of diagrams generated during the design phase of the system. For our running example, we use the architecture excerpt depicted in Figure 2. It states that a robot consists of a LHD and other unspecified elements. Furthermore, the LHD consists of a laser beam, rollers and a barrier.
4. Failure Modes and Effects Analysis: FMEA looks at all potential failure modes, their effects and causes and determines a risk priority factor. FMEA can be used to determine which potential errors are critical, how they can be pinpointed, and how the effects thereof can be avoided [18]. We incorporate the failure modes into the resulting behavior models to diagnose these known

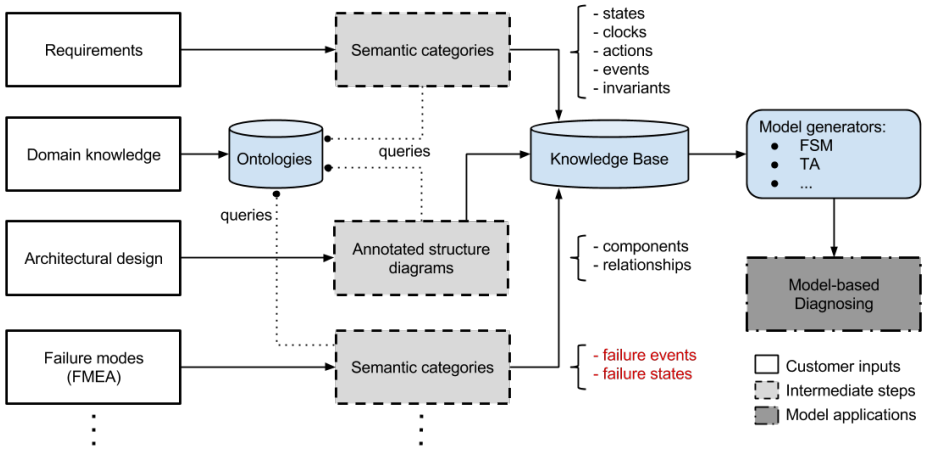

Figure 1: Abstract work-flow for the proposed framework. Starting from left with inputs in natural language, we generate models that can be applied for diagnosis (right).

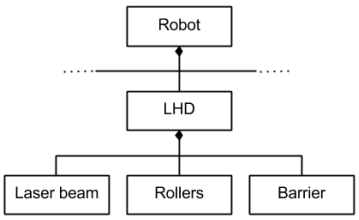

Figure 2: Robot architecture excerpt. The figure shows relations of the type *part of* for components of the Robot.

failures. For our running example, we include the two failure modes that can occur during the load operation, depicted in Table 1.

The biggest challenge for handling all these inputs is to understand semi-structured information. So, we will depict a NLP/KRR tool-chain using state-of-the-art techniques in the following section.

5 NLP/KRR tool chain

The process generates three intermediate artifacts: semi-formal text (boilerplates), syntax trees and semantic categories. As a showcase, we will concentrate on the requirements of our running example, though these techniques can be extended to other textual inputs, as we will see at the end of this section.

5.1 Boilerplates

This is a semi-formal representation where most of the spelling errors, poor grammar and ambiguities have been removed. Boilerplates also enforce the use of a consistent naming scheme. There exist tools such as [19] to perform this task semi-automatically. In our example, the four requirements become the four equivalent boilerplates:

(a) when the robot is docked, it lower the barrier.
(b) when the robot is ready to load, the lhd start rotating backward.
(c) when the lb is triggered, the lhd stop backward rotation.
(d) after stopping the lhd, the barrier is raised.

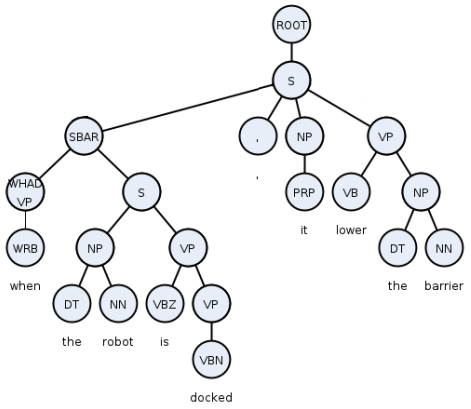

Figure 3: Sample syntax tree of the first sentence (a) of the running example.

Note for example that the 3rd person “s” has been removed from the verbs. Furthermore complex terms such as “load handling device” have been replaced by lhd. Finally, the propositions order is rearranged in a consistent structure.

5.2 Syntax trees

A syntax tree comprises the information of the type of each word in the sentence, e.g. ”lower“ is a verb. Furthermore, the tree specifies how the sentence is constructed with these words. For example, the syntax tree of the first requirement in our running example is depicted in Figure 3. In this syntax tree we can identify that “robot” is a noun and “the robot” is a so called noun phrase. An example of a tool to extract syntax trees is the probabilistic context free grammar parser, described in [20].

5.3 Semantic categories

The semantic categories conceptually describe our system, e.g. a transition describing the motion of an actuator. These semantic categories are hierarchical in nature, as more complex and abstract concepts are composed of simpler ones, e.g. a transition is composed by an action, pre and post conditions, etc. We obtain the semantic categories by parsing the syntax trees and applying transformation rules in a

	Component	Failure	Observations
Failure 1	Barrier	Barrier stuck up	Barrier stuck up regardless commands
Failure 2	Load Handling Device (LHD)	Rotation fail	Laser beam not triggered

Table 1: FMEA from the running example.

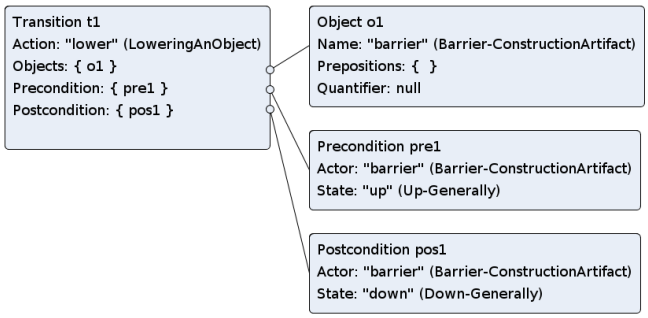

Figure 4: Concepts created from the syntax tree in Figure 3. The word in quotes is the word as it appears in the sentence. The word in parenthesis is the Cyc concept it belongs to.

bottom up fashion, following [8]. We start at the leafs of the syntax tree, containing single words. Each word has assigned a part-of-speech (POS) label describing its grammatical role in the sentence. Furthermore, each word has an additional label with its WordNet [21] synset, used to derive its semantics from the common sense knowledge base. From the leafs, higher level transformations can be applied to create more complex semantic categories. For example, on our running example we create a semantic category for each word in the sentence "lower the barrier". Then, we can derive that "lower" is an action acting on something. We can after that use the semantic category of the word together with its position in the syntax tree to apply further transformation rules. This process is repeated till the root node is reached. Then, a new semantic category is assigned to the sentence capturing its semantics. For the running example, the semantic category for "lower the barrier" is a transition. A transition must contain a precondition, a post condition, an action and optionally an object of the action. The semantic category specifies that the action "lower" is performed on the object "barrier". With the help of common sense (Cyc ontology [16]) we can reason that this action causes the "barrier" from state "up" to state "down". Thus, we can infer the pre and post conditions of "lower". Finally, the semantic category together with the reasoning results are packed into statements on our knowledge base, as it is depicted in Figure 4.

We can incorporate other documents into the knowledge base by using a similar NLP tool chain. However, how the information is treated depends heavily on the context inherent to each document type.

6 Model generation for behavior diagnosis

To illustrate how the framework can be used to diagnose the behavior of the robot, we create an automaton as output model. To use techniques such as [22], the automaton must describe both nominal and faulty behaviors of the system. To generate this automaton from the knowledge base, we use four different relations stated on it as transitions:

1. Relations representing a direct transition, as depicted in Figure 4. Such a transition can be directly mapped into a transition on the automaton, as can be seen in Figure 5 through the transitions from state 1 to 2.
2. Relations representing an action with a duration. Such a relation must be translated into several transitions: the start of the action, the termination event and a transition to a final state. Such transformed relation is depicted in Figure 5 through the transition from state 2 to 5.
3. Relations representing a failure of the system. The failure event is represented as a divergent path from a normal transition. Thus, the start state is the same as the one of the normal transition. Afterwards, we need a state representing the failure. Finally, we need an observation transition that leads to a final state representing a general failure of the system. The observable transition is cased due to the fact that use a fault model which is derived from the FMEA. Thus every fault has an observable discrepancy to the real system. Additionally it is important to notice that the state representing the general failure is state where the system can exhibit arbitrary behavior. Thus we can model the lack of knowledge which impact the fault has on the system. The transformed failure is is depicted in Figure 5 through the transitions from state 2 to 9.
4. Relations representing a failure of a system component. The failure event is represented as a divergent path from a normal transition. To determine all the possible affected transitions, we must perform an inference of the effects each transition has. This inference is based on common sense and domain knowledge. In our running example, we can infer that lowering the barrier causes the barrier to be finally down. A failure such as *barrier_stuck_up* can prevent this transition, and so they can share a common source state. Then, as before we need an observation transition that leads to a final state representing a general failure of the system. Such a sequence is depicted in Figure 5 though the transitions from state 1 to 9 through the states 7 and 8.

7 Conclusion and future work

In this paper we propose a framework to automatically generate formal models out of documents represented in semi-structured form and natural language (requirements, domain knowledge, architecture, failure modes, etc.). The parsed information is gathered together with domain knowledge in a knowledge base. Accessing this common repository, a variety of algorithms can generate different kinds of models for different purposes. Our main target is to derive models suitable for state-of-the-art MBD techniques applied to autonomous systems. We plan to implement this framework to assist us on creating the models required for MBD. Doing so, we expect to improve the dependability in the industrial application of a fleet of transport robots in a warehouse.

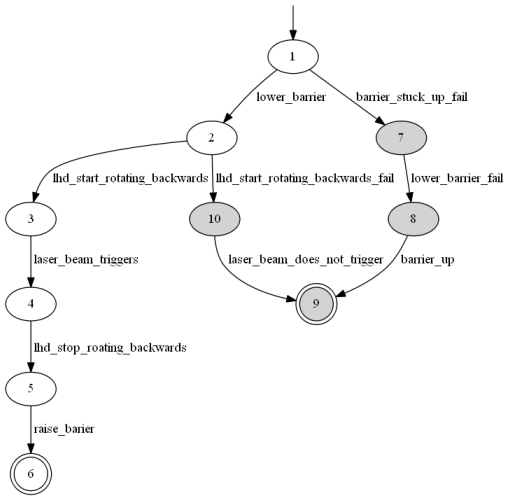

Figure 5: Automaton generated from the running example. Shaded states are reached through some fault. Double circled states represent final states. State number 9 is the general failure state for readability the self loops with all possible labels are omitted.

Besides this immediate result, we expect that the proposed framework will ease the creation of formal models for other applications. Thus, we hope to contribute to the widespread use of MBD techniques, with the consequent improve of autonomous systems dependability.

Acknowledgments

The research presented in this paper has received funding from the Austrian Research Promotion Agency (FFG) under grant 843468 (Guaranteeing Service Robot Dependability During the Entire Life Cycle (GUARD)).

References

[1] Stuart Kent. Model driven engineering. In Michael Butler, Luigia Petre, and Kaisa Sere, editors, *Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer Berlin Heidelberg, 2002.

[2] Mark Utting and Bruno Legeard. *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2010.

[3] Peter Struss, Raymond Sterling, Jesús Febres, Umbreen Sabir, and Marcus M. Keane. Combining engineering and qualitative models to fault diagnosis in air handling units. In *European Conference on Artificial Intelligence (ECAI) - Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 1185–1190, 2014.

[4] Safdar Zaman and Gerald Steinbauer. Automated Generation of Diagnosis Models for ROS-based Robot Systems. In *International Workshop on Principles of Diagnosis (DX)*, Jerusalem, Israel, 2013.

[5] Dennis Klar, Michaela Huhn, and J Gruhser. Symptom propagation and transformation analysis: A pragmatic model for system-level diagnosis of large automation systems. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–9. IEEE, 2011.

[6] Cynthia Matuszek, John Cabral, Michael Witbrock, and John Deoliveira. An introduction to the syntax and content of Cyc. In *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49, 2006.

[7] Markus Waibel, Michael Beetz, Raffaello D'Andrea, Rob Janssen, Moritz Tenorth, Javier Civera, Jos Elfring, Dorian Gálvez-López, Kai Häussermann, J.M.M. Montiel, Alexander Perzylo, Björn Schießle, Oliver Zweigle, and René van de Molengraft. RoboEarth - A World Wide Web for Robots. *Robotics & Automation Magazine*, 18(2):69–82, 2011.

[8] Moritz Tenorth, Daniel Nyga, and Michael Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1486–1491. IEEE, 2010.

[9] Shalini Ghosh, Daniel Elenius, Wenchao Li, Patrick Lincoln, Natarajan Shankar, and Wilfried Steiner. Automatically extracting requirements specifications from natural language. *arXiv preprint arXiv:1403.3142*, 2014.

[10] Sven J Körner and Mathias Landhäußer. Semantic enriching of natural language texts with automatic thematic role annotation. In *Natural Language Processing and Information Systems*, pages 92–99. Springer, 2010.

[11] Gerald Steinbauer. A survey about faults of robots used in robocup. In Xiaoping Chen, Peter Stone, LuisEnrique Sucar, and Tijn van der Zant, editors, *RoboCup 2012: Robot Soccer World Cup XVI*, volume 7500 of *Lecture Notes in Computer Science*, pages 344–355. Springer Berlin Heidelberg, 2013.

[12] Gerald Steinbauer, Franz Wotawa, et al. Detecting and locating faults in the control software of autonomous mobile robots. In *IJCAI*, pages 1742–1743, 2005.

[13] Mathias Brandstötter, Michael Hofbaur, Gerald Steinbauer, and Franz Wotawa. Model-based fault diagnosis and reconfiguration of robot drives. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA, 2007.

[14] Kostyantyn Shchekotykhin, Gerhard Friedrich, Patrick Rodler, and Philipp Fleiss. A direct approach to sequential diagnosis of high cardinality faults in knowledge-bases. In *International Workshop on Principles of Diagnosis (DX)*, Graz, Austria, 2014.

[15] Bernhard K Aichernig, Klaus Hormaier, Florian Lorber, Dejan Nickovic, Rupert Schlick, Didier Simoneau, and Stefan Tiran. Integration of Requirements Engineering and Test-Case Generation via OSLC. In *Quality Software (QSIC), 2014 14th International Conference on*, pages 117–126. IEEE, 2014.

[16] Stephen L Reed, Douglas B Lenat, et al. Mapping ontologies into Cyc. In *AAAI 2002 Conference Workshop on Ontologies For The Semantic Web*, pages 1–6, 2002.

[17] Moritz Tenorth, Alexander Clifford Perzylo, Reinhard Lafrenz, and Michael Beetz. The roboearth language: Representing and exchanging knowledge about actions, objects, and environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1284–1289. IEEE, 2012.

[18] Hongkun Zhang, Wenjun Li, and Jun Qin. Model-based functional safety analysis method for automotive embedded system application. In *International Conference on Intelligent Control and Information Processing*, 2010.

[19] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Herbert Zojer, and Christian Panis. Dodt: Increasing requirements formalism using domain ontologies for improved embedded systems development. In *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2011 IEEE 14th International Symposium on*, pages 271–274. IEEE, 2011.

[20] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[21] George Miller and Christiane Fellbaum. Wordnet: An electronic lexical database, 1998.

[22] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *Automatic Control, IEEE Transactions on*, 40(9):1555–1575, 1995.