Dynamic Allocation of Service Robot Resources to an Order Picking Task Considering Functional and Non-Functional Properties

Timo Blender
Christian Schlegel
timo.blender@thu.de
christian.schlegel@thu.de
Technische Hochschule Ulm
Ulm, Germany

ABSTRACT

Industry 4.0 processes have often varying requirements. A service robot and a team of service robots respectively represent a flexible resource. That means, it possesses variability that can possibly be configured in such a way that it is able to fulfill the requirements of industry 4.0 processes. Determining whether that is the case and how that has to happen is an important part of variability management. Based on a model-driven general method for variability management in a robotics software ecosystem, we present here a concrete use case (model) in which we allocate for an order picking task with specific time requirements either a single fitting service robot or a collaboration of two fitting service robots. Relevant properties of the service robots considered are both functional (are the capabilities to execute the tasks available?) as well as non-functional (the desired velocity parameterization while executing the individual navigation sub tasks limited by the respective maximum speed of a service robot).

KEYWORDS

robotics software ecosystem, model-driven software development, variability management, non-functional properties, service robot collaboration

ACM Reference Format:

1 INTRODUCTION

Service robots are software-intensive, multi-purpose and highly flexible machines, that are expected to successfully and robustly perform a variety of tasks even in open-ended environments. As their flexibility and their capabilities foremost depend on software, mastering the software engineering challenge for such universal machines in an economically feasible way is pivotal. Thereto, the RobMoSys project has introduced a business ecosystem for robotics [rob] based on separation of roles and on composition [SLLS21].

A robotic system is built by composing (software) components that come with a model of relevant aspects from which finally the entirety of provided system-level capabilities and properties is determined.

Industry 4.0 serves as a useful and targeted approach to deal with challenges like volatile markets and demands, required customization, as well as decreasing product life cycles [KMAV17]. Thus, various tasks with individual requirements arise in the everyday life of a company which are to be fulfilled by the available resources (including service robots).

The research question now is: how can we model, manage and evaluate functional and non-functional properties of building blocks and systems as well as of tasks to be performed while following separation of roles and composition as these form the foundation of a business ecosystem for robotics software? One particular aspect of the general approach for variability management in a robotics software ecosystem (shortly introduced in section 2) is the possibility to get out of the composition process for a service robot a model of its capabilities and properties, of its exploitable variation points and of what the impact of configurations of its variation points is on its capabilities. On the other hand, the very same modeling elements for functional and non-functional properties can be used in descriptions of tasks to be performed. Beyond only matching which robot or team of robots is able to execute a task given available capabilities, we can now determine suitable assignments and individual configurations of variation points such that also non-functional properties fulfill specified non-functional requirements e.g. minimize resource usage. The focus of this paper is on a concrete use case in the domain of intralogistics to clarify the principle and the resulting advantages. Sections 3 and 4 describe the problem of our use case while section 5 implements the use case using our model-driven tools that make the general approach accessible within an ecosystem. Section 6 summarizes related work and section 7 presents a short conclusion.

2 VARIABILITY MANAGEMENT IN A ROBOTICS SOFTWARE ECOSYSTEM

A robotics software ecosystem is organized in three tiers (ecosystem drivers, domain experts and ecosystem users). A tier must conform to the structures defined by the next upper tier. Therefore tier 1
defines the most general structures and principles of a robotics software ecosystem. Tier 1 is in turn organized in different levels (see figure 1). The different composition structures conform to the block-port-connector principle and the block-port-connector principle in turn conforms to hypergraphs as the scientific grounding (see [SLLS21]). Based on these structures we developed a general method for variability management that allows to model building blocks and the (composed) possible properties taking into account their configuration variants and states of the environment in form of dependency variability graph models. Knowing possible properties of a building block is important to match it with the requirements, to compare it with other building blocks and to determine resulting (possible) properties of new compositions of building blocks in which this building block is involved. Next to the modeling aspect, the method also comes with a generic mechanism to resolve the problem space described by a dependency variability graph model, i.e. it determines the associated variants in dependence of the specified property requirements. Note that building block is a generic term: A whole service robotic application can be considered as a building block that is composed by several tasks, a task is composed by different skills and a skill is composed by different software components interacting with each other via services and executing activities and functions. One approach to model robotic behavior in a robotics software ecosystem is the use of hierarchical task-nets. These arrange actions that are to be executed and that end up in skills. These are realized by coordinated interplays of software components which are situationally configured, started and stopped, respectively. We use SmartTCL and its sequencer for task-net modeling and execution [SS11]. Applications, tasks, skills, components, activities and functions are therefore all building blocks at different abstraction levels and have properties that must be explicaded, composed and associated with their variants in order to determine requirement-related configurations. Hence, the provided method is one more contribution in a robotics software ecosystem to further simplify the process of developing and configuring service robotic applications.

3 THE ORDER PICKING USE CASE

We use the example of Order Picking throughout this paper to illustrate our approach. There are two models to realize Order Picking: NotShared and Shared depending on whether the task is executed by a single robot or by a collaboration of two robots. General representations are shown in figures 2 and 3. A corresponding model relevant for variability management is shown in figure 4. It refers to the meta-model for modeling building blocks which is part of our general approach. A behavior developer arranges such task plots by reusing (already existing) other task plots. In this example, a behavior developer might simply compose the topmost building block OrderPicking that is of interest for the end user by reusing the existing task plots NotShared and Shared and specifying that these two building blocks are functional equivalent alternatives to realize OrderPicking.

The root task OrderPicking as well as all other tasks in this model have the property Time assigned. It is representing the estimated time duration until the associated task is finished. OrderPicking also has an input (NumberObjects) which is the number of objects to be picked. The context Capacity represents the number of objects.
Figure 4: The building block model of the order picking use case utilizing composition and separation of roles. The arrows with the solid lines define the decomposition structure (decomposition sequence (from left to right) and the respective decomposition type(s)) of a building block. The arrows with the dotted lines point from building block instances to corresponding building blocks of the same types that occur several times in the decomposition structure but that must be modeled only once.

that can be transported by the allocated service robot. Note that all orders with NumberObjects > Capacity are rejected.

3.1 NotShared
The hierarchical task-tree of NotShared describes the workflow of actions to fulfill the order picking task by a single service robot. It is a sequence of the tasks CollectBox, Commissioning and DeliverBox.

3.1.1 CollectBox. CollectBox is a sequence of the tasks GotoLocation, DockToStation, LoadFromStation and UndockFromStation.

The task GotoLocation is needed at several points in this example. It represents the general service robot task of moving autonomously from A to B. Therefore it has the input GoalPosition and the context StartPosition. Furthermore, we also modeled a parameter Velocity representing a discretized value range (real numbers) to choose from before executing the task. The StartPosition of this specific GotoLocation task instance is the current position of a service robot candidate and the GoalPosition is a location in the warehouse where a specific station is placed from where an empty transport box can be fetched (FetchStation). DockToStation has the boolean context HasCapability assigned, because we assume that not every service robot resource available in our warehouse has the capability to dock to specific stations used there. LoadFromStation describes the process of receiving a transport box from the station. Finally, UndockFromStation follows.

3.1.2 Commissioning. Commissioning is a sequence of the tasks GotoLocation, DetectObjects and PickAndPlace. StartPosition of this
GotoLocation task instance is the location of the station from where the transport box was fetched (FetchStation) and GoalPosition is a location in the warehouse where the ordered objects to be picked can be found (PickingPlace). DetectObjects orients the camera to the location of the objects to be picked and determines their poses. We assume that the number of objects that can be recognized there is at least as high as NumberOfObjects. PickAndPlace is a loop of a sequence of Pick and Place where the number of iterations is NumberOfObjects. Pick moves a manipulator from its current position to an object pose from the set of detected object poses and finally grasps the object. Place moves the manipulator from this pose where the object was grasped to the place where the transport box is located on the docked transportation robot. After the object was grasped to the pose where the transport box is located GoalPosition is the received docking pose. After the docking pose is received from the manipulation robot, PickAndPlace has the boolean context HasCapability assigned, because we assume that not every service robot resource available in our warehouse has the capability to manipulate objects.

### 3.1.3 DeliverBox
DeliverBox is a sequence of the tasks GotoLocation, DockToStation, UnloadToStation and UndockFromStation. StartPosition of this GotoLocation task instance is the location where the ordered objects were picked (PickingPlace) and GoalPosition is a location in the warehouse where a station is placed to which the transport box containing the picked objects can be delivered (DeliverStation).

### 3.2 Shared
The hierarchical task-tree of Shared describes the workflow of actions to fulfill the order picking task by a collaboration of service robots (a manipulation robot and a transportation robot). First, there is a parallel execution of the tasks CollectBox and ApproachAndDetect. CollectBox was already explained in the NotShared part. ApproachAndDetect is a sequence of GotoLocation and DetectObjects which were also already explained. StartPosition of this GotoLocation task instance is the current position of a service robot and GoalPosition is a location in the warehouse where the ordered objects to be picked can be found (PickingPlace). The service robot assigned to the ApproachAndDetect task serves as the manipulation robot in this collaboration and informs the transportation robot (the service robot assigned to CollectBox) about where to dock to the manipulation robot after ApproachAndDetect is finished. If the transportation robot is faster in the execution of its assigned tasks, the transportation robot will wait until the docking pose is received from the manipulation robot.

After the docking pose is received, the transportation robot executes a GotoLocation task instance where StartPosition is the location of the station from where the transport box was fetched and GoalPosition is the received docking pose. After the docking pose is reached by the transportation robot, the transportation robot executes DockToRobot to approach and dock with the manipulation robot at a bit closer by executing relative movements. After that, the manipulation robot executes PickAndPlace that was already described previously. Place now moves the manipulator from the pose where the current object was grasped to the pose where the transport box is located on the docked transportation robot. After the PickAndPlace task is finished, the transportation robot executes the DeliverBox task already described previously.

### 4 PROBLEM DESCRIPTION
Let us suppose that a OrderPicking task is commanded with a specific NumberOfObjects and \( R = \{ R_1, ..., R_n \} \) service robots are currently available in our warehouse. The problem of this use case is to allocate either a \( R_i \in \{ R_1, ..., R_n \} \) to NotShared or \( R_j, R_k \in \{ R_1, ..., R_n \} \) with \( j \neq k \) to Shared by considering the required and provided capabilities as well as a time requirement.

#### 4.1 Functional Constraints
\( R_i \in \{ R_1, ..., R_n \} \) is a suitable candidate for a task \( T \) if:

\[
C_r(T) \subseteq C_p(R_i) \quad (1)
\]

where \( C_r(T) \) is the set of required capabilities of a task \( T \) and \( C_p(R_i) \) is the set of provided capabilities of a service robot \( R_i \). In our example we assume that all service robots in our warehouse have some basic capabilities such as navigation and object detection to realize most of the tasks. But as stated previously, in our example the tasks DockToStation and PickAndPlace can not be handled by all service robots because not every service robot has a conveyor belt mounted that can be docked to a corresponding station or has a manipulator attached. Therefore, we need to check for all service robot candidates \( R_i \in \{ R_1, ..., R_n \} \):

\[
R_i^N = \begin{cases} 
\text{true if } & \text{DockToStation} \in C_p(R_i) \land \text{PickAndPlace} \in C_p(R_i) \\
\text{false else} 
\end{cases} \quad (2)
\]

**NotShared**

\[
R_i^{ST} = \begin{cases} 
\text{true if } & \text{DockToStation} \in C_p(R_i) \\
\text{false else} 
\end{cases} \quad (3)
\]

\[
R_i^{SM} = \begin{cases} 
\text{true if } & \text{PickAndPlace} \in C_p(R_i) \\
\text{false else} 
\end{cases} \quad (4)
\]

where \( R_i^{ST} / R_i^{SM} / R_i^N \) denotes if the \( i \)-th service robot is a suitable candidate for NotShared/Shared (transportation part)/Shared (manipulation part).

#### 4.2 Non-Functional Constraints
The problem including the just described functional constraints as well as an additional non-functional constraint in form of the overall time can be formulated in two ways: Optimal or adequate. The first variant determines the allocation with the minimum time and the second variant is a constraint satisfaction problem where we are satisfied with any first allocation able to fulfill a specific constraint (a specific threshold value for time that should not be exceeded). Equation 5 formulates the optimal case where \( t_N(R_i) \) is the estimated time for the task NotShared when executed by service robot \( i \) and \( t_S(R_j, R_k) \) is the estimated time for the task Shared when executed in collaboration by service robot \( j \) (transportation part) and service robot \( k \) (manipulation part).

\[
\arg \min_{i=1..n, j=1..n, k=1..n} \left( t_N(R_i), t_S(R_j, R_k) \right) \quad (5)
\]

where: \( R_i^N \land R_j^{ST} \land R_k^{SM} \land j \neq k \)
4.3 Composition of Time

The equations 6 - 17 show the composition of the overall time of the OrderPicking task for its two variants NotShared and Shared based on the time of the contained sub tasks. The time of the GotoLocation task instances is a function of StartPosition, GoalPosition and the Velocity variable with a finite domain. $t_{GotoLocation}(R_i)$ denotes the estimated time of the $i$-th GotoLocation task instance executed by service robot $R_i$. Note that some time values of certain tasks are assumed to be constant and therefore do not depend on the allocated service robot $R_i$.

$$t_N(R_i) = t_{CollectBox}(R_i) + t_{Commissioning}(R_i) + t_{DeliveryBox}(R_i)$$

$$t_S(R_j, R_k) = \max(t_{CollectBox}(R_j), t_{ApproachAndDetect}(R_k)) + t_{GotoLocation}(R_j) + t_{DockToFStation}(R_k) + t_{DeliveryBox}(R_j)$$

$$t_{CollectBox}(R_i) = t_{GotoLocation}(R_i) + t_{DockToFStation} + t_{LoadFromStation} + t_{UndockFromStation}$$

$$t_{Commissioning}(R_i) = t_{GotoLocation}(R_i) + t_{DetectObjects} + t_{PickAndPlace}(R_i)$$

$$t_{PickAndPlace}(R_i) = \text{NumberOfObjects} \cdot (t_{pick}(R_i) + t_{place}(R_i))$$

$$t_{DeliveryBox}(R_i) = t_{GotoLocation}(R_i) + t_{DockToFStation} + t_{UnloadToFStation} + t_{UndockFromStation}$$

$$t_{ApproachAndDetect}(R_k) = t_{GotoLocation}(R_k) + t_{DetectObjects}$$

$$t_{GotoLocation}(R_i) = f(sp_1(R_i), gp_1, v_1(R_i))$$

$$sp_1(R_i) = sp_4(R_k) = \text{Current location of } R_j/R_k$$

$$gp_1 = sp_2 = sp_5 = \text{Location of FetchStation}$$

$$gp_2 = sp_3 = gp_4 = gp_5 = \text{Location of PickingPlace}$$

$$g_p = \text{Location of DeliveryStation}$$

4.4 Complexity Analysis

This section provides additional insights about the described problem by analyzing the complexity depending on the number of available and suitable service robots.

Let $R^{St}/R^{Sm}$ be the set of service robots suitable for the transportation/manipulation part of task Shared because they can DockToFStation/PickAndPlace and $R^N$ the set of service robots suitable for NotShared because they can both DockToFStation and PickAndPlace. $R^{St} = R^S \setminus R^N$ and $R^{Sm} = R^S \setminus R^N$ respectively is then the set of service robots that can DockToFStation but not PickAndPlace and vice versa. The number of possible allocations is then given by equation 18 if $|R^N| = 1$ ($|A_{=1}|$) and by equation 19 if $|R^N| > 1$ ($|A_{>1}|$).

$$|A_{=1}| = \left( |R^{St}| \cdot |R^{Sm}| \right) + \left( |R^{St}| \cdot |R^N| \right) + \left( |R^{Sm}| \cdot |R^N| \right) + |R^N|$$

(18)

$$|A_{>1}| = |A_{=1}| + \frac{|R^N|!}{(|R^N| - 2)!}$$

(19)

Equation 20 then indicates the worst case overall number of possible velocity combinations for the GotoLocation task instances of OrderPicking that must be considered if the whole variability problem space must be explored. The term $v_{max}$ denotes the highest possible translational velocity considering all potential candidates $R_i$ and $d$ is a discretization constant. Hence, if all potential service robot candidates have the same maximum translational velocity, the equation gives the exact number of possible combinations.

$$|V| = \left( |R^N| \cdot \left( \frac{v_{max}}{d} \right)^3 \right) + \left( |A| - |R^N| \right) \cdot \left( \frac{v_{max}}{d} \right)^4$$

(20)

5 REALIZATION AND RESULTS

This section illustrates how the problem described previously can be realized by a dependency variability graph model that considers composition and separation of roles as part of our general approach.

Figure 5 presents the dependency variability graph model for the building block OrderPicking of figure 4. The figure already shows the complete composed model where the modeler was able to reuse the existing dependency variability graph models of Shared and NotShared to represent the new problem space easily without the need to design everything from scratch again. The concrete available service robot candidates and their properties are retrieved from the corresponding service robot models and are connected to the dependency variability graph model either at design-time or dynamically at run-time (3 service robots in our example shown in the lower part of the model). The generic solver that is finally applied to the described problem space makes sure that only valid assignments are produced, is able to go through all possible valid combinations if desired and can determine the bindings of modeled variants that fulfill the specified requirements.

The 3 service robot candidates available for evaluation in this experiment are shown in figure 6. Their properties are listed in table 1. The estimation function $f(sp_1(R_i), gp_1, v_1(R_i))$ for the time of the GotoLocation task instances in this example is optimistic as equation 21 shows. Depending on the obstacle configuration in the environment, this estimation function can be inaccurate. However, better estimation functions could be applied (e.g. asking a path planner, a function approximated by machine learning). As soon as they are available, they can be easily applied with the help of our tooling to improve the accuracy.

Based on table 1 and equation 18 (because $|R^N| = 1$), the number of possible allocations is 4. Table 2 lists all the possible allocations and table 3 shows the constant values we use for this example.
Figure 5: The dependency variability graph model of the OrderPicking building block model (figure 4) utilizing composition and separation of roles. The requirement specified in this model is minimizing time.

Table 1: Properties of our service robot candidates.

<table>
<thead>
<tr>
<th>Robot</th>
<th>DockToStation</th>
<th>PickAndPlace</th>
<th>Max Vel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robotino</td>
<td>true</td>
<td>false</td>
<td>1200.0</td>
</tr>
<tr>
<td>Larry</td>
<td>false</td>
<td>true</td>
<td>1000.0</td>
</tr>
<tr>
<td>Macy</td>
<td>true</td>
<td>true</td>
<td>1000.0</td>
</tr>
</tbody>
</table>

\[
f(s_{p1}(R_i), g_{p1}, v_i(R_i)) = \sqrt{(g_{p1}^x - s_{p1}^x(R_i))^2 + (g_{p1}^y - s_{p1}^y(R_i))^2} / v_i(R_i)
\]

(21)

Table 2: The possible task allocations in this example based on the available service robots (table 1).

<table>
<thead>
<tr>
<th>Allocation</th>
<th>R^S_T</th>
<th>R^S_M</th>
<th>R^N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td></td>
<td>Macy</td>
</tr>
<tr>
<td>2</td>
<td>Robotino</td>
<td>Larry</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Robotino</td>
<td>Macy</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Macy</td>
<td>Larry</td>
<td>-</td>
</tr>
</tbody>
</table>
5.1 Experiments

5.1.1 Minimizing Time. The decision about the final allocation depends on the maximum velocity of the service robot candidates and their current positions when an order is placed. We carry out 3 different experiments with varying start positions for the service robots (table 4, figures 7, 8 and 9). Since we want to minimize time, it is clear that the velocity assignment for the different GotoLocation task instances should be as low as possible in that case. It is possible to model and solve these aspects with our dependency variability graph method. However, for reasons of space we cannot present it here. In [BS20] we show a similar use case based on the same method but only for a single service robot executing an order picking task. Note that in contrast to minimizing time, velocities can not be constants anymore but need to be defined as variables with the corresponding maximum velocities of the different service robots as the upper limits. Therefore, the complexity of the problems minimizing time and staying below a time limit are hard to compare and is out of scope of this paper.

5.1.2 Staying Below a Time Limit. Minimizing time may not always be desired. Let us assume that there is a time limit in which the order must be completed. Then it is fully satisfying if it is completed within that time limit. From this point of view, the variant with the minimum time is as useful as any other variant with a time below the time limit. However, if we consider a further property Energy in our allocation problem, one may conclude that the optimal variant in presence of a time limit is the variant that fulfills the time limit with the minimum energy consumed. If we assume, that the energy consumption increases with the driven velocity and distance, the assignment of velocities to the different GotoLocation task instances should be as low as possible in that case. It is possible to model and solve these aspects with our dependency variability graph method. However, for reasons of space we cannot present it here. In [BS20] we show a similar use case based on the same method but only for a single service robot executing an order picking task. Note that in contrast to minimizing time, velocities can not be constants anymore but need to be defined as variables with the corresponding maximum velocities of the different service robots as the upper limits. Therefore, the complexity of the problems minimizing time and staying below a time limit are hard to compare and is out of scope of this paper.

Table 3: Constant values in the example

<table>
<thead>
<tr>
<th>Names</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumberObjects</td>
<td>(5, 6, 7)</td>
</tr>
<tr>
<td>FetchStation</td>
<td>(5.0, -10.0)</td>
</tr>
<tr>
<td>PickingPlace</td>
<td>(3.0, 2.0)</td>
</tr>
<tr>
<td>DockToRobot</td>
<td>(2.0, 2.0)</td>
</tr>
<tr>
<td>DockFromStation</td>
<td>(10.0, 5.0)</td>
</tr>
<tr>
<td>Pick</td>
<td>(8.0, 8.0)</td>
</tr>
</tbody>
</table>

Table 4: Varying start positions for the service robots in the different experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Robotino</th>
<th>Larry</th>
<th>Macy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-11.0, -14.0)</td>
<td>(3.0, 12.0)</td>
<td>(1.0, 1.0)</td>
</tr>
<tr>
<td>2</td>
<td>(1.0, 1.0)</td>
<td>(-2.0, -8.0)</td>
<td>(-11.0, -14.0)</td>
</tr>
<tr>
<td>3</td>
<td>(-11.0, -14.0)</td>
<td>(-2.0, -8.0)</td>
<td>(1.0, 1.0)</td>
</tr>
</tbody>
</table>
6 RELATED WORK

The works in [SLL**15] and [LIRS**14] illustrate examples for variability management in service robot systems by relying on model-driven tools. The general approach of including particular solvers at decision points of task-trees at run-time containing modeled variability at design-time is similar to here. However, we now focus on composition of building blocks and their properties. Variability management plays an important role in all kinds of software-intensive systems and is therefore a very complex domain. A comprehensive survey of different approaches and their classification is given in [KBR**18] for the domain of self-adaptive systems. Overlaps can also be found in the area of dynamic software product lines for which [GSSC15] and [BBD17] provide detailed examinations and categorisations. Also the work [ABG**13] presents a thorough literature review of software architecture optimization methods, whose collected approaches partially correlate in motivation and methodology with the approaches from the aforementioned references.

7 CONCLUSION

We presented a concrete order picking use case with specific requirements for which suitable service robots and teams of service robots must be determined depending on their properties. The use case is realized based on a general approach for variability management in a robotics software ecosystem. The general approach does not only allow that now service robot resources can be determined automatically according to specified requirements of tasks but also reduces the modeling effort of these problems significantly by utilizing composition and separation of roles. In the future, we will extend and further generalize the models of the presented use case by removing some made assumptions.

8 ACKNOWLEDGMENTS

This paper is supported by EFRE Program Baden-Württemberg 2014-2020 (ZAFH Intralogistik).

REFERENCES


