Work With What You've Got: An Approach for Resource-driven Adaptation

Paul A. Akiki
School of Computing and Communications
The Open University
Milton Keynes, United Kingdom
paul.akiki@open.ac.uk

Andrea Zisman
School of Computing and Communications
The Open University
Milton Keynes, United Kingdom
andrea.zisman@open.ac.uk

Amel Bennaceur
School of Computing and Communications
The Open University
Milton Keynes, United Kingdom
amel.bennaceur@open.ac.uk

***Abstract*—Resource-driven systems are affected by resource variability, which prevents the timely completion of important tasks. This paper presents BOND, a hyBrid resOurce-driveN aDaptation approach which addresses the issue of resource variability by (i) prioritising tasks and making resources available for tasks with higher priorities, (ii) considering alternative task executions when resources are not available, (iii) substituting resources with alternative ones, and (iv) changing tasks into similar ones. The approach supports a proactive and reactive adaptation plan. A prototype tool has been implemented as a proof of concept and used for an initial evaluation of the approach in terms of its feasibility and scalability.**

Keywords—Resource-driven Adaptive System, Self-Adaptive System, Task Prioritisation

I. Introduction

A Resource-driven System (RS) consists of tasks that represent activities [1], which are bound by limited resources. For example, automated warehouses rely on robots as resources for performing tasks such as preparing customer orders for delivery; car manufacturers rely on parts to assemble cars; while recipes rely on available ingredients. The variability of resources due to reasons such as unexpected hardware failures, excess of workloads, or lack of raw materials prevents RSs to execute important tasks on time. In addition, it is costly to compensate for short-term resource variability by over-provisioning resources.

Adaptation enables an RS to continue operating with its limited resources. A Resource-driven Adaptive System (RAS) is a type of Self-Adaptive System (SAS) [2]–[5], where adaptation is triggered by the variability of resources [6]. For example, the variability of available robots at a warehouse prevents the timely fulfilment of customer orders and incurs financial losses. To prevent this, a RAS could adapt the way order preparation is performed, to speed up the work, by allocating order preparations to other robots that may be executing tasks with lower priorities.

Collective Adaptive Systems (CASs) are ensembles of collaborating entities that adapt their behaviour to accommodate changes in their environment [7]. One example of these changes is concerned with changes in the available resources. Therefore, CASs would benefit from a resource-driven adaptation approach to operate successfully, as per the one presented in this paper.

Several resource-driven adaptation approaches have been proposed. These approaches adapt RSs in different ways such as disabling optional components [8] and reducing the data returned by a query [9]. RASs would have more versatility in coping with resource variability by supporting multiple adaptation types. For example, by substituting resources with alternative ones, by changing the order of or delaying the execution of some tasks, or even by changing the tasks themselves into similar ones to cope with available resources. Additionally, RASs would benefit from task prioritisation to decide which tasks to adapt during situations of resource variability. Some scheduling approaches support task prioritisation [10] but do not consider adaptation or multiple prioritisation criteria such as variations of a task based on its parameters and the role of the user who is executing the task. Furthermore, existing approaches support specific resource types such as CPU [8] and battery [11] and could be more useful if they support various resource types [12].

This paper presents BOND (hyBrid resOurce-driveN aDaptation), a resource-driven adaptation approach, which supports RSs in handling variability of limited resources by: (i) making resources available for tasks in the cases where they are most needed, and (ii) considering different viable task execution options when resources are unavailable. BOND addresses resource variability by (a) prioritising tasks using multiple criteria and (b) applying the least costly adaptation types to low-priority tasks. The approach supports both proactive and reactive adaptation plans. In this paper, we focus on the proactive plan.

The remainder of this paper is structured as follows. Section II presents an account of related work and discusses its strengths and shortcomings. Section III presents a motivating example. Section IV describes BOND's architecture, concepts, and components. Section V discusses the proposed approach and its initial evaluation. Section VI concludes the paper and provides an overview of future work.

II. Related Work

This section presents a summary and comparison of existing resource-driven adaptation approaches. Some approaches follow the brownout paradigm [12] to handle variability in computational resources of cloud systems due to hardware failures and flash crowds. These approaches temporarily deactivate optional webpage contents [8] and components [13]. As explained in Section I, supporting additional adaptation types makes RASs more versatile. Additionally, since components are shared by different tasks, if RASs make adaptation choices at the component level it would not be possible to vary these choices among tasks. Hence, instead of considering a component to be either optional or mandatory, its importance could be more accurately represented by prioritising the tasks that will use the component.

Some scheduling approaches support prioritisation [10], but they do not consider adaptation. Furthermore, these approaches

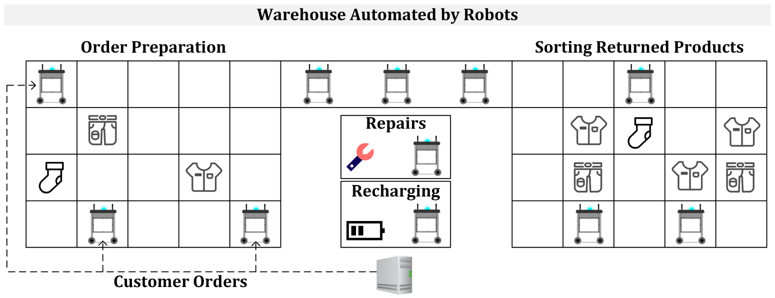


Fig. 1. Motivating Example

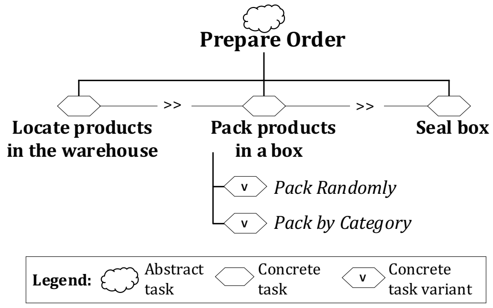


Fig. 2. Example task: Prepare order task tree (excerpt)

order tasks using priority assignment policies of the type deadline-monotonic and rate-monotonic [14], [15], which assign the highest priorities to the tasks with the earliest deadlines and shortest periods, respectively. These policies rely on a single criterion to assign priorities to tasks. When multiple tasks are assigned the same priority, the priority scheduler uses a default mode, for instance a first-come-first-serve prioritisation approach. This is not suitable for applications where task priorities can differ due to multiple criteria like variations of a task based on its parameters, user's role, and even the time of the day.

Other resource-driven adaptation approaches work at the task level [1], [16], but only consider the perspective of a single user who is performing a task, rather than multiple users executing tasks that have different priorities. Resource-driven adaptation must be done at runtime because task priorities and suitable adaptation types are unknown at design time. Therefore, adaptation approaches that perform source code reduction at design time [17], [18] are not suitable.

Runtime adaptation of software systems has been the focus of several works [5]. Some existing works focus on the adaptation of parameters and structure of the system. Parameter adaptation has been used to reduce the data returned by a query [9], support substitution of resources [19], and reduce battery usage [11]. Structure adaptation has been used to reduce RAM usage by modifying source code [17], [18], and reduce mobile battery usage by modifying adaptation rules [20] and altering the autonomic manager [21]. However, these approaches support only one type of adaptation. This is not sufficient when dealing with resource-driven adaptation, given that RSs use various types of resources. Existing works focus on specific resource types such as CPU [8], [13], RAM [17], [18], and battery [9], [11]. Approaches that consider several resource types are useful as suggested in the survey described in [12].

III. Motivating Example

Consider an example of a warehouse that is automated by robots as shown in Fig. 1. The automated warehouse system is an example of CAS. This system is part of a broader logistics scenario involving product delivery among manufacturers, retailers, and customers [7]. Suppose the warehouse is part of a retail store that receives customer orders throughout the day. Robots perform order preparation tasks (Fig. 2) by retrieving from the warehouse the products corresponding to customer orders and packing them in boxes ready for delivery. Consider that robots should place products of the same type together in a pile in boxes. Robots are essential resources for the retail store's operations. Robots can temporarily go out of service due to unexpected errors or due to the need of recharging, thereby, delaying order fulfilment and causing financial losses.

The negative implications of resource variability could be avoided by using BOND to support task prioritisation and adaptation. Preparing customer orders for shipping is a high-priority task, but robots also work on various low-priority tasks such as sorting returned products. Hence, it is possible to substitute inoperative order preparation robots with robots that are working on low-priority tasks that could be delayed. In this way, the order preparation is kept on schedule until the inoperative robots are back in service. This demonstrates situations of resource substitution and prioritisation of tasks with alternative task execution (leaving the sorting of returned products to be executed later).

Another possibility is to alter robots' behaviour by changing the way products are packed. Assume that the robots can pack products in a box using two techniques: (i) placing similar products next to each other in a box (e.g., trousers and shirts in different piles), or (ii) placing products randomly in a box. The first technique provides a better presentation for the customer, while the second technique is executed in a faster way because robots do not have to arrange the products. BOND's ability to support task variants could be used to keep order preparations on track, by using the first technique for the orders of VIP customers and the second technique for other orders. In this case, BOND is supporting changing of tasks with similar ones by allowing products to be packed, but in a random way.

IV. BOND: Proposed Approach

This section describes the proposed resource-driven adaptation approach. An overview of BOND's architecture is given, followed by an explanation of its concepts and components.

A. Architecture Overview

Fig. 3 shows an overview of BOND's architecture, which is based on MAPE-K [22]. The architecture illustrates BOND's stakeholders, components, and data. As shown in Fig. 3, a system administrator uses a tool to define setup data such as resources and task priorities of an RS. The setup data is stored in a knowledge base. The task usage is monitored and logged in a knowledge base as historical data. The adaptation plan is prepared proactively, using setup and historical data, by three components, namely: (i) priority calculator, (ii) priority adjuster, and (iii) adaptation type selector. The use of resources is monitored,

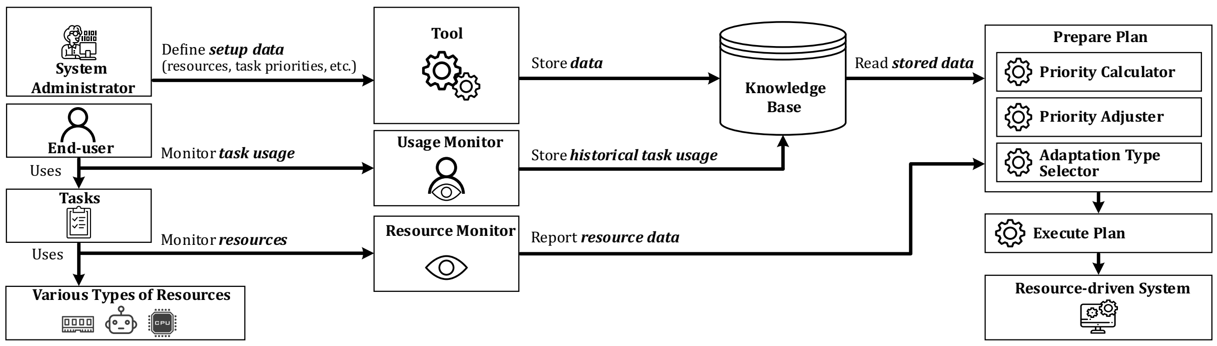


Fig. 3. Summarised overview of BOND's architecture

and the proactively prepared plan is partially updated in a reactive way. The use of hybrid planning (proactive and reactive) helps to avoid the need to recreate a plan reactively over very short periods. This is preferable from a user's point of view [5].

BOND considers different perspectives of stakeholders (e.g., system administrators and end-users). For example, enterprise resources collectively contribute to the accomplishment of tasks that fulfil an enterprise's goals. System administrators offer the managerial view of how a system should adapt to maintain these goals during situations of resource variability (it is common for managers to consider that certain tasks are more important due to financial reasons). End-users' perspectives complement system administrators' perspectives by indicating that an adaptation type is more favourable in certain situations (e.g., VIP customers want to receive products packed in a certain way).

B. *Tasks, Task Variants, and Resources*

A task can be either abstract or concrete. An abstract task has concrete subtasks that realise what it is expected to achieve. In the example given in Section III and shown in Fig. 2, "prepare order" is an abstract task that has three concrete subtasks: locate products in the warehouse, pack products in a box, and seal box. Dividing an abstract task into subtasks is useful for specifying different priorities and variants for the subtasks. The task tree shown in Fig. 2 uses the ConcurTaskTrees (CTT) notation [23] that is adjusted to represent task variants.

A task variant represents a version of a concrete task that is executed with certain parameter values and differs in priority and resource consumption. For example, "pack products in a box" has two variants, one that the robot can complete faster; and another one that takes more time, but offers a better presentation of the packed products. It is not necessary to define task variants for many combinations of possible parameter values. In some cases, parameters have a limited number of possible values. In the case that "pack products in a box" has a parameter called *technique* with two possible values, categorised and random, then this task will only have two variants.

Consider another case where a parameter has an integer value. Instead of creating a task variant for each discrete datum (e.g., 1, 2, 3, …, n), variants are created for continuous data (e.g., 1 to 10) or sets of data (e.g., {1, 5, 20}). In order to illustrate, assume that task "locate products in the warehouse" has an integer parameter representing the robot's speed with values between 1 and 100, where 100 is the fastest. Task variants are represented for meaningful ranges of speed values. There could be one variant for a slow speed with values between 1 and 50, and another variant for a fast speed with values between 51 and 100. The first speed would conserve battery power whereas the second speed would get the robot to its destination faster. Task variants are defined by the system administrators according to the needs of their organisations. The types of resources, e.g., robot, which a task requires are either assigned to a task individually or to a category that groups several tasks.

BOND considers four resource groups based on the analysis of existing literature and examples of resources [8], [17], [19]. These groups include static, dynamic, reusable, and depletable resources. A *static* resource does not have behaviour (e.g., raw materials), whereas a *dynamic* resource has behaviour (e.g., robots). A *reusable* resource is used multiple times (e.g., CPU), whereas a *depletable* resource is used once (e.g., flour). For example, CPU and RAM are static and reusable, IoT devices and robots are dynamic and reusable, and raw materials and fuel are static and depletable. An overview of BOND's concepts of tasks and resources is given by the class diagram shown in Fig. 4.

C. *Priority Calculator*

Priorities play an important role in the adaptation planning process used in BOND because they determine whether tasks are executed by gaining access to resources or are adapted. The first step in task prioritisation involves computing an initial priority using two inputs that provide complementary perspectives: (i) priority assigned by the system administrator, and (ii) priority from forecasted task usage. The calculated priority value (P_v) is a real number between 1 and 3, as shown in (1).

$$P_v = \{ p \mid p \in \mathbb{R} \wedge 1 \leq p \leq 3 \} \quad (1)$$

The system administrator's assigned priority (AAP) takes into account the following criteria: timeframe of the task execution, the role of the user who is attempting to execute the task, and the task variants (Section IV-B). Timeframes represent time intervals that are meaningful for a particular domain. For example, order preparation has a higher priority than sorting returned products during the daytime, when most of the orders are shipped. Roles characterise users and differ among applications. For instance, roles can represent job titles such as warehouse clerk and manager in a warehouse system. Task priorities can differ according to roles because roles indicate that certain users are more privileged, and a task can have a higher priority for those with more privileged roles. A shelf stock count task has a higher priority when requested by a manager.

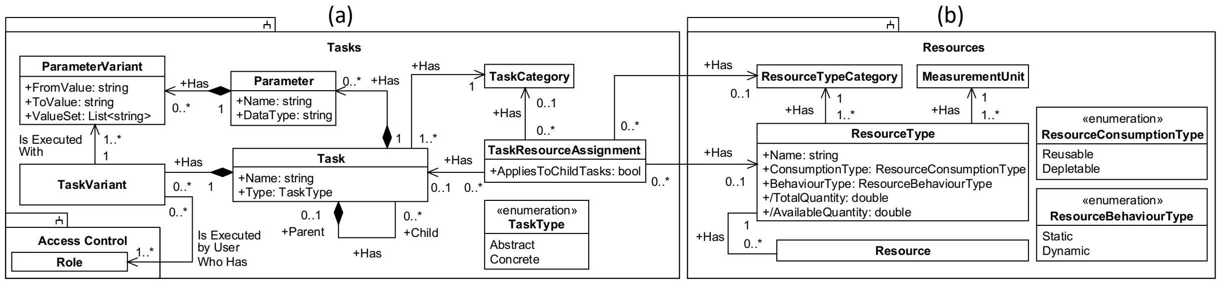


Fig. 4. Summarised overview of BOND's concepts related to tasks and resources

Let *FTUC* be the forecasted task usage count based on historical task usage data. BOND uses thresholds to calculate priority values within the range specified in (1). A threshold (TH) value is calculated as described in (2), whereby tasks with a higher forecasted usage count have a higher priority.

$$TH = \frac{min(FTUC) + max(FTUC)}{max(P_v)} \tag{2}$$

TH is compared to *FTUC* to deduce the threshold forecasted task usage priority (TFTUP) as shown in (3). *TFTUP* represents a task priority value that is between 1 and 3.

$$TFTUP = \begin{cases} 1 & if\ FTUC < TH \\ 2 & TH \le FTUC \le TH \times 2 \\ 3 & FTUC > TH \times 2 \end{cases} \tag{3}$$

The initial priority (P_I) is calculated for a task using *AAP* and *TFTUP*. Each value is multiplied by its corresponding weight as shown in (4), where the sum of the weights is equal to 1.

$$P_I = (AAP \times W_{AAP}) + (TFTUP \times W_{TFTUP}) \tag{4}$$

D. Priority Adjuster

It is possible to have multiple tasks that compete for the same resources with the same initial priority P_I. The priority adjuster is used to further differentiate task priorities (deprioritisation) so that tasks get unique priority values. Interdependent concrete tasks that are mandatory for the completion of the same abstract task are given the same priority. Tasks are grouped by their initial priority and a cost function is applied to adjust the task priorities in each group. The cost of deprioritising a task (C_{DP_T}) is shown in (5), which is measured via four criteria: cost of adaptation (C_{AD}), which is explained in Section IV-E, the sum of adjusted priorities (P_A) from previous timeframes, the total number of dependent tasks (DT), and the estimated execution duration (ED). Each of the inputs is multiplied by its corresponding weight, where the sum of the weights is equal to 1.

$$C_{DP_T} = C_{AD} \times W_{C_{AD}} + \sum P_A \times W_{\sum P_A} + |DT| \times W_{|DT|} - ED \times W_{ED} \tag{5}$$

Grouped tasks are then sorted by the cost function's output, and the initial priority value of each task is increased by an epsilon value ε (i.e., small positive number). The epsilon value is incremented after adding it to a task's initial priority to differentiate the adjusted priorities (P_A) among the grouped tasks.

$$P_A = P_I + \varepsilon \tag{6}$$

Equation (7) is used for calculating ε to ensure P_A is between the *Current Task Group P_I* (CTG_{P_I}) and the *Next Task Group P_I* (NTG_{P_I}). For example, assume there are two groups G_1 and G_2, where G_1 has 4 tasks and P_I is equal to 1, and G_2 has 7 tasks and P_I is equal to 1.1. For each task in G_1, P_A should be exclusively between 1 and 1.1. By (7), ε is equal to 0.02. By (6), the tasks in G_1 have 1.02, 1.04, 1.06, and 1.08 as P_A values, respectively. Therefore, for *n* tasks per each group, the value of P_A for each task in the group will not coincide with the priority values in the next group, making all the P_A values unique among the tasks.

$$\varepsilon = \frac{NTG_{P_I} - CTG_{P_I}}{|CTG| + 1} \tag{7}$$

E. Adaptation Type Selector

After computing task priorities, BOND selects for each task the adaptation types that are viable when resources are unavailable. BOND supports multiple adaptation types that are explained in the following paragraphs.

A *task is changed into a similar one* by executing a task variant with alternative parameter values so a task consumes fewer resources. One example is using an alternative packing method for customer orders (refer to Section III). Another example is related to variants of a "verify order" task, which checks whether the products packed in the box are the ones ordered by the customer. One task variant performs the verification by scanning the products while the second one weighs the box and compares the result to the expected weight. The trade-off here is between accuracy and speed whereby scanning is generally more accurate whereas weighing the box is faster and has a satisfactory accuracy when the products are not very light. The faster variant is beneficial when many robots malfunction unexpectedly.

Resources are substituted with alternative ones so a task can be executed when the resources it requires are not available [19]. For example, in an automated warehouse, a robot that needs repairs is substituted by another type of robot to avoid interrupting high-priority tasks. In a semi-automated warehouse, where humans collaborate with robots, a malfunctioning robot that was supposed to perform a high-priority task can be substituted by a human employee who is working on a low-priority task.

Alternative task executions are considered when resources are not available by postponing the execution of low-priority tasks to another time until the required resources become available. For instance, the low-priority task of sorting returned products is delayed until more robots are available to enable high-priority order preparation tasks to complete on time. Another example involves queuing robot repair tasks to be processed by order of their priority when a single robot repair bay is operational because the others are out-of-service due to machinery malfunctioning or technicians being sick.

The *block* adaptation type is used when no other adaptation type is applicable. An example is blocking optional log data transmission of a robot to a server for conserving battery power. Another example is blocking an "add decoration to box" task that involves adding optional decorative items such as stickers to the packaging. This task is blocked in situations where there is a low stock of decorative items due to an unexpected delay in the supply chain. Therefore, the addition of these optional items is blocked for low-priority orders and kept for the orders of VIP customers. This is an example where the resource is depletable.

The abovementioned adaptation types consider the computed priorities for the tasks. For example, resources of a task are not substituted with alternatives that are needed by higher priority tasks. Another example is delaying a task to a time when resources are available, and it is important enough to execute.

Selecting an adaptation type for a task depends on a cost, which is calculated using a cost function that takes four inputs representing sub-costs related to changing a task's execution schedule (CES), sacrificing functionality (SF), sacrificing quality (SQ), and financing a task's execution (FTE). Each *Input* is represented via a rating scale as shown in (8).

$$Input = \{ v \mid v \in \mathbb{R} \wedge 1 \leq v \leq 5 \} \qquad (8)$$

CES is related to the effect of changing when a task begins execution. Even among low priority tasks, delaying one task can have a lower *CES* than delaying another. *SF* means functionality is not available during the execution of a task at a certain time. A functionality represents a concrete task or the ability to execute it in a certain way. A functionality is sacrificed by performing a block adaptation on a concrete task or by changing a task into a similar one if it has parameters that affect functionality. *SQ* means quality is reduced during the execution of a task at a certain time. For example, using a faster variant of an order verification task could affect accuracy. *FTE* is related to the financial cost of performing an adaptation (e.g., substituting a cheap robot that is malfunctioning with an expensive robot incurs long-term financial costs on an enterprise because robots are assets that are depreciated and replaced over time). *FTE* is a monetary value that is converted to a rating scale value. Each adaptation type affects one or more of the abovementioned costs. The total cost of an adaptation type (C_{AD}) is computed as shown in (9).

$$C_{AD} = CES + \frac{1}{n}\sum_{i=1}^{n} SF_i + \frac{1}{n}\sum_{i=1}^{n} SQ_i + \frac{1}{n}\sum_{i=1}^{n} FTE_i \qquad (9)$$

The cost values are provided by the system administrators as setup data and the end-users through a feedback loop. The system administrators specify their perceived cost, for example, of sacrificing a certain quality for a task. The end-users provide

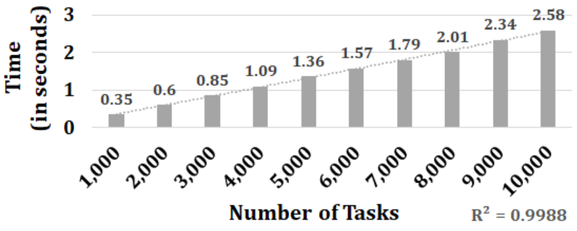
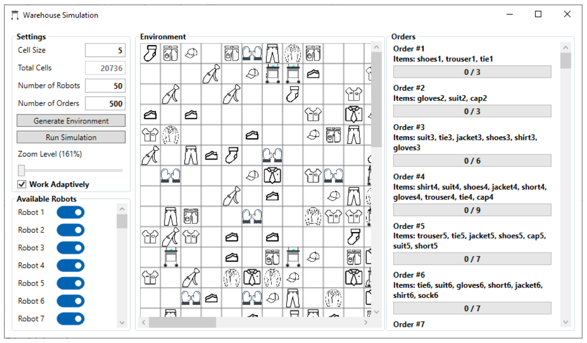


Fig. 5. Running time scalability of proactive adaptation planning



Settings

Cell Size 5

Total Cells 20736

Number of Robots 50

Number of Orders 500

Zoom Level (161%)

☑ Work Adaptively

Orders

Order #1
Items: shoes1, trouser1, tie1
0 / 3

Order #2
Items: gloves2, suit2, cap2
0 / 3

Order #3
Items: suit3, tie3, jacket3, shoes3, shirt3, gloves3
0 / 6

Order #4
Items: shirt4, suit4, shoes4, jacket4, short4, gloves4, trouser4, tie4, cap4
0 / 9

Order #5
Items: trouser5, tie5, jacket5, shoes5, cap5, suit5, short5
0 / 7

Order #6
Items: tie6, suit6, gloves6, short6, jacket6, shirt6, sock6
0 / 7

Order #7

Fig. 6. Simulation of order preparation in an automated warehouse

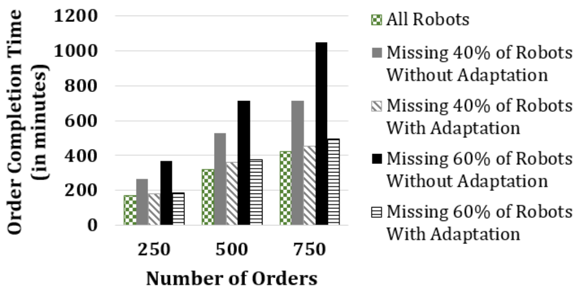


Fig. 7. Order completion during resource variability

their feedback after a task is done to indicate their thoughts about sacrificing the quality that affected their ability to do their work.

V. Evaluation and Discussion

BOND's proactive planning has been implemented as a C# program with 5000 lines of code and a SQL Server database, including the priority calculator, priority adjuster, and adaptation type selector components presented in Fig. 3. ML.NET framework is used for forecasting task usage via a regression algorithm. Task models are represented by extended CTT to accommodate task variants and priorities. The task models are stored in the database and are integrated into the implementation of the software system by associating each task with a corresponding method when a task model is defined. Method calls are filtered to check whether to execute them directly because no adaptation is required, or to execute an updated call after performing an adaptation. If a task variant should be executed the method call's parameters would be changed. Task models are defined during software development and are accessible at runtime to enable the addition of priorities and variants.

The scalability of the proactive planning was evaluated using a varying number of business tasks, which are commonly found in enterprise systems, and generated task usage data. As shown in Fig. 5, the fitting curve of the running time is polynomial with

$R^2 = 0.9988$. This evaluation was done on a Windows 10 computer with a Core i7 1.8 GHz CPU and 16 GB of RAM.

A simulation of an automated warehouse with a grid structure was also developed, as shown in Fig. 6, to demonstrate the feasibility of the proposed approach. This simulation implements the "prepare order" task from the motivating example (Section III), which considers that there are two variants of the packing task with a trade-off between presentation and speed. Orders were generated with random items and the stock items were dispersed across the grid. The robots moved around the grid and located the items of the orders. The simulation included scenarios with full robot capacity and others with missing robots. The system adapts itself when robots are missing by using the faster packing task variant to speed up order preparation. The percentage of packing tasks that are adapted to use the faster variant is reactively adjusted based on the number of missing robots. The chart presented in Fig. 7 shows how adaptation improved the order completion time when robots are unavailable. More diverse scenarios with additional tasks, resources, and adaptation types will be added in the future.

The motivating example presented in this paper considers robots as resource types. However, BOND can work with other types of systems and resources such as web applications and computational resources, respectively. Other case studies are required to demonstrate how BOND works with other types of resources (e.g., depletable). Using historical task usage data and user feedback enables BOND to incorporate a user perspective of priorities and adaptation type costs in the proactively prepared adaptation plan. A limitation is that an RS has to run for a test period to collect this data. To circumvent this limitation, BOND uses the priorities and costs provided by the system administrator until data are available to incorporate user's perspective.

VI. Conclusion and Future Work

This paper presented an overview of BOND, a resource-driven adaptation approach that supports multicriteria task prioritisation and multiple adaptation types. BOND's reactive adaptation part is under development. We are also developing techniques to integrate BOND into RSs and to support system administrators to define task variants and manage priorities using an extended task model. BOND will be evaluated with additional examples and metrics such as the efficiency of reactive adaptation and the intrusiveness of integrating it into RSs.

References

[1] J. P. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw, 'Task-based adaptation for ubiquitous computing', *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 3, pp. 328–340, 2006, doi: 10.1109/TSMCC.2006.871588.

[2] B. H. C. Cheng *et al.*, 'Software Engineering for Self-Adaptive Systems: A Research Roadmap', in *Software Engineering for Self-Adaptive Systems*, vol. 5525, B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26. doi: 10.1007/978-3-642-02161-9_1.

[3] R. De Lemos *et al.*, 'Software engineering for self-adaptive systems: A second research roadmap', in *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, pp. 1–32.

[4] D. Weyns, 'Software engineering of self-adaptive systems: an organised tour and future challenges', *Chapter in Handbook of Software Engineering*, 2017, doi: 10.1007/978-3-030-00262-6_11.

[5] C. Krupitzer, M. Breitbach, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, 'A survey on engineering approaches for self-adaptive systems (extended version)', 2018.

[6] A. Christi, A. Groce, and A. Wellman, 'Building Resource Adaptations via Test-Based Software Minimization: Application, Challenges, and Opportunities', in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 73–78.

[7] A. Bucchiarone *et al.*, 'On the Social Implications of Collective Adaptive Systems', *IEEE Technology and Society Magazine*, vol. 39, no. 3, pp. 36–46, 2020.

[8] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, 'Brownout: building more robust cloud applications', in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, Hyderabad, India, 2014, pp. 700–711. doi: 10.1145/2568225.2568227.

[9] S. Gotz, I. Gerostathopoulos, F. Krikava, A. Shahzada, and R. Spalazzese, 'Adaptive Exchange of Distributed Partial Models@run.time for Highly Dynamic Systems', in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Florence, Italy, May 2015, pp. 64–70.

[10] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, 'A review of priority assignment in real-time systems', *Journal of systems architecture*, vol. 65, pp. 64–82, 2016, doi: 10.1016/j.sysarc.2016.04.002.

[11] S. Yan, S. Li, X. Liu, C. Zeng, X. Liao, and J. Wang, 'Conf-Adaption: Adaptive Adjustment of Software Configuration On UAV by Resource Dependency Analysis', in *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, 2019, pp. 155–161. doi: 10.1109/ITAIC.2019.8785785.

[12] M. Xu and R. Buyya, 'Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions', *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–27, 2019.

[13] M. Maggio, C. Klein, and K.-E. \AArzén, 'Control strategies for predictable brownouts in cloud computing', *IFAC proceedings volumes*, vol. 47, no. 3, pp. 689–694, 2014.

[14] X. Dai and A. Burns, 'Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems', *Journal of Systems Architecture*, vol. 103, p. 101691, 2020.

[15] X. Dai, S. Zhao, Y. Jiang, X. Jiao, X. S. Hu, and W. Chang, 'Fixed-priority scheduling and controller co-design for time-sensitive networks', in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9. doi: 10.1145/3400302.3415715.

[16] P. Rigole, T. Clerckx, Y. Berbers, and K. Coninx, 'Task-driven automated component deployment for ambient intelligence environments', *Pervasive and Mobile Computing*, vol. 3, no. 3, pp. 276–299, 2007.

[17] A. Christi, A. Groce, and R. Gopinath, 'Resource adaptation via test-based software minimization', in *2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2017, pp. 61–70. doi: 10.1109/SASO.2017.15.

[18] A. Christi and A. Groce, 'Target selection for test-based resource adaptation', in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2018, pp. 458–469.

[19] A. Bennaceur, A. Zisman, C. McCormick, D. Barthaud, and B. Nuseibeh, 'Won't Take No for an Answer: Resource-Driven Requirements Adaptation', in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Montreal, QC, Canada, May 2019, pp. 77–88. doi: 10.1109/SEAMS.2019.00019.

[20] G. G. Pascual, M. Pinto, and L. Fuentes, 'Run-time adaptation of mobile applications using genetic algorithms', in *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2013, pp. 73–82.

[21] N. Ali and C. Solis, 'Self-adaptation to mobile resources in service oriented architecture', in *2015 IEEE International Conference on Mobile Services*, 2015, pp. 407–414. doi: 10.1109/MobServ.2015.62.

[22] J. O. Kephart and D. M. Chess, 'The vision of autonomic computing', *Computer*, vol. 36, no. 1, pp. 41–50, 2003, doi: 10.1109/MC.2003.1160055.

[23] G. Mori, F. Paternò, and C. Santoro, 'CTTE: support for developing and analyzing task models for interactive system design', *IEEE Transactions on software engineering*, vol. 28, no. 8, pp. 797–813, 2002.