SERIES: A Task Modelling Notation for Resource-driven Adaptation

Paul A. Akiki, Andrea Zisman and Amel Bennaceur
School of Computing and Communications, The Open University, Milton Keynes, U.K.

Keywords: Task Modelling Notation, Resource-driven Adaptation, Enterprise System.

Abstract: Enterprise Systems (ESs) can make use of tasks that depend on various types of resources such as robots and raw materials. The variability of resources can cause losses to enterprises. For example, the malfunctioning of robots at automated warehouses could delay product deliveries and cause financial losses. These losses can be avoided if resource-driven adaptation is supported. In order to support resource-driven adaptation in ESs, this paper presents a task modelling notation called SERIES, which is used for specifying the tasks of ESs at design time and the enterprise-specific task variants and property values at runtime. SERIES is complemented by a visual tool. We assessed the usability of SERIES using the cognitive dimensions framework. We also evaluated SERIES by developing resource-driven adaptation examples and measuring the performance overhead and source-code intrusiveness. The results showed that SERIES does not hinder performance and is non-intrusive.

1 INTRODUCTION

Enterprise Systems (ESs) can make use of tasks that represent activities, which depend on various types of resources. There are several reasons for resources to be variable. Examples are unexpected hardware failures, excess workloads, or lack of (raw) materials. It is costly to over-provision resources to compensate for short-term resource variability. The lack of resources prevents ESs from executing important tasks on time, thereby, causing losses to enterprises and people. For example, the malfunctioning of robots at automated warehouses could delay product deliveries and drive customers to buy from a competitor. Medically-critical tasks could be obstructed if low-priority tasks deplete medical supplies facing shortages.

Resource-driven adaptation is a type of self-adaptation (Cheng et al., 2009; De Lemos et al., 2013) driven by resource variability. The unavailability of resources to carry out a certain task may trigger an adaptation that involves the execution of a similar task that requires less resources, substitution of alternative resources, execution of tasks in a different order, or even cancelling execution of low-priority tasks.

Existing resource-driven adaptation approaches work in different ways like disabling optional components (Xu & Buyya, 2019), reducing the data returned by a query (Gotz et al., 2015), changing system configurations through policies (Keeney & Cahill, 2003), and reducing source-code that consumes a lot of computational resources (Christi et al., 2017). While the abovementioned approaches do not focus on tasks in their decision-making process, others do (Perttunen et al., 2007; Sousa et al., 2006). However, these approaches do not consider characteristics that are important to support enterprise-specific adaptation decisions, as explained next.

ESs execute tasks that support enterprises from different domains (Lucas et al., 2013). Given that these tasks are different and that the needs of enterprises vary, adaptation decisions should consider characteristics like (i) task priorities; (ii) applicability of an adaptation type to a task; and (iii) task variants that differ according to parameter values, user roles, and resource consumption. The priority of tasks can be different due to corporate decisions. Additionally, parameter values and user roles, that distinguish task variants, differ based on enterprises' data. Hence, a task modelling notation is needed for specifying tasks in ESs at design time and for setting enterprise-specific task variants and properties at runtime.

The abovementioned resource-driven adaptation approaches that focus on tasks do not offer a task modelling notation. Furthermore, existing task modelling notations, like ConcurTaskTrees CTT (Paterno et al., 1997) and HAMSTERS (Martinie et al., 2011), also have missing characteristics (i)-(iii) above that are useful for resource-driven adaptation, as discussed in this paper.

Akiki, P., Zisman, A. and Bennaceur, A.
SERIES: A Task Modelling Notation for Resource-driven Adaptation.
DOI: 10.5220/0011001800003179
In *Proceedings of the 24th International Conference on Enterprise Information Systems (ICEIS 2022) - Volume 2*, pages 29-39
ISBN: 978-989-758-569-2; ISSN: 2184-4992

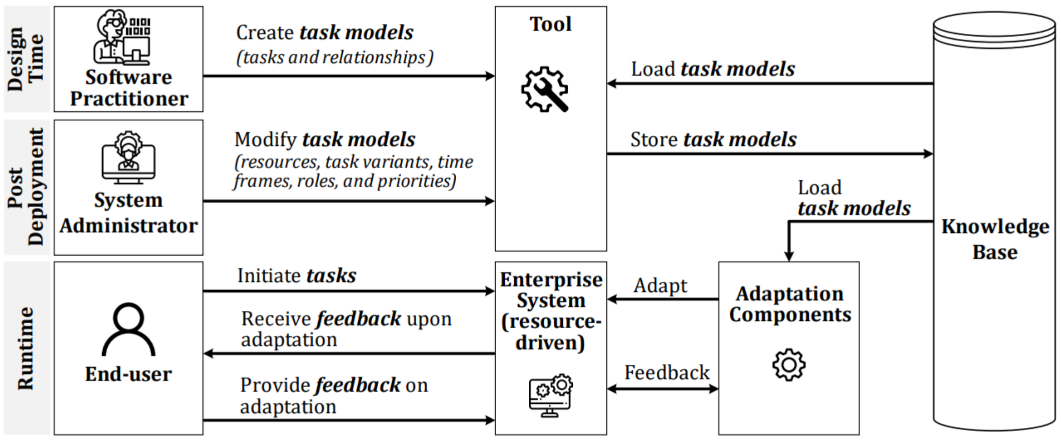

Design Time
Software Practitioner
Create task models (tasks and relationships)
Tool
Load task models
Post Deployment
System Administrator
Modify task models (resources, task variants, time frames, roles, and priorities)
Store task models
Load task models
Knowledge Base
Runtime
End-user
Initiate tasks
Receive feedback upon adaptation
Provide feedback on adaptation
Enterprise System (resource-driven)
Adapt
Feedback
Adaptation Components

Figure 1: Architecture summarising the use of SERIES task models for resource-driven adaptation.

Given the limitations of existing task modelling notations in terms of what they can represent, in this paper, we propose SERIES (taSk modElling notation for Resource-driven adaptatIon of Enterprise Systems). SERIES is a task modelling notation that supports resource-driven adaptation during the development and runtime configuration of ESs. SERIES is based on CTT (Paterno et al., 1997) - a notation for representing task models hierarchically using a graphical syntax. SERIES has a supporting tool to be used by software practitioners and system administrators, as shown in Figure 1. Software practitioners can create, at design time, task models that contain the tasks of an ES; and system administrators can extend these models, at runtime, by adding enterprise-specific information such as task priorities, task variants, and properties related to end-user feedback.

We assessed the usability of SERIES and its tool using the cognitive dimensions framework (Green & Petre, 1996). We also evaluated SERIES by developing resource-driven adaptation examples and measuring its performance overhead and source-code intrusiveness. This evaluation was executed by using adaptation components (Figure 1) we proposed in previous work (Akiki et al., 2021). These components were integrated into the source code of a prototype system using .NET actions filters (Larkin et al., 2021).

The results of the evaluation showed that the use of SERIES does not hinder performance and is non-intrusive. The contributions of this paper are:

- a task modelling notation for resource-driven adaptation; and
- a tool for software practitioners and system administrators to create and modify task models using the proposed notation.

The remainder of this paper is structured as follows. In Section 2 we discuss existing task modelling notations. In Sections 3 and 4 we present SERIES and its supporting tool, respectively. In Section 5 we evaluate the work in terms of usability, performance, and intrusiveness. Finally, in Section 6, we conclude our work and discuss directions for future work.

2 RELATED WORK

This section provides a brief overview of existing task modelling notations and their shortcomings in the development of ESs that use resource-driven adaptation. This section also briefly discusses feature modelling notations.

2.1 Task Modelling Notations

Task modelling notations are used to represent task models, which comprise tasks and relationships that describe how to perform activities. These notations have been used by various model-based development approaches that target user interfaces (Calvary et al., 2003), serious games (Vidani & Chittaro, 2009), and collaborative learning systems (Molina et al., 2014).

Several task modelling notations were proposed as described by existing surveys (Guerrero-García et al., 2012; Limbourg & Vanderdonckt, 2004; Martinie et al., 2019). Some notations like UAN (Hartson & Gray, 1992) and GOMS (Kieras, 2004) are textual, while other notations like CTT and HAMSTERS are graphical. In general, task modelling notations follow a hierarchical structure. Hence, graphical notations are used to visualise a hierarchy of tasks and relationships in a way that is easier to interpret (e.g., by software practitioners and system administrators).

Task modelling notations support various modelling operators and task types. Some notations offer more operators than others. For example, AMBOSS (Giese et al., 2008), HTA (Annett, 2003), GTA (Van Der Veer et al., 1996), and Diane+ (Tarby & Barthet, 1996) have more operators than TSK (Johnson & Hyde, 2003), GOMS, and UAN. CTT and UsiXML (Limbourg et al., 2004) are notations that offer the most types of operators. Examples of these operators include interruption and optionality, which describe tasks and their relationships. With the interruption operator, a task is suspended until another task finishes its work, or a task is completely disabled by another one. The optionality operator either specifies that a task is optional or gives a choice between multiple tasks so that when one task starts the others are disabled. HAMSTERS offers a wider variety of task types and supports extending task types with new ones.

Existing task modelling notations are useful for representing tasks and relationships (Martinie et al., 2019) but are not sufficient for developing ESs that support resource-driven adaptation. Existing notations do not support the association of resource types and priorities with tasks, which are important to identify potentially adaptable tasks due to variations in certain resource types. Furthermore, these notations do not support task variants that differ according to priorities, resource consumption, user roles, and parameters. In situations in which there are variations in resources only tasks with lower priorities would be adapted, rather than adapting the task in all cases.

Another issue is concerned with the lack of stereotypes (tags) indicating which adaptation types apply to a task. If a task requires only a certain type of resource, the system cannot perform resource substitution as an adaptation action and should consider another type of adaptation (e.g., delay the task until the resource becomes available).

Furthermore, in the case of an adaptation action, it is useful to give feedback to end-users about the rationale for the action and to get feedback from them to inform the system whether it should improve its adaptation type choices. However, existing notations do not support properties for specifying how a system should present and receive adaptation-related feedback to and from end-users.

2.2 Feature Modelling Notations

A Feature Model is a hierarchical organisation representing the constraints for valid configurations in a Software Product Line (Hallsteinsen et al., 2008).

Similar to existing task modelling notations, feature modelling notations do not support resource-driven adaptation. More specifically, feature modelling notations do not allow for the representation of resource types; priorities that differ according to parameter values, user roles, timeframes, and resource intensiveness; information that affects which adaptation types are applicable; and configuration information regarding whether and how feedback is elicited from the end-users and presented to them.

While modelling of variants is required to support resource-driven adaptation, those variants are used to distinguish among different ways of executing a task, and not to distinguish among different products. Our aim is to be able to represent explicit concrete tasks rather than high-level features. Task models support temporal operators that are useful for anticipating which task will be executed next and, therefore, support adaptation decisions.

2.3 Why We Chose to Extend CTT

As we explained in Section 2.1, existing task modelling notations have useful features such as tasks and relationships, but they also lack some characteristics that are important for supporting resource-driven adaptation for ESs. Based on our study of existing notations, we realised that notations like CTT, HAMSTERS, and UsiXML could be extended to support resource-driven adaptation. We decided to propose SERIES as an extension of CTT given its wide use in academia, government, and industry: its tool has been downloaded over 26,000 times and has over 10,000 registered users (Vigo et al., 2017).

Additionally, CTT supports useful task types like the system task that SERIES extends with variants. Furthermore, CTT tasks can be associated with User Interface (UI) elements (Calvary et al., 2003). This is useful for displaying adaptation-related feedback to end-users on the UI. Rigole et al. (2007) used CTT in an approach for gradual component deployment to avoid needless consumption of computing resources on mobile devices. However, they do not propose any extensions of CTT as we shall do in this paper.

3 THE SERIES NOTATION

SERIES offers task types and properties that support resource-driven adaptation. Its meta-model is shown in Figure 2. The concepts from this meta-model are illustrated by examples in Figure 3 as excerpts of SERIES task models. Figure 3 shows examples of a retail store website and an automated warehouse system. SERIES task models represent information that is interpreted by resource-driven adaptation components (refer to Figure 1). However, SERIES does not aim to represent procedural logic like the Business Process

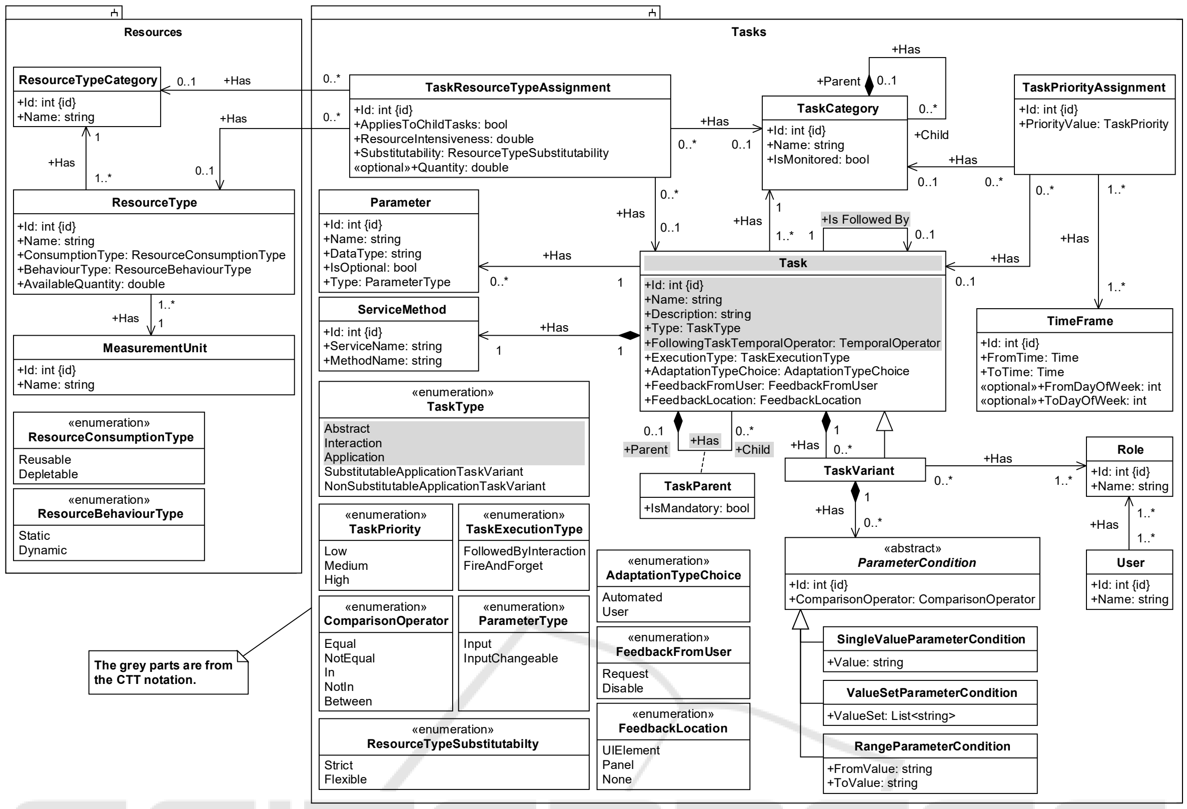

Resources
ResourceTypeCategory
+Id: int {id}
+Name: string
ResourceType
+Id: int {id}
+Name: string
+ConsumptionType: ResourceConsumptionType
+BehaviourType: ResourceBehaviourType
+AvailableQuantity: double
MeasurementUnit
+Id: int {id}
+Name: string
«enumeration» ResourceConsumptionType
Reusable
Depletable
«enumeration» ResourceBehaviourType
Static
Dynamic
The grey parts are from the CTT notation.
Tasks
TaskResourceTypeAssignment
+Id: int {id}
+AppliesToChildTasks: bool
+ResourceIntensiveness: double
+Substitutability: ResourceTypeSubstitutability
«optional»+Quantity: double
TaskCategory
+Id: int {id}
+Name: string
+IsMonitored: bool
TaskPriorityAssignment
+Id: int {id}
+PriorityValue: TaskPriority
Parameter
+Id: int {id}
+Name: string
+DataType: string
+IsOptional: bool
+Type: ParameterType
ServiceMethod
+Id: int {id}
+ServiceName: string
+MethodName: string
Task
+Id: int {id}
+Name: string
+Description: string
+Type: TaskType
+FollowingTaskTemporalOperator: TemporalOperator
+ExecutionType: TaskExecutionType
+AdaptationTypeChoice: AdaptationTypeChoice
+FeedbackFromUser: FeedbackFromUser
+FeedbackLocation: FeedbackLocation
TimeFrame
+Id: int {id}
+FromTime: Time
+ToTime: Time
«optional»+FromDayOfWeek: int
«optional»+ToDayOfWeek: int
«enumeration» TaskType
Abstract
Interaction
Application
SubstitutableApplicationTaskVariant
NonSubstitutableApplicationTaskVariant
TaskParent
+IsMandatory: bool
TaskVariant
Role
+Id: int {id}
+Name: string
User
+Id: int {id}
+Name: string
«enumeration» TaskPriority
Low
Medium
High
«enumeration» TaskExecutionType
FollowedByInteraction
FireAndForget
«enumeration» AdaptationTypeChoice
Automated
User
«abstract» ParameterCondition
+Id: int {id}
+ComparisonOperator: ComparisonOperator
«enumeration» ComparisonOperator
Equal
NotEqual
In
NotIn
Between
«enumeration» ParameterType
Input
InputChangeable
«enumeration» FeedbackFromUser
Request
Disable
SingleValueParameterCondition
+Value: string
ValueSetParameterCondition
+ValueSet: List<string>
«enumeration» ResourceTypeSubstitutabilty
Strict
Flexible
«enumeration» FeedbackLocation
UIElement
Panel
None
RangeParameterCondition
+FromValue: string
+ToValue: string
+Has
+Parent
+Child
+Is Followed By
0..1
0..*
1..*
1

Figure 2: Meta-model of SERIES represented as a class diagram.

Model and Notation (von Rosing et al., 2015). The term behaviour in “ResourceBehaviourType” on the meta-model (Figure 2) is an enumeration for a configuration property and does not indicate that SERIES aims to support procedural logic. In the following text, the names of the meta-model elements are italicised.

3.1 What SERIES Uses from CTT

SERIES incorporates from CTT the concepts of tasks and relationships with temporal operators, which are represented in Figure 2 by the *Task* class and its self-association “*is followed by*” and property *Following-TaskTemporalOperator*. *Tasks* are connected using relationships that are annotated with temporal operators, which express how the *Tasks* relate to each other. As shown in Figure 3(a), the *Task* “Search for Item” enables the *Task* “Get Item Details”.

SERIES also incorporates from CTT three task types: abstract, interaction, and application, which are represented within the *TaskType* enumeration (Figure 2). Abstract tasks require complex actions and are broken down into child (sub) *Tasks*, which are represented on the *Task* class by the *parent-child* self-composition and *ParentTask* association class (Figure 2). Examples of abstract tasks are “Get Item Details” and “Prepare Order” from Figure 3 (a) and (b) respectively. Interaction tasks involve user interactions with the system like “Search for Item” from Figure 3(a). Application tasks are performed entirely by the system. An example of an application *Task* is “Get Item Information” (Figure 3(a)). All the other concepts from the meta-model are characteristics of SERIES.

3.2 SERIES Meta-model

The SERIES meta-model (Figure 2) includes various concepts that are needed for supporting resource-driven adaptation in ESs. These concepts are explained in the following subsections alongside examples from the task models shown in Figure 3.

3.2.1 Resource Types

ResourceTypes represent the varieties of resources required by *Tasks*. A *ResourceType* has a *Consumption-Type* (reusable or depletable) and a *BehaviourType* (static or dynamic). A reusable *ResourceType* is available to another *Task* after the *Task* that is using it is done, whereas a depletable one is used once. A static *ResourceType* does not have a behaviour whereas a dynamic one has a behaviour. Adaptation

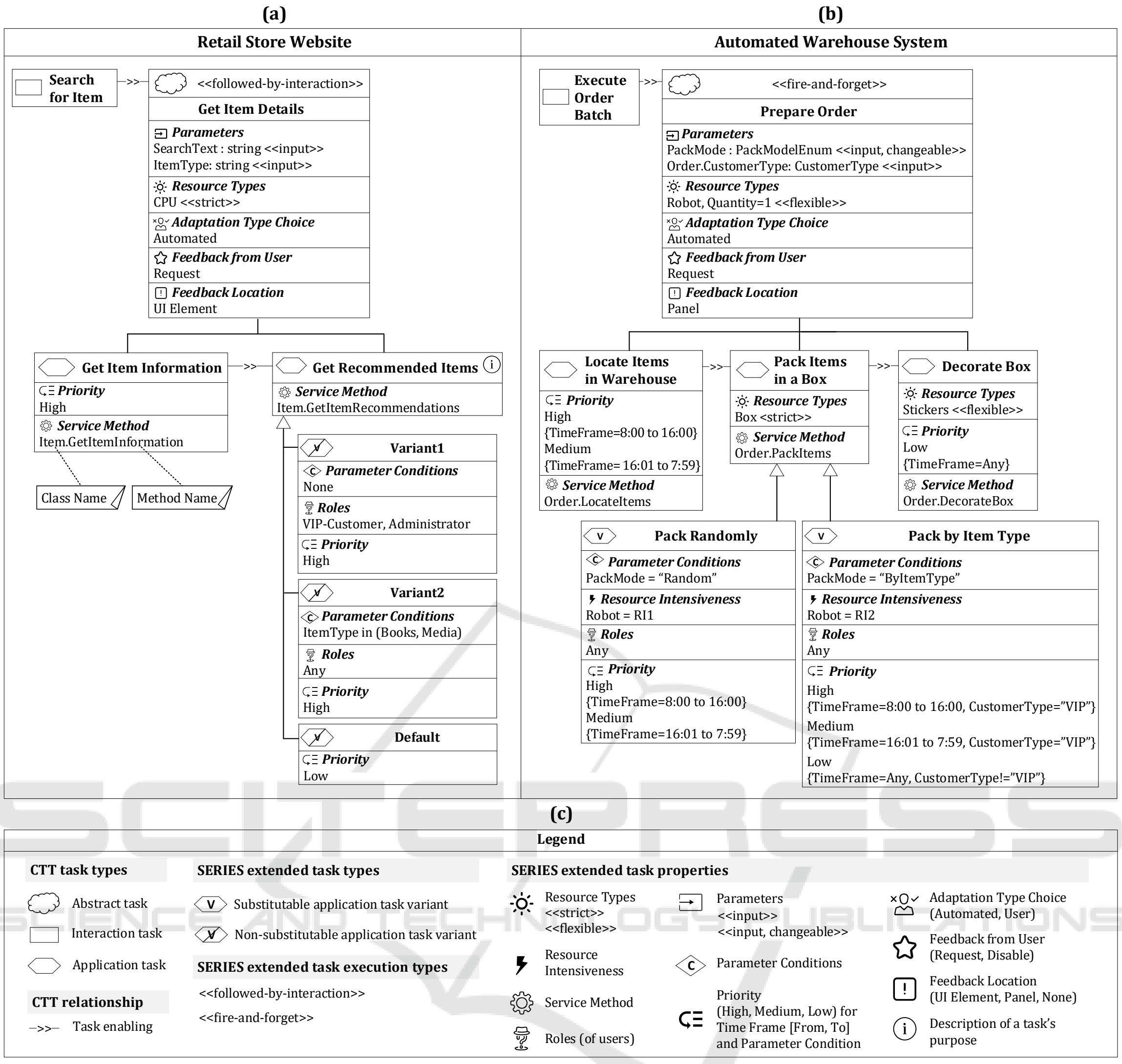


Figure 3: SERIES notation demonstrated by excerpts of two task model examples.

choices are done based on the *ConsumptionType* and *BehaviourType*. For example, certain *Tasks* are preemptively restricted from using scarce depletable resources. A *ResourceType* has an *AvailableQuantity* specified in terms of a *MeasurementUnit* (e.g., litres).

Tasks are associated with their required *Resource-Types* as denoted by *TaskResourceAssignment* in Figure 2. The *ResourceTypes* that are assigned to *Tasks* are marked as either strict or flexible. Strict *Resource-Types* cannot be substituted with alternatives. Hence, in resource variability situations the system should seek another adaptation choice. Flexible *Resource-Types* are substitutable with alternatives.

Example. As Figure 3 shows, the *Tasks* "Get Item Details", "Prepare Order", and "Pack Items in a Box" require CPUs, robots, and boxes respectively. CPUs are reusable static, robots are reusable dynamic, and boxes are depletable static. Additionally, as shown in Figure 3(b), "Box" is marked as strict for the *Task* "Pack Items in a Box" whereas "Stickers" is marked as flexible for the *Task* "Decorate Box".

3.2.2 Service Methods, Parameters, and Task Execution Types

Each *Task* has a *ServiceMethod*, which represents the function in the ES's source code that is called when the *Task* is executed. Method calls are intercepted by the system to make adaptation decisions. The method call's corresponding *Task* is partly identified by relating the method's code name to a *ServiceMethod* in the task model. *Tasks* have *Parameters* that represent data values, which are passed to the corresponding *ServiceMethod* and are available during its execution.

Each *Task* has a *TaskExecutionType* that is either fire-and-forget or followed-by-interaction. The results of a fire-and-forget *Task* can be viewed at a later stage. On the other hand, the results of a followed-by-interaction *Task* are needed directly after the *Task* is done because the end-users need to perform additional interactions with the system based on these results. *TaskExecutionTypes* are used to determine whether delaying a *Task* is a viable adaptation option when resources are unavailable.

Example. As shown in Figure 3(a), the *TaskExecutionType* of "Get Item Details" is "followed-by-interaction". Hence, once the results of this *Task* are shown the end-users will perform interactions like selecting an item from the results or searching for other items. On the other hand, as shown in Figure 3(b), the *TaskExecutionType* of "Prepare Order" is "fire-and-forget" because this *Task* is executed as part of a batch and its results are viewed later.

3.2.3 Task Priorities, Task Variants, Roles, and Parameter Conditions and Types

TaskPriorities indicate the importance of *Tasks* and are needed to decide on which *Tasks* should execute and gain access to the resources they require and which ones should be adapted. A *TaskPriorityAssignment* represents the *TaskPriority* (high, medium, or low) that is assigned to a *Task*. The assigned priority differs by *TimeFrames*, which represent time intervals that are meaningful for a certain enterprise and domain. For example, in an automated warehouse system, order preparation has a higher priority during the daytime, when most of the orders are shipped.

A *TaskVariant* is a special case of a *Task* and is needed to (1) avoid treating all *Task* executions equally when adapting and (2) to identify how to execute a *Task* with fewer resources. *TaskVariants* differ in terms of *Parameter* values (refer to Section 3.2.2), the *Role* of the initiating *User*, *TaskPriority*, and resource consumption, as explained next.

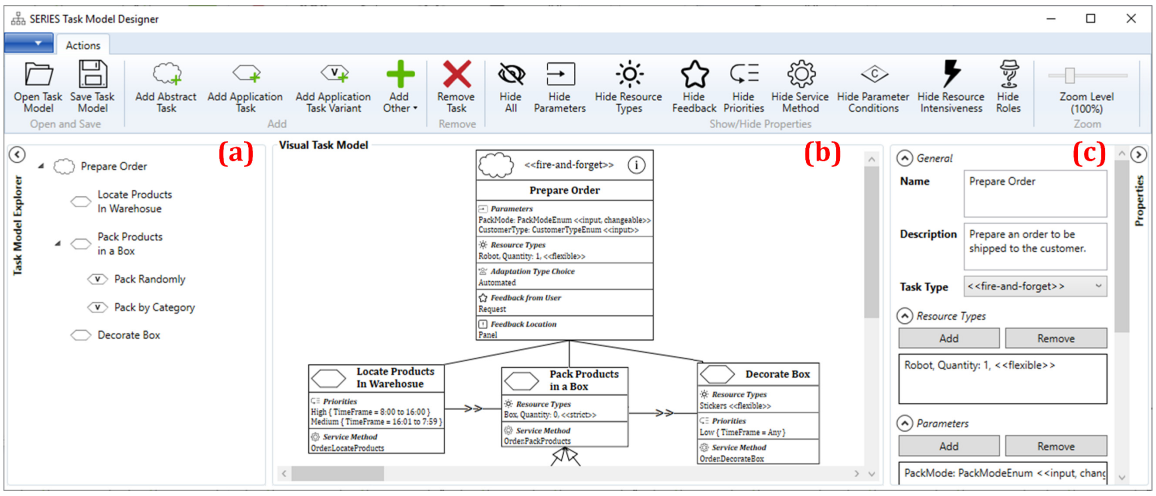
As shown in Figure 2, a *TaskVariant* is associated with *ParameterConditions* that are needed to identify the *Parameter* values that differentiate the variants. A *ParameterCondition* works on a single value, a set of values, or range of values (e.g., "PackMode = Random", "ItemType in (Books, Media)", and "Amount between 1 and 100"). A *TaskVariant* is also associated with *Roles* to indicate its target *Users*. A *TaskVariant* is more important when it is executed by a *User* with a privileged *Role* (e.g., manager). As previously explained, a *Task* is assigned a priority. When *Tasks* have *TaskVariants*, the priorities are assigned to the variants. *ResourceIntensiveness* values are assigned for the *ResourceTypes* that a *TaskVariant* uses. *ResourceIntensiveness* is required to compare *TaskVariants* in terms of resource consumption to adapt by executing the less resource-intensive variants when needed. Examples of *TaskVaraints* are shown in Figure 3 and explained as follows.

Example. The *Task* "Get Recommended Items" has three *TaskVariants* (Figure 3(a)). Instead of deactivating this *Task* for all cases or random *Users*, as is done by brownout approaches for components (Xu & Buyya, 2019), its importance is more accurately represented using prioritised *TaskVariants*. The first variant of "Get Recommended Items" has a high priority for any *Parameter* values when the *Role* of the *User* who is initiating the *Task* is either VIP-customer or administrator. The second variant has a high priority for any *Role* if the value of the "item type" parameter is either books or media, as specified by the *ParameterCondition*. The third (default) variant, which represents all other cases, has a low priority.

Example. The *Task* "Pack Items in a Box" has two variants (Figure 3(b)). The variant "Pack Randomly" instructs robots to pack items in a box in a random order to improve packing speed. The variant "Pack by Item Type" instructs robots to stack items by type (e.g., shirts together). The method of packing items by type provides a better aesthetic outcome but consumes more of a robot's time. Hence, the *ResourceIntensiveness* for the robot is higher for "Pack by Item Type" (RI2) than for "Pack Randomly" (RI1). The variants of "Pack Items in a Box" are differentiated by the value of the "pack mode" *Parameter* that is either "Random" or "ByItemType", as the *ParameterConditions* specify. In this example, the *Priorities* of the variants are the same for any *Role* but differ for *TimeFrames* and customer type. For example, "Pack by Item Type" has a high priority for VIP customers and when a *Task* is executed between 8:00 and 16:00.

TaskVariants are either *Substitutable* or *Nonsubstitutable* as shown by *TaskType* in Figure 2. Both types enable the system to identify different modes of executing a *Task* to avoid treating *Task* executions equally. A *Substitutable TaskVariant* can be executed instead of another to reduce resource consumption.

Example. Figure 3 shows examples of both *Substitutable* and *Nonsubstitutable TaskVariants*. The variants of "Get Recommended Items" are *Nonsubstitutable*. Hence, for example, if the search was issued for items of the type shoes it is not possible to execute the variant related to books and media. On the other hand, the variants of "Pack Items in a Box" are *Substitutable* because it is possible to pack items randomly instead of by type to avoid straining the robots.



Name: Prepare Order

Description: Prepare an order to be shipped to the customer.

Task Type: <<fire-and-forget>>

Robot, Quantity: 1, <<flexible>>

PackMode: PackModeEnum <<input, chang

Figure 4: Tool for creating tasks models using SERIES.

It is possible to execute a task variant that is less resource-intensive. For example, in the case in which robots are malfunctioning due to hardware failures, or when there are excess workloads (e.g., robots are expected to pack orders and to sort returned items, but there is a high number of returned items). In such cases, the execution of a less resource-intensive task variant for low-priority tasks reduces the strain on the robots and dedicates more capacity to the execution of high-priority tasks.

ParameterTypes are needed to decide if it is possible to change a *Parameter's* value when a *Substitutable TaskVariant* is changed to another one. *ParameterTypes* include *Input* and *InputChangeable*. *Input Parameters* are read-only. The system can change the values of *InputChangeable Parameters*.

Example. The *Input Parameter* "SearchText" from "Get Item Details" is read-only (Figure 3(a)). The value of the *InputChangeable Parameter* "Pack Mode" is changed from "ByItemType" to "Random" when the *TaskVariant* "Pack by Item Type" is changed to "Pack Randomly" (Figure 3(b)).

3.2.4 Feedback from and to End-users

A system could keep end-users informed by giving them feedback about the reasons for self-adapting. It could also improve adaptation decisions based on feedback given by end-users to indicate whether an adaptation hindered their ability to perform a *Task*. As shown in Figure 2, a *Task* has three feedback properties: *AdaptationTypeChoice*, *FeedbackFromUser*, and *FeedbackLocation*. These properties are used to configure how the system manages end-user feedback.

The property "*AdaptationTypeChoice*" indicates whether the adaptation type is selected automatically by the system or manually by the end-user (refer to Section 1 for examples of adaptation types). The automated selection of adaptation types relieves end-users from having to frequently make manual choices. Adaptation types can be automatically selected based on costs (Akiki et al., 2021). However, if multiple adaptation types share the lowest cost and the "*AdaptationTypeChoice*" property was set to "User" then the system prompts users to select one of the least costly adaptation types. Otherwise, the system automatically selects one of the least costly adaptation types.

If the property "*FeedbackFromUser*" was set to "Request", then the system would request feedback from end-users on how the adaptations affected their work and use this feedback to adjust the costs of the adaptation types. If a *Task* only has one applicable adaptation type, then this property will be automatically set to "Disable". An example is a *Task* that cannot be delayed because its *ExecutionType* is "followed-by-interaction", its required *ResourceTypes* are strict and hence non-substitutable, it has no substitutable variants, and the only choice for the system is to block it.

The "*FeedbackLocation*" property indicates if the system should display feedback messages to the end-users on a task's corresponding UI or in a separate panel. The first option is suitable when end-users need immediate feedback as they work, while the second option groups feedback so it can be checked later.

Example. If the *Task* "Get Recommended Items" (Figure 3(a)) was blocked from executing, the system could display a message on the UI where the recommendations were supposed to appear. The mapping between task models and UI widgets has been addressed by existing work (Calvary et al., 2003). If the *Task* "Pack by Item Type" (Figure 3(a)) was changed to its variant "Pack Randomly" a feedback message could be displayed in a separate UI panel for the end-users to check after a batch of orders is prepared. End-users give feedback to the system via a rating widget that is shown next to the feedback messages.

3.2.5 Task Categories, Resource Type Categories, and Default Values

TaskCategories and *ResourceTypeCategories* group *Tasks* and *ResourceTypes* respectively (Figure 2). This categorisation is needed to speed up the work of system administrators by assigning *Priorities* and *ResourceTypeCategories* to *TaskCategories* when *Tasks* that belong to the same *TaskCategory* have common *Priorities* and *ResourceTypes*. As shown in Figure 2, the classes *TaskResourceAssignment* and *TaskPriorityAssignment* are associated with *ResourceType* and *ResourceTypeCategory* and *Task* and *TaskCategory* respectively. Therefore, assignments that are done at the level of a category apply to all its corresponding *ResourceTypes* or *Tasks*. For example, assume that several tasks depend on RAM and CPU. System administrators could place these *Tasks* under a *TaskCategory* that is associated with a computer hardware (RAM and CPU) *ResourceTypeCategory*.

The property values that are set on a parent task also apply to its child tasks, unless specified otherwise (overridden).

Example. The *Parameters*, *ResourceTypes*, and feedback properties that are specified for the *Tasks* "Get Item Details" (Figure 3(a)) and "Prepare Order" (Figure 3(b)), apply to their subtasks. Similarly, the *ServiceMethod* specified for the *Task* "Pack Items in a Box" (Figure 3(b)) applies to its variants "Pack Randomly" and "Pack by Item Type".

4 TOOL SUPPORT

SERIES is supported by a visual tool, shown in Figure 4, which enables software practitioners and system administrators to create and modify task models. This tool was developed using C#, and it stores task models as data in a SQL Server database (Figure 1).

The tool is composed of three main panels, namely: (a) Task Model Explorer, (b) Visual Task Model, and (c) Properties. The Task Model Explorer offers a compact hierarchical view of the tasks (Figure 4(a)). The Visual Task Model panel displays task models, using the visual notation of SERIES, and supports dragging, dropping, and selecting elements (Figure 4(b)). The Properties panel, shown in Figure 4(c), allows for specifying the characteristics of tasks.

Software practitioners can use the tool at design time to define the tasks and their relationships based on their systems' specifications. System administrators can modify the task models at runtime based on the needs of their enterprises. User roles and parameter values differ among enterprises and are, therefore, added at runtime. For example, roles like "VIP-Customer" from "Variant1", shown in Figure 3(a), are created by system administrators at runtime via their ESs. Hence, the association of roles with tasks is also done at runtime. Consider that "Get Recommended Items" (Figure 3(a)), could be blocked to reduce CPU consumption. Some enterprises could decide to keep the recommended items for certain item types like books and media as indicated on "Variant2" in Figure 3(a). However, these choices differ among enterprises and are therefore specified at runtime.

5 EVALUATION

We assessed the usability of SERIES and evaluated its performance overhead and source-code intrusiveness as explained in the following subsections.

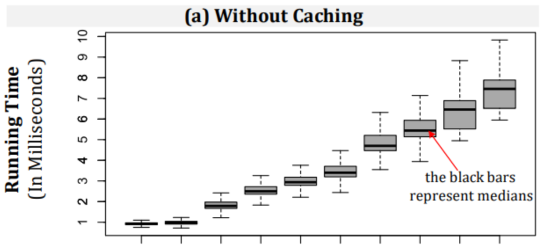
5.1 Usability

SERIES extends CTT and the visual representation of its extended elements is also inspired by UML class diagrams. SERIES uses the shape-based icons of CTT to represent the interaction and application task types. However, it is also possible to use the alternative graphical icons (Paterno et al., 1997). We assessed SERIES based on the recommendations of the cognitive-dimensions framework (Green & Petre, 1996). In the following text, the names of the dimensions are shown in bold.

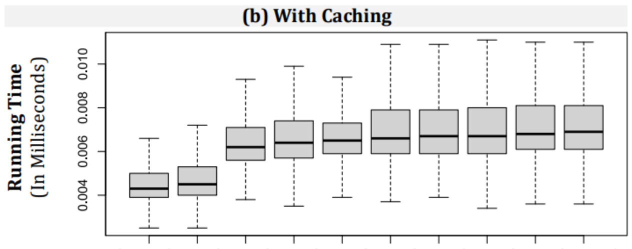
SERIES represents tasks **consistently** using boxes with multiple parts, which are based on UML classes. These box parts represent properties that support resource-driven adaptation (Figure 3(c)).

SERIES is **abstraction-tolerant** since it supports the representation of task models using predefined visual elements of tasks, properties, and relationships. It is not possible to add new visual elements. However, new abstractions are defined in the form of parent tasks that comprise default property values, which do not need to be specified for the child tasks.

A **premature commitment** is not needed since a SERIES task model may comprise part of the tasks and property values at design time. Task variants and properties, like roles and priorities, are set at runtime when their corresponding data is available (refer to



Test Runs	Tasks	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
	Variants Per Task	0	0	2	2	4	4	8	8	16	16
Running Time	Mean	0.95	0.99	1.88	2.62	3.09	3.58	5.03	5.73	6.36	7.36
	Median	0.92	0.97	1.79	2.50	2.94	3.40	4.70	5.44	6.46	7.46
	SD	0.17	0.18	0.42	0.53	0.62	0.75	1.08	1.19	1.05	1.05



Test Runs	Tasks	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
	Variants Per Task	0	0	2	2	4	4	8	8	16	16
Running Time	Mean	0.0047	0.0050	0.0068	0.0069	0.0070	0.0073	0.0074	0.0074	0.0074	0.0075
	Median	0.0043	0.0045	0.0062	0.0064	0.0065	0.0066	0.0067	0.0067	0.0068	0.0069
	SD	0.0031	0.0036	0.0034	0.0036	0.0031	0.0040	0.0042	0.0036	0.0038	0.0043
Cached Model Size(In MB)		0.02	0.04	0.74	0.98	2.32	2.76	6.36	7.26	8.16	17.84

Figure 5: Performance evaluation for the identification of tasks and variants from SERIES task models.

Section 4). Furthermore, a SERIES task model is arranged hierarchically to avoid "visual spaghetti" that could occur with some box-and-line notations. Even in the case in which new tasks are introduced at a later stage, the model is automatically rearranged without needing to look ahead to avoid a messy layout.

Concerning **diffuseness**, each meaning of a task or property in SERIES is denoted by one box part that has an icon and a description that makes it easy to remember. The ability to add default property values at the parent task level reduces the number of properties in the boxes. This improves the overall visibility of the task model and makes the visual notation **terse** (compact) enough to represent multiple tasks on the screen. Furthermore, it is possible to suppress a group of task properties by hiding its box parts as is done with UML class diagrams. Hence, the notation supports different levels of terseness that are changeable according to how much detail a person wants to see.

SERIES does not have complex conditionals that create **hard mental operations**. Parameter conditions are defined as simple textual statements such as "PackMode = Random". Hence, these conditions do not use complex line connections that cause software practitioners and system administrators to resort to tracking what is happening with their fingers.

There are no **hidden dependencies** between the elements of a SERIES task model. The dependencies between tasks are shown as relationships. For example, task variants are linked to their base task using relationships that resemble UML generalisation to indicate that the variants are special cases of a general case. Furthermore, properties like parameter conditions and priorities are shown on the model without hidden formulas (e.g., like the ones in spreadsheets).

Concerning **role-expressiveness**, a task's name indicates its purpose. A description can also be added to a task to further explain its purpose. This description is a **secondary notation** and is viewed by clicking on an information icon, which appears on tasks that have a description as shown on the "Get Recommended Items" task in Figure 3(a).

SERIES has a **low viscosity** since little effort is needed to change tasks and properties using the task model explorer and properties box (refer to Section 4).

5.2 Performance and Intrusiveness

We used SERIES to develop working resource-driven adaptation examples that build on an adaptation approach we proposed in previous work (Akiki et al., 2021). We measured the performance overhead and source-code intrusiveness of SERIES task models.

For evaluating the performance, we used NBomber (*NBomber*, 2021) to simulate user requests to C# web service methods representing tasks in an ES. We measured the time it took to identify the tasks and variants from a SERIES task model corresponding to the service methods being called.

The identification of tasks and variants is a step in the adaptation process concerning how and when an adaptation should occur. This step involves comparing the source-code name of the service method being called to the service methods from the task model, dynamically evaluating whether the parameter conditions from the task model match the parameter values passed to the service method and comparing the role(s) of the end-user to the roles in the task model.

Figure 5 shows the results of this evaluation, including multiple test runs with an increasing number of tasks and variants ranging from 1000 to 10,000 tasks and 0 to 16 variants per task. The evaluation was done with two implementations, one that caches the task models in memory and another one that does not perform caching. NBomber simulated user requests for 10 minutes per test run. The mean running time was measured in milliseconds and ranged between 0.95 and 7.36 without caching and 0.0047 and 0.0075 with caching. Both results show a small overhead that does not hinder a system's performance. For example, web users find it tolerable to wait for 2 to 4 seconds (Nah, 2004). Both fitting curves of the mean running times are polynomial with $R^2 = 0.9924$ (without caching) and 0.9797 (with caching). We favour using caching since it improves performance without burdening the memory (cached model size ranged from 0.02 MB to 17.84 MB—Figure 5(b)). This evaluation was done on a Windows 10 computer with a Core i7 1.8 GHz CPU and 16 GB of RAM.

Intrusiveness is evaluated in terms of the LOC and source-code files that are added or modified to integrate resource-driven adaptation into an ES. If minor changes are needed, then the integration is non-intrusive. It is more desirable to perform the integration with minor changes because this means that less effort is needed and there is less chance of introducing errors into the source code of the ES.

We used .NET actions filters (Larkin et al., 2021) to intercept the service method calls and execute the code that we implemented to identify tasks and variants. Our implementation included one action filter with either 112 LOC (without caching) or 136 LOC (with caching), which are global for the entire system. We can say that the integration is non-intrusive because we only needed to add 136 LOC in two source-code files to make our approach work for any number of tasks. We did not modify task-specific source code (e.g., business logic). This makes it easier to integrate resource-driven adaptation capabilities into ESs that comprise thousands of tasks. If we had to add or modify LOC in several source-code files per task, then the integration would become more intrusive as the number of tasks increases, due to the widespread changes.

6 CONCLUSION AND FUTURE WORK

This paper presented a task modelling notation called SERIES, which offers task types and properties required for the development of ESs that support resource-driven adaptation. SERIES is complemented by a tool that enables software practitioners to define task models at design time, and system administrators to extend these models at runtime, based on enterprises' needs.

We assessed the usability of SERIES following the cognitive dimensions framework (Green & Petre, 1996). We also evaluated SERIES by using it to develop working examples that build on an existing resource-driven adaptation approach and measure its performance overhead and source-code intrusiveness. The results showed that SERIES does not hinder performance and is non-intrusive. Although we evaluated SERIES with an adaptation approach, this notation is also useful for other approaches.

In the future, we aim to extend the tool by supporting the generation of code that corresponds to the tasks' service methods. We also aim to implement a technique that reads the feedback properties from SERIES task models to present and elicit adaptation-related feedback to and from end-users. It could be possible to extend SERIES with additional properties after investigating end-user feedback preferences. Other areas for future work include usability study of SERIES with software practitioners and the use of the Physics of Notations (Moody, 2009) as part of our evaluation. We also plan to extend the evaluation to include case studies across multiple domains.

ACKNOWLEDGEMENTS

This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/V026747/1, EP/R013144/1].

REFERENCES

Akiki, P. A., Zisman, A., & Bennaceur, A. (2021). Work With What You've Got: An Approach for Resource-Driven Adaptation. *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, 105–110.

Annett, J. (2003). Hierarchical task analysis. *Handbook of Cognitive Task Design*, *2*, 17–35.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, *15*(3), 289–308.

Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., … Whittle, J. (2009). Software Engineering for Self-Adaptive Systems: A Research Roadmap. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, & J. Magee (Eds.), *Software Engineering for Self-Adaptive Systems* (Vol. 5525, pp. 1–26). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-02161-9_1

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

De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., & others. (2013). Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II* (pp. 1–32). Springer.

Giese, M., Mistrzyk, T., Pfau, A., Szwillus, G., & Von Detten, M. (2008). AMBOSS: a task modeling approach for safety-critical systems. In *Engineering Interactive Systems* (pp. 98–109). Springer.

Gotz, S., Gerostathopoulos, I., Krikava, F., Shahzada, A., & Spalazzese, R. (2015). Adaptive Exchange of Distributed Partial Models@run.time for Highly Dynamic Systems. *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 64–70. https://doi.org/10.1109/SEAMS.2015.25

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, *7*(2), 131–174.

Guerrero-García, J., González-Calleros, J., & Vanderdonckt, J. (2012). A Comparative Analysis of Task Modeling Notations. *Acta Universitaria*, *22*, 90–97.

Hallsteinsen, S., Hinchey, M., Park, S., & Schmid, K. (2008). Dynamic software product lines. *Computer*, *41*(4), 93–95.

Hartson, H. R., & Gray, P. D. (1992). Temporal aspects of tasks in the user action notation. *Human-Computer Interaction*, *7*(1), 1–45.

Johnson, H., & Hyde, J. (2003). Towards modeling individual and collaborative construction of jigsaws using task knowledge structures (TKS). *ACM Transactions on Computer-Human Interaction (TOCHI)*, *10*(4), 339–387.

Keeney, J., & Cahill, V. (2003). Chisel: A policy-driven, context-aware, dynamic adaptation framework. *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 3–14.

Kieras, D. (2004). *GOMS Models for Task Analysis. The Handbook of Task Analysis for Human-Computer Interaction, Ed. Dan Diaper, Neville A. Stanton*. Lawrence Erlbaum Associates.

Larkin, K., Anderson, R., Dykstra, T., & Smith, S. (2021). *Filters in ASP.NET Core*. https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-5.0

Limbourg, Q., & Vanderdonckt, J. (2004). Comparing task models for user interface design. *The Handbook of Task Analysis for Human-Computer Interaction*, *6*, 135–154.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., & Lopez-Jaquero, V. (2004). USIXML: A language supporting multi-path development of user interfaces. *IFIP International Conference on Engineering for Human-Computer Interaction*, 200–220.

Lucas, W. T., Xu, J., & Babaian, T. (2013). Visualizing ERP Usage Logs in Real Time. *ICEIS (3)*, 83–90.

Martinie, C., Palanque, P., Bouzekri, E., Cockburn, A., Canny, A., & Barboni, E. (2019). Analysing and demonstrating tool-supported customizable task notations. *Proceedings of the ACM on Human-Computer Interaction*, *3*(EICS), 1–26.

Martinie, C., Palanque, P., & Winckler, M. (2011). Structuring and composition mechanisms to address scalability issues in task models. *IFIP Conference on Human-Computer Interaction*, 589–609.

Molina, A. I., Redondo, M. A., Ortega, M., & Lacave, C. (2014). Evaluating a graphical notation for modeling collaborative learning activities: A family of experiments. *Science of Computer Programming*, *88*, 54–81.

Moody, D. (2009). The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, *35*(6), 756–779.

Nah, F. F.-H. (2004). A study on tolerable waiting time: How long are web users willing to wait? *Behaviour & Information Technology*, *23*(3), 153–163.

NBomber. (2021). https://nbomber.com/

Paterno, F., Mancini, C., & Meniconi, S. (1997). ConcurTaskTrees: A diagrammatic notation for specifying task models. *Human-Computer Interaction INTERACT'97*, 362–369.

Perttunen, M., Jurmu, M., & Riekki, J. (2007). A QoS model for task-based service composition. *Proc. 4th International Workshop on Managing Ubiquitous Communications and Services*, 11.

Rigole, P., Clerckx, T., Berbers, Y., & Coninx, K. (2007). Task-driven automated component deployment for ambient intelligence environments. *Pervasive and Mobile Computing*, *3*(3), 276–299.

Sousa, J. P., Poladian, V., Garlan, D., Schmerl, B., & Shaw, M. (2006). Task-based adaptation for ubiquitous computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *36*(3), 328–340. https://doi.org/10.1109/TSMCC.2006.871588

Tarby, J.-C., & Barthet, M.-F. (1996). The DIANE+ Method. *CADUI*, *96*, 95–119.

Van Der Veer, G. C., Lenting, B. F., & Bergevoet, B. A. (1996). GTA: Groupware task analysis—Modeling complexity. *Acta Psychologica*, *91*(3), 297–322.

Vidani, A. C., & Chittaro, L. (2009). Using a task modeling formalism in the design of serious games for emergency medical procedures. *2009 Conference in Games and Virtual Worlds for Serious Applications*, 95–102.

Vigo, M., Santoro, C., & Paternò, F. (2017). The usability of task modeling tools. *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 95–99.

von Rosing, M., White, S., Cummins, F., & de Man, H. (2015). *Business Process Model and Notation-BPMN.*

Xu, M., & Buyya, R. (2019). Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions. *ACM Computing Surveys (CSUR)*, *52*(1), 1–27.