Adaptive Deployment of Safety Monitors for Autonomous Systems

Aswinkumar Vijayananth
Hochschule Bonn Rhein Sieg [aswinkumar.vijayananth@smail.inf.h-brs.de]
Jithin Sasikumar
Hochschule Bonn Rhein Sieg [jithin.sasikumar@smail.inf.h-brs.de]
Manoj Kolpe Lingappa
Hochschule Bonn Rhein Sieg [manoj.kolpe@smail.inf.h-brs.de]

***Abstract* — A systematic deployment of safety-critical software in the collaborative autonomous system is presented in this paper. A deployment architecture is proposed to enhance collaborative activities between humans and robots to perform a task by ensuring safety-critical standards. The proposed architecture enables the system to deploy safety-critical software on the collaborative robot autonomously based on the changing requirements specified by the task, platform, and the environment. The requirements are framed as a constraint satisfaction problem. The safety level of the autonomous system is maintained by solving the constraint satisfaction problem and deploying the correct monitoring strategy.**

I. Introduction

Autonomous systems have to guarantee safety while coping with execution in various environments particularly when robots interact with humans. These are the critical systems that need to be working without any failure. Failure to execute on the right time and place may lead to damage to humans or even death, or financial loss. To carry out robot work it needs to deploy a wide variety of software components such as planning, perception and safety critical components. Safety can be defined in absolute terms as "the absence of catastrophic consequences on the user(s) and the environment" [1]. Safety critical software involves diagnosis and fault detection and isolation, execution monitoring and isolation. A system is said to be autonomous when it has the ability to execute action autonomously also satisfying varying requirements demanded by the software. The varying requirements are difficult to track or predict due to the change in the environment feature, tasks and goals in real time. The changing requirements affect the functional components of the system and also safety critical components that help to handle the risk thereby protecting the system. In the field of robotics deployment is considered as an ongoing activity in real time that expects redeployment of functional and safety critical software depending on the requirement posed by the system. Autonomous behavior of the system expects the deployment of software components by themselves without any help from the external agent. By doing so human intervention is minimized and that justify the name autonomous. Robotics deployment is already available in terms of architectural and platforms models [2,3] and configuration files [4,5]. It is found in [6] that the implementation of the deployment is done offline rather than run time changes. To solve the real time deployment of software components reference architecture [6] is taken that can deploy the software components in different platforms. The reference architecture is realized in this paper for a collaborative robot application and validates the functioning of the system. Real time autonomous deployment of the safety critical software with varying requirements is realized in this paper.

II. Motivation

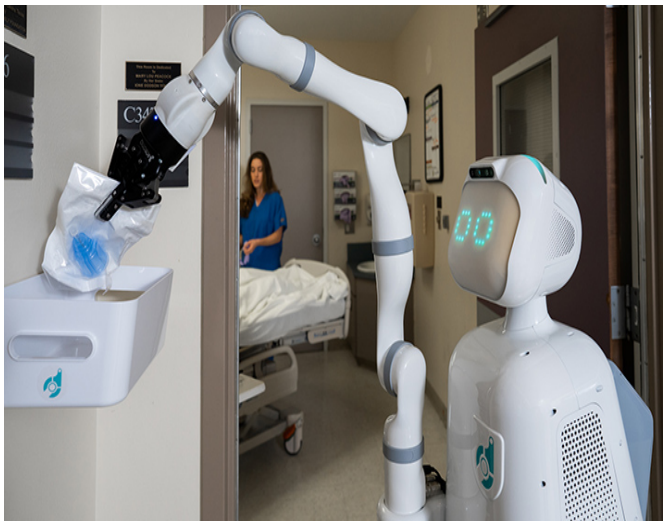

Figure 1: Collaborative robot in hospital environment

A collaborative robot helping humans to do their task is taken as an example in our case as shown in fig 1.0 to explain the deployment of safety critical software. Collaborative robots are a relatively new invention in the robotics community and nowadays they are widely used in industrial [7], home and office environments. Traditionally collaborative robots existed only in the factory environment and are separated from direct contact with humans for safety reasons. In modern days collaborative robots are expected to work along with humans to carry out a wide variety of tasks from assisting with small tasks such as picking a fallen target material to helping to move huge objects. The close interaction with humans demands high safety standards over the life cycle of the robotics applications. ISO TS 15066 [8] and ISO 12011 [9] standards help the safety critical engineers to perform the risk assessment of the collaborative robots while performing different tasks in different environments. Autonomous robot helper (fig 1) a collaborative robot is taken as the example to show the application area of deployment of safety critical software. The autonomous robot is capable of performing a wide variety of tasks such as preparing, transporting and handling different common things in a hospital environment. Risk assessment needs to be carried out on such robots to ensure safety. Risk can be in any form for example robots could damage a glass by dropping a glass while it tries to keep it on the table. So it is important to assess the risk, namely the safety monitoring system as explained in the below section

A. *Safety Monitor for detecting glass slippage*

Collaborative robots are equipped with different kinds of sensors to detect slippage to safely execute the manipulation task such as picking an object, placing and human handover. That is to detect whether the object is slipping from the robot's grasp. To detect the in hand slippage on the autonomous robot an architecture is proposed by Sanchez that has three different types of slip detectors based on tactile and force measurements and combination of these two sensors. The force slip detector detects a slip whenever there is a force exerted in the right direction. The tactile slip detector estimates the tangential force on the sensor caused by a sliding pressure. The combined slip detector combines the signal received by both tactile and force slip detectors in a rule such that the signal from force and tactile sensor is above a certain threshold value. These three slip detectors represent an active safety feature called as safety monitors that enables by checking the safety relevant information. Action that leads to danger or harm for example dropping of glass while a robot tries to place it on a table can be avoided by deploying safety critical software on the target platforms. Three slip detectors come into picture in different parts of the time and their performance is affected at the same time for example when the robot is in moving condition. Thus the evaluation of the autonomous robot suggests that all the action needs to be considered for improved safety performance. Hence it is necessary to adapt the safety monitor to the current robot's actions at run time.

III. CONCEPT

A. *System Architecture*

The proposed concept is realized in form of a system with the following components:

- Context monitor
- Repository
- Safety monitors

A1.1 Context Monitor

Context monitor is responsible for observing the autonomous system's environment and current status. In the given example, the robot's states of motion and gripper status are constantly checked by the context monitor. The component monitors the environment with the help of sensors. On the other hand, it is subscribed to different components of the robot so that it stays well-informed about their current statuses. The context monitor instantly updates the observed changes to the repository so that the information is shared with other necessary components.

A1.2 Repository

Repository is considered to be the central component in the proposed system. It acts as the knowledge base that contains data that were added during the design-time and also the information updated during the run-time. Data added during the design-time mostly include static(does not change during run-time) information like total number of existing hardware platforms in the system with their features, constraints to be satisfied for selecting a platform etc., On the other hand, information updated during the run-time are dynamic in nature. Memory availability in each platform is a good example for a property that changes during run-time. It is desirable to consider current working status of a sensor to be dynamic in nature. This is helpful in avoiding deployments in the platform with faulty sensors.

The Repository offers an interface to other components to insert new data to the repository and to query some specific data from it. It broadcasts the updates to other components with the help of the "notify" functionality. Broadcasting is triggered by events and changes that happen in the component.

A1.3 Safety Monitors

Safety Monitors are responsible for constantly checking whether the task designated to the autonomous system is executed in a safe manner. In the given example where the robot is responsible for picking, transporting and placing various objects, it is important to keep an eye on how safely these tasks are executed without harming humans in a collaborative environment. One of the ways to ensure safety in this case is to constantly check for slippage in the gripper with help of suitable sensors such as force and tactile. Slip detectors correspond to software components of the sensors in our system. Whenever a slippage is detected, appropriate force has to be exerted on the object by the gripper in order to prevent its fall.

Requirement type	**Example**
Quantity	Minimum number of tactile sensors to select a platform
Capacity	Minimum memory availability required for a deployment
Minimum	Minimum system latency
Maximum	Maximum system latency
Attribute	Presence of force sensor in a platform
Selection	Suitable force sensor type

Table 1: Examples of different requirement types

It turns out that the performance of slip detectors (force and tactile) in detecting slippage differs according to the context (robot's environment). The tactile slip detector is suitable to detect slips from a firmly held object whereas the force slip detector is more accurate than the other in detecting a complete slip. There are cases in which the fusion between force and tactile slip detectors are effective.

B. *Deployment Infrastructure*

There are often more than hardware platforms available in a distributed autonomous system. In order to achieve maximum utilization of available platforms, there arises the need to adaptively deploy software components on them based on the current context. At the same time, there is a need to firstly resolve suitable platforms/platform for the chosen software component based on some or all of the following requirement types as specified by OMG deployment specification [10]. The following table gives examples for each of them in the context of the object picking robot that is considered for our discussion.

C. Working

As soon as the context monitor reports a change in the robot's context which expressed in terms of its current state of motion and gripper status, the repository broadcasts the information so that it enables both selection of the right safety monitor and platform for its deployment.

C1. Selection

C1.1 Safety Monitors

Selection of the right safety monitor according to the context in [11] is carried out with help of the Perception graph deployment module. However, its working is specific to RPSL. In our implementation, this achieved differently with help of the safety monitor selector as described under IV. D. safety monitors are selected based on the current context.

C1.2 Platforms

C1.2.1 Criteria for selection

Once the slip detector according to the context is chosen, it has to deploy on an appropriate platform. In our implementation, platform selection is done with help of the platform selector module which In the object picking robot example, the suitability of platform is described is expressed in terms of the following requirements. The first three requirements are identical to our reference [11], whereas the other three are our contribution:-

R1: The force slip detector should be deployed on a platform to which the force sensor is connected.

R2: The tactile slip detector should be deployed on a platform to which all tactile sensors are connected.

R3: The combined slip detector should be deployed on a platform with at least 250MB working memory

R4: The combined slip detector has to be deployed on a platform whose latency is at least 200 ms

R5: The combined slip detector has to be deployed on a platform whose latency is 5 ms at the maximum

R6: The force slip detector has to deploy on a platform that contains a simple pressure sensor.

C1.2.2 Possible scenarios

Following are possible scenarios during the selection of platforms:

1. Selection of exactly one suitable platform: This is the ideal case where the selector returns exactly one platform meeting the requirement and is suitable for deployment.
2. Selection of multiple suitable platforms: This is the scenario in which at a given time more than a platform satisfies the given requirements. In this case, the selector returns all of them. However, there arises the need to rank and choose the most suitable platform.
3. Unavailability of a suitable platform: In this case, the platform selector returns none or any other equivalent to convey the absence of a suitable of platform meeting the specified requirements.

IV. IMPLEMENTATION

A. Constraint Satisfaction Problem

Constraint Satisfaction Problems (CSP) are mathematical questions defined as a set of objects whose state must satisfy a number of constraints or limitations [12]. Constraint refers to rule, limitation or restriction. CSP describes an efficient way to solve a wide variety of problems. A **factored representation** is used for each state which is represented as a set of variables, each of which has a value. The problem is solved when each variable has a value that satisfies all the constraints on the variable. [12]

A constraint satisfaction problem consists of the set of variables (X), domain for each variable (D), and set of constraints (C). It can be formulated as follows:

X is a set of variables, {X_1. . . X_n}.
D is a set of domains, {D_1, . . . D_n}.
C is a set of constraints that specify allowable combinations of values.

Each domain consists of a set of allowable values {v_1. . . v_k} for variable X_i. For example, A1 and A2 are the two variables with domain {X, Y}, then the constraint stating the two variables must have different values can be defined as $C = \{A1 \neq A2\}$ [12]

The values will be assigned to some or all of the variables by means of assignment. An assignment is said to be complete when each and every variable is assigned with a value. A solution obtained by solving CSP will be an assignment that is complete satisfying all the constraints.

A1.1 Advantages

1. CSP can keep track of the variables violating a constraint.
2. It helps in clean specification of many problems, generic goal, successor function and heuristic. When a problem is represented as a CSP, it can be solved with general package.
3. All the branches that violate constraints will be pruned off automatically.
4. CSP is faster than state space searchers because the large swatches of the search space can be eliminated quickly. The problems that are hard to control for regular state space search can be solved quickly when formulated as a CSP

B. MiniZinc

MiniZinc is a free and open source constraint modeling language. It can be used to model constraint satisfaction and optimization problems as a higher level. It is a language designed where constrained optimization and decision problems can be specified over integers and real numbers. It is designed to act as an interface between different backend solvers easily. An input Minizinc (.mzn) model and data file is transformed into a FlatZinc model. FlatZinc model consist of variable declaration along with constraint definitions and objective function definition for optimization problem. The translation from MiniZinc to FlatZinc is performed to individual backend solvers, so that the constraints will be controlled. Particularly, MiniZinc allows the specification of global constraints by means of decomposition. [13]

B1.1 MiniZinc as an IDE

MiniZinc comes with an Integrated Development Environment (IDE) that makes it easier for developing and executing constraint models. It is user friendly. IDE facilitates in writing and running MiniZinc models. It is provided with tabbed editor with highlighter for MiniZinc syntax followed by configuration dialogs for solver options and model parameters, and an integrated environment where the models can be compiled and running solvers. [14]

MiniZinc IDE's latest version is 2.4.3 which was released on 2020. It can be installed as bundled package that consists of Minizinc library, IDE, solvers and interfaces. Minizinc is used to solve the CSP and return the solutions satisfying given constraints.

B1.2 MiniZinc Python API

It is a python package that allows accessing all of Mini Zinc functionalities directly from Python. It provides an interface from Python to MiniZinc driver. It provides easy access to MiniZinc using native Python structures. This allows more scripts to run MiniZinc, but will also allow the integration of MiniZinc models within bigger python projects. It can be installed using pip command. [15]

C. Framing deployment planning as a CSP

Let us consider the robot containing the platforms and sensors along with their attributes and properties.

C1.1 Variables and domains

Variable	Domain
force_platform, tactile_platform, fused_platform	Set of all available platforms
force_sensor_presence	True, False
tactile_sensor_count	$\mathbb{Z}\geq 0$
memory_availability	$\mathbb{Z}\geq 0$
Minimum, maximum latency	$\mathbb{Z}\geq 0$
force_sensor_type	{"simple_pressure_sensor","capacitive and resistive flexible force sensor","piezoelectric","strain gauge based"}

Table 2: Table of variables and corresponding domains used in the CSP

C1.2 Constraints

The constraints must be formulated in a way that satisfies the system requirements R1 to R6(as listed in C1.2.1) to ensure deployment of the right safety monitor on the right platform.

C = {
force_sensor_presence(platform) = True,
force_sensor_type(platform) = simple pressure sensor,
tactile_sensor_count(platform) > min_tactile_sensor_count,
memory_availability(platform) > = min_fused_memory_availability,
latency(platform) > = min_latency,
latency(platform) <= max_latency }

C1.3 Solving CSP using MiniZinc

Deployment planning expressed as a Constraint Satisfaction Problem (CSP) is programmed and solved using the Mini Zinc modeling language. The Mini Zinc model is responsible for returning platforms matching requirements for all the slip detectors at a given point in time. The Mini Zinc model is contained in the platform selector class which returns the platform in which the active safety monitor/slip detector has to be deployed.

The Mini Zinc model takes the constraint thresholds as inputs as listed in the second column of Table[1] and it outputs platform assignments for every slip detector. Figure 2. contains the piece of code that expresses the constraints using the Mini Zinc modeling language:

```
26  n = ["nvidia_force","amd_tactile","intel_fused","nxp_force"];
27
28  %Platform properties
29  dd =             [| 1,   1,   50, 400,    % PF1
30                    | 2,   0,  100, 100,    % PF2
31                    | 3,   0,   10, 400,    % PF3
32                    | 4,   1,    0,  50|];  % PF4
33
34  %Platforms to be assigned for every sensor
35  array[FEATURES] of var int: force_platform;
36  array[FEATURES] of var int: tactile_platform;
37  array[FEATURES] of var int: fused_platform;
38
39
40  %Constraint of platform names to be in the provided table
41  constraint force_platform[name] in platforms;
42  constraint tactile_platform[name] in platforms;
43  constraint fused_platform[name] in platforms;
44
45
46  %Constraint that the values to be assigned to platform names are available in the table
47  constraint table(force_platform, dd);
48  constraint table(tactile_platform, dd);
49  constraint table(fused_platform, dd);
50
51  %Technical constraints
52  constraint force_platform[force_sensor] >= min_force_sensor_count;
53  constraint tactile_platform[tactile_sensor] >= min_tactile_sensor_count;
54  constraint fused_platform[memory_availability]>=min_memory_fused;
55
56  solve satisfy;
```

Figure 2: Mini Zinc code snippet containing constraints

Properties of various available platforms are represented using Mini Zinc's table data structure (lines: 29-32). Columns of the table corresponding different properties/features of the platform. Each row of the platform defines an individual platform along with their features. Columns of the table represent platform id, presence of the force sensor, tactile sensor count, memory availability, latency and force_sensor_type of a platform. Values of features of a given platform are accessed using the property name as index. force_platform, tactile_platform and fused platform correspond to variables that are assigned with platforms matching their requirements. Lines (52 – 54) express the constraints described in **R1 – R6**. The satisfy command directs the solver to satisfy the given constraints.

The Mini Zinc model therefore returns platforms for all the slip detectors whenever it is triggered. However, the platform selector which has the information about the active safety monitor takes the responsibility of picking only the platform which is needed at that moment. Following is an use case that describes the working of the system. Let us suppose , the context monitor reads that the robot is

stationary and gripper is open. This change triggers the Safety monitor selector to pick the tactile slip detector. Platform selector fetches this information from the Repository. The context change invokes the CSP solver which returns all valid platforms. Finally the platform selector updates the repository with the platform suitable for the currently active safety monitor.

D. Extensibility of the proposed solution

The proposed solution (CSP) is executed in real time. The running (or) execution time to select a suitable platform by solving the CSP problem is approximately 0.70 seconds. It is repeated for a set of robot's context and all the execution takes the similar time approximately for execution.

CSPs are NP-Complete. The solutions can be counted by ignoring all the constraints for a given problem instance. It is dependent on the number of decision variables and the sizes of their domains. Every solution will be checked by this algorithm. It can be stated that the computational complexity grows very fast with size of problem instance. The number of decision variables along with their sizes can be the factors that have an impact on the complexity.

E. Class Diagram

The class context monitor provides the contextual information needed for selection and deployment. It consists of attributes namely robot in motion and gripper status where the values are fetched from the environment by the robot (i.e.) values are updated dynamically at every time interval t. The state of the robot is sent to the repository and updated.

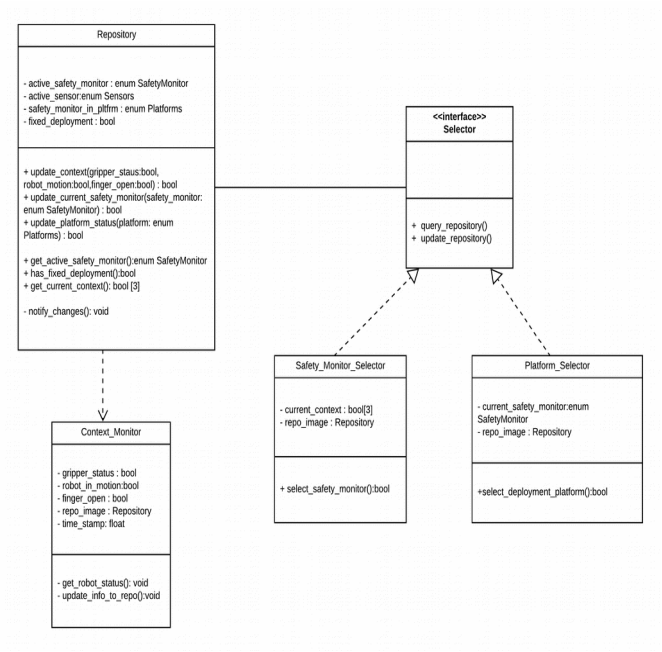

Figure 3: UML Class Diagram

Repository contains information for deployment. It is the central component in the architecture. Every component updates the information to the repository. All the components are connected to the repository. The repository class provides attributes for storing the values of robot's current context, selected safety monitor and platform for deployment.

Selector is an interface that acts as a communicator between repository class and other two classes namely safety monitor selector, platform selector. The selector interface provides two methods for querying and updating information to the repository. Safety monitor selector class contains definition for selecting the safety monitor based on the current context. The selected safety monitor information is updated in the repository using interface. The safety monitor class selects a suitable safety monitor using select safety monitor method by checking the current context of the robot and updates the repository accordingly. The platform selector class selects the suitable platform based on the selected safety monitor. Even the selected platform information is updated to the repository. The platform that satisfies the given requirements will be selected. This is facilitated by formulating the problem as CSP and solved using MiniZinc. The requirements are formulated into suitable constraints and platforms that satisfy the given constraints will be selected. Minizinc contains definitions for the formulated problem along with the constraints.

The design pattern followed is interface pattern. The main idea is the functions are separated from implementations. It can be used when it is necessary to know how the messages are exchanged within classes (i.e.) when a class should be reused each and every time, the communication interface will be declared as an Interface type. [16]

V. CONCLUSION

Safety monitoring systems for runtime is a critical component of any autonomous system. This article proposed and realized an approach to deploy the safety critical software on a virtual autonomous robot platform in real time with varying requirements posed by the system. This on-line deployment of safety monitoring strategy requires safety properties that need to be checked. This paper provides a solid approach to define the requirement types as a constraint and solve the constraint with the help of constraint solver software.

VI. FUTURE WORK

The major contribution of this work is re-framing deployment planning as a Constraint Satisfaction Problem. This work however only discusses the basic idea with a simple example. The proposed idea can be extended further by specifying preferences for the properties of platforms. For instance, in the object picking robot, deploying fused slip detector on a platform with the maximum processing speed can be a preference. Preferences are helpful in selecting the most suitable platform in a situation where more than a platform satisfying the given requirements exists. In CSP's language, such preferences are termed as soft constraints. The Mini Zinc modeling language inherently does not contain any provision to specify the soft constraints. However, Mini Brass [17] which is an extension of Mini Zinc can be used to achieve the same.

References

[1] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic ˇ concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1:11–33, 2004

[2] Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., Ziane, M.: RobotML, a domainspecific language to design, simulate and deploy robotic applications. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) SIMPAR 2012. LNCS (LNAI), vol. 7628, pp. 149–160. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34327-8 16

[3] Nordmann, A., Hochgeschwender, N., Wigand, D., Wrede, S.: A survey on domain specific modeling and languages in robotics. J. Softw. Eng. Rob. 7(1), 75–99 (2016).

[4] Bruyninckx, H., Soetens, P., Koninckx, B.: The real-time motion control core of the OROCOS project. In: Proceedings of the IEEE International Conference on Robotics and Automation (2003)

[5] Quigley, M., et al.: ROS: an open-source robot operating system (2009)

[6] Hochgeschwender, N., Biggs, G., Voos, H.: A reference architecture for deploying component-based robot software and comparison with existing tools. In: 2018 Second IEEE International Conference on Robotic Computing (IRC), pp. 121–128, Jan 2018.

[7] Villani, V., Pini, F., Leali, F., Secchi, C.: Survey on human-robot collaboration in industrial settings: safety, intuitive interfaces and applications. Mechatronics 55, 248–266 (2018)

[8] Robots and robotic devices - Collaborative robots. Technical specification, International Organization for Standardization, Geneva, CH (2016)

[9] Safety of machinery - General principles for design - Risk assessment and risk reduction. Standard, International Organization for Standardization, Geneva, CH (2010)

[10] Object Management Group: Deployment and configuration of component-based distributed applications specification (2004). http://www.omg.org/spec/DEPL/4.0/. Accessed 05 May 2019

[11] Hochgeschwender, Nico. “Adaptive Deployment of Safety Monitors for Autonomous Systems.” International Conference on Computer Safety, Reliability, and Security, 2019, pp. 346–357.

[12] Peter Norvig and Stuart J. Russell, Artificial Intelligence: A Modern Approach, Prentice Hall, 1994

[13] Kim Marriott and Peter J. Stuckey, “MiniZinc Tutorial”, 2007. [Online]. Available: https://www.minizinc.org/tutorial/minizinc-tute.pdf

[14] Mozilla Public License version 2.0, “IDE”, 2014. [Online]. Available: https://www.minizinc.org/ide/

[15] Mozilla Public License 2.0, Jip J. Dekker, "Minizinc Python, 2019. [Online]. Available: https://pypi.org/project/minizinc/

[16] Schatten, Alexander & Demolsky, Markus & Winkler, Dietmar & Biffl, Stefan & Gostischa-Franta, Erik & Östreicher, Thomas. (2010). Best Practice Software-Engineering. 10.1007/978-3-8274-2487-7.

[17] Schiendorfer, Alexander, et al. "MiniBrass: soft constraints for MiniZinc." Constraints 23.4 (2018): 403-450.