Using Dynamic Workflows for Coordinating Self-adaptation of Software Systems

Carlos Eduardo da Silva
University of Kent
Computing Laboratory
Canterbury, Kent, CT2 7NF, UK
ces26@kent.ac.uk

Rogério de Lemos
University of Coimbra
Dept. of Informatics Engineering
3030-290 Coimbra, Portugal
rdelemos@dei.uc.pt

Abstract

The self-adaptation of a software system is a complex process that depends on the system requirements, its operational state and environment, which may change during the system operational lifetime. Hence, the process for coordinating the self-adaptation should also be adaptable to changes that may occur during run-time. As a means for coordinating the self-adaptation process of software systems, the proposed approach employs workflows that are dynamically generated for dealing with the variability associated with the self-adaptation process. In this context, our aim is to define and develop techniques for automatically generate workflows for coordinating the self-adaptation of software systems. For demonstrating the feasibility of the proposed approach, architectural reconfiguration of software systems is used as an example, whereby the reconfiguration is managed by workflows that are dynamically generated depending on the availability of resources.

1. Introduction

It is commonly agreed that a self-adaptive software system is able to modify its own structure and/or behaviour at run-time due to changes in the system, its requirements, or the environment in which it is deployed. Compared to traditional systems, one of the features of self-adaptive software systems is that some design decisions are moved from development-time towards run-time. Because of this, processes usually executed in a controlled environment at development-time, need now to be executed at run-time in a dynamic and distributed environment without any human intervention. In order to determine the actions to be taken to adapt itself, a self-adaptive software system observes and analyses itself and its environment, and if an adaptation is deemed to be necessary, a plan is generated for altering the system in a controlled manner. These activities are usually captured in terms of a feedback control loop containing four key phases, that is, monitoring, analysis, planning and execution [17].

Since self-adaptation is a complex process that can impact multiple system components, some kind of coordination is required for carrying out the adaptation in a concerted fashion. As a means for coordinating the self-adaptation process, we employ in this paper workflow management technology. Typically, workflow management technology coordinates and controls the flow of work and information between participants of business processes [25]. A workflow represents a business process in terms of activities (or tasks) and their relationships, rules and control data in a form that can be manipulated by a Workflow Management System (WfMS). A WfMS is a software system that provides support for the definition, management and execution of workflows. The process associated with the self-adaptation of software depends on several factors that may change during the system operational lifetime, thus it is expected that the workflow for coordinating the self-adaptation should also adapt depending on changes that may occur. In order to handle the variability associated with the self-adaptation process, we propose the use of dynamic workflows. In this context, workflow management technology can be used to coordinate the process of self-adaptation by representing the adaptation plan as a workflow, and using a WfMS for its coordination. This adaptation plan should be dynamically generated to deal effectively with the variability and uncertainty involved in the self-adaptation process. Otherwise, the self-adaptation process would cease to be in synchrony with the actual system that it controls, thus restricting what and how can be adapted. Hence the motivation for defining dynamic processes that would enable a

*Carlos Eduardo da Silva is supported by the Programme Al\’yan, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E07D401107BR.
system to be more flexible to what it should be able to adapt. Otherwise, the adaptation process may become brittle if the operational profile of the system is not fully considered during development-time.

In this context, the aim of our work is to design and develop mechanisms for automatically generate workflows for coordinating the self-adaptation of software systems. In this paper, we present our efforts in this direction, introducing an approach for generating workflows dynamically, which can then be applied, as an example, to the architectural reconfiguration of software systems. Based on the proposed approach, the process for managing architectural reconfiguration is dynamically modified depending on the availability of resources. In order to evaluate the feasibility of the proposed approach, we have developed a web service application that relies on the dynamic reconfiguration of its architecture for the provision of dependable services. Based on the system requirements, alternative configurations, and the attributes of the available resources, the appropriate tasks are composed in terms of workflows for coordinating the architectural reconfiguration of software systems.

The rest of this paper is organised as follows. Section 2 presents some background information related to workflows and dynamic workflows. Our model for workflow generation is presented in section 3. Section 4 presents the adaptation process applied to the architectural reconfiguration of fault-tolerant software systems. Section 5, for demonstrating the feasibility of the proposed approach, presents a case study and some preliminary results. Section 6 discuss some related work and the conclusions of the paper are presented in section 7.

2. Background

2.1. Workflow Management Technology

Workflow management technologies allow people and organization to automate, manage, and improve business processes by coordinating and controlling the flow of work and information between participants. A business process is defined as a set of one or more linked procedures or tasks which collectively realise a business objective or policy goal, defining functional roles and relationships. The business process is defined through a workflow (or process definition) in a form which supports automated manipulation or enactment by a Workflow Management System (WfMS).

A workflow identifies the various tasks and their relationships, rules and control data, and may contain references to sub-processes, separately defined, which make up part of the overall process definition. A WfMS is a software system which provides support for the definition, management and execution of workflows through the use of software running on one or more workflow engines, which is able to interpret the workflow, interact with workflow participants, and when required invoke the use of IT tools and applications [25].

Workflow management technology has reached a significant level of maturity by supporting heterogeneous environments and technologies, and different distributed computing architectures. In addition to that, workflows support the specification of complex processes by using modelling languages that are able to express complicated coordination patterns [24]. Workflow management technology has been applied to a variety of domains beyond traditional business processes [12], and it has also been shown to be a suitable technology for coordinating software self-adaptation [20] [22].

2.2 Dynamic Workflows

Dynamic workflows provide support for modifying the workflow specification during run-time in order to handle changing requirements, availability of resources, and different working contexts. This is achieved with the support of the WfMS, which should be able to respond to changes that might occur in the workflow specification [13]. The implementation of dynamic workflows varies from the dynamic restructuring of the workflow activities to the support for flexible composition and flexible deployment. Flexible composition can be achieved through alternative inputs or outputs [11], and tasks that can be dynamically instantiated until the definition of the workflow is needed [1] [6] [13]. The flexible deployment may be supported through late binding mechanisms [11], which allow the completion of activities using available resources when they are actually needed [12] [13]. Our approach follows the techniques associated with flexible composition mechanisms where tasks are dynamically instantiated when needed. However, instead of selecting the tasks from a repository of pre-specified tasks, the objective of the proposed approach is to dynamically generate these tasks.

The techniques for the generation of workflows can be divided in static and dynamic [2]. Static techniques receive as input an abstract model of the tasks that should be carried-out. This abstract workflow describes the set of tasks and the data dependency among them, and the generation process consists of selecting the actual resources for executing the tasks specified in the abstract model, generating a concrete workflow which identifies the actual resources associated with the tasks. On the other hand, dynamic generation techniques automatically create the abstract process model and select the appropriate tasks that define the workflow. The generation of abstract workflows is usually achieved based on a library of tasks with associated pre- and post-conditions, where different techniques can be applied for finding the sequence of tasks for a par-
ticular objective [3] [16]. The mapping of an abstract workflow to a concrete workflow involves choosing the appropriate resources for the specified tasks respecting the data dependency between them. One of the approaches for generating concrete workflows is to define abstract tasks with query clauses that will be used for identifying the actual resource [2], or using logical information in the tasks to query some type of registry service [6] [15].

3. Model and Infrastructure for Workflow Generation

Workflow generation is a composition process involving tasks and the dynamic interconnection of tasks, in a way that the sequence of tasks achieves a particular objective. This is similar to the process for dynamically reconfigure software architectures in which the interconnection between components may change on the fly. Hence the similarities between workflows and architectural configurations in which tasks are equivalent to components, and the connectors are equivalent to interconnections between tasks. In this view, some of the concepts and techniques applied in the context of dynamic reconfiguration of software architectures can be easily adapted for dealing with dynamic workflows. Based on this, we have adopted as a basis for generating dynamic workflows the three-layer reference model for self-managed systems introduced by Kramer and Magee [14]. Taking as a basis the three-layer reference model, the resources to be controlled at the component control layer are the workflow tasks instead of components. At the change management layer, the plans to be managed are workflows themselves. Finally, at goals management layer, goals are objectives associated with workflows, from which workflows are generated.

Based on the three-layer reference model, the proposed approach for generating dynamic workflows can be summarised as follows. At the goals management layer, abstract workflows are generated according to a particular objective (or goal). These workflows are used as a basis for generating the concrete workflows, which correspond to the plans of the change management layer. Once a concrete workflow has been selected, it is executed at the component control layer. In case of an error during the execution of a concrete workflow, a new concrete workflow is generated. If it is not possible to generate a concrete workflow based on a particular abstract workflow, a new abstract workflow should then be identified.

3.1. Model for Workflow Generation

Our model for workflow generation is based on a wide range of concepts and techniques from other domains [9], [16] [19]. In our approach, the generation of an abstract workflow considers a library of tasks templates (with their respective pre- and post-conditions), the workflow objective, and the workflow initial state. The process of generating workflows consists in finding a sequence of tasks templates, starting from the workflow initial state, which is able to meet the workflow objective. This is achieved by using a backward search from the workflow objective (which should correspond to the post-condition of a task template) to the workflow initial state (which should correspond to the pre-condition of a task template) through a breadth-first search algorithm, which returns the first workflow found (the shortest one), as long as there is one. For that, we modify and combine the algorithms presented in [16] and [19] for generating abstract workflows from task templates. It is important to note that at this level, the resources associated with the tasks are referred to by a logical name, which should be sufficient to identify the actual resources at the next stage.

A task template captures through a sub-workflow any operation, or sequence of operations. In our model, every task template is a sub-workflow with two possible outcomes: commit or failure. A commit represents the successful execution of the associated operation meaning that the post-condition of the task template has been met. When the operation fails, the task template finishes with a failure, which signals that the post-condition has not been met. A task template also allows to include an exception handling path in sub-workflow for dealing with the possible failures in the execution of the operation. In our model, exceptions are not propagated outside the tasks templates.

Every generated workflow consists also of two possible outcomes: commit or failure. They are structured in such a way that, whenever there is a failure in one of its tasks, the workflow stops its execution and finishes. For that, the generated workflows are structured in a manner similar to the approach presented by Andrzejak et al. [3] in which a Failure task represents a kind of ”guard” task that is activated when one of the tasks fails, thus cancelling the execution of all remaining tasks.

Once an abstract workflow has been identified, the next step is to map this abstract workflow into a concrete one. This mapping consists in populating the tasks in the abstract workflow with the actual resources that will be used for executing the task. Since the actual resources are associated with the logical names, the process verifies the state of these resources to determine the necessity of augmenting the workflow with other tasks, or reducing it by eliminating tasks for which the desired workflow objective can be achieved by using a subset of the resources.

The separation between abstract and concrete workflows is similar to that of strategies and tactics [7]. This separation is interpreted as follows. Once a strategy has been selected, it can be implemented using different tactics. At the strategy
level, we establish the composition of tasks without identifying the actual resources that are used by the workflow. In this way, the abstract workflow can be implemented using different combination of resources (which is done at the tactics level). At the tactics level, resources are allocated to each task of the abstract workflow for defining the concrete workflow.

3.2. Infrastructure for Workflow Generation

The diagram of Figure 1 presents an overview of the infrastructure for dynamically generating workflows. The diagram is presented in the context of architectural reconfiguration, which will be detailed in section 4.1. The Task library is a repository where all task templates with its associated pre- and post-conditions are stored, and the Workflow generator is responsible for generating the abstract and concrete workflows. In the present context, the generation process is controlled by the WfMS, that is responsible for providing the Workflow generator the appropriate information according to the step in the generation process.

For generating an abstract workflow the Workflow generator receives from the WfMS the workflow initial state and objective. In the context of architectural reconfiguration, the initial state would correspond to the actual architectural configuration of the system (configuration A), and the objective would correspond to the desired architectural configuration (configuration B). One of the ways of changing an architectural configuration A into an architectural configuration B is disconnecting all involving components of configuration A, reaching a state of non-configuration, and then connecting all components of configuration B. The first step demands tasks templates related to the disconnection of components, while the templates of the second step are related to the connection of components. In this context, we assume that the workflow generation process is applied for the connection of the involved components, where the initial state corresponds to the non-configuration state, and the workflow objective corresponds to the desired configuration B. In this scenario, the available task templates are related to the connection, blocking and unblocking of components, and the generated abstract workflow consists of the sequence of task templates that takes this state of non-configuration and finishes with the desired configuration B. This abstract workflow is then sent to the next activity in the generation process (generate concrete workflow) and to the WfMS, which will look for the appropriate resources to be used for generating the concrete workflow.

There are two cases that must be considered when generating a concrete workflow in the context of architectural reconfiguration. The first case concerns the disconnection of an existing configuration, where all the involved resources are known, and cannot be changed, while the second case, which is the case being focused, concerns the connection of an architectural configuration, where the components involved can be changed for different reasons. When generating a concrete workflow for connecting an architectural configuration, the WfMS is responsible for dealing with the situations where the involved components may change, providing the workflow generator the actual resources that will be used for mapping an abstract workflow into a concrete workflow.

The concrete workflow is then executed by the WfMS, effectively altering the system architecture. In case of a

Figure 1. Infrastructure for workflow generation.
failure during the execution of the concrete workflow, a feedback is provided to the workflow generator through the WfMS for generating a new concrete workflow, and if it is not possible to find a concrete workflow based on an abstract workflow, a feedback is provided through the WfMS for generating a new abstract workflow.

4. Architectural Reconfiguration Based on Dynamic Workflows

In the following, the dynamic generation of workflows is applied to the architectural reconfiguration of a software system. For that, we have defined a reconfiguration process based on a feedback control loop, and used dynamic workflows to control and coordinate this process. The workflows coordinating the reconfiguration process are generated according to the resources available. In other words, the system resources not only constraint its architecture, but also affect the workflows that manage the architectural reconfiguration.

In the following, we present dynamic workflows applied to architectural reconfiguration process, and provide details on how workflows are generated for coordinating architectural reconfiguration.

4.1. The Reconfiguration Process

Taking as basis the three-layer reference model introduced by Kramer and Magee [14], in this paper we are specifically concerned with the workflows that define the plans of the change management layer, even though workflows can be generated for managing the activities associated with each of the three layers.

Figure 2 presents an overview of the reconfiguration process. This process considers the connection of architectural configurations. We assume that if there is an existing configuration, this configuration is fully disconnected before the reconfiguration process starts, thus assuming a non-configuration state as initial state for the reconfiguration process. In order to control the architectural reconfiguration of the Target System, the WfMS requires as input the system requirements, architectural configurations, and the resources available. The Requirements enumerate the properties that specify the quality of services (QoS) expected from the system. The Abstract configurations is a repository that contains a set of abstract architectural configurations. An abstract configuration is an architectural configuration that identifies the structure of the system and the functional and non-functional requirements of the involved components, independently of the actual components instances. A concrete configuration, on the other hand, is an architectural configuration where all components instances and their respective attributes are identified. The Registry is where all available resources and their respective attributes are stored. In this approach, we assume that the Registry is responsible for monitoring the available resources of the system, and for providing an accurate view of the system to the WfMS. If there are changes in the resource availability during the reconfiguration process, for example, a failure of a component, the process will query the Registry again for obtaining an updated view of the available resources.

The defined reconfiguration process is also divided in
two different levels of concerns: strategies and tactics, corresponding to the partition that is key to Cheng et al approach [7]. At the strategy level, associated with the Select configuration activity, an abstract configuration is selected based on the system requirements and the available resources, while at the tactics level, associated with the Connect configuration activity, a concrete configuration is defined based on the selected abstract configuration and in the available instances. In this way, an abstract architectural configuration can be instantiated using different concrete configurations containing different combinations of components. At the strategy level, the Identify abstract configurations activity takes into account the requirements and available resources for identifying the set of abstract configurations that can be instantiated, and the Select abstract configuration activity is responsible for selecting one configuration from this identified set. The selected abstract configuration is used as input for the tactics level. At the tactics level, the Identify concrete configurations activity builds all the possible concrete configurations involving the available instances, while the Select concrete configuration is responsible for selecting one concrete configuration considering the available alternatives. The Connect concrete configuration activity is responsible for connecting the components instances according to the concrete configuration, effectively, altering the system. If there is a failure during the connection of resources, new instances need to be selected, and if there are no available instances that meet the specifications of the selected configuration, an alternative configuration needs to be selected.

Since faults can occur during architectural reconfiguration, it is fundamental for the process controlling the reconfiguration to be fault-tolerant, so the integrity of the system can be maintained. This is achieved by structuring the reconfiguration process according to strategies defined in [8] in which the activities are structured using atomic actions, thus allowing the handling of failures during the reconfiguration.

4.2. Workflows

In our approach, the two processes for controlling workflow generation and architectural reconfiguration are interconnected in a way that the workflows used for controlling architectural reconfiguration also control the process for generating workflows. The reconfiguration process is responsible for providing the necessary information to the workflow generator depending on the stage of the generation process. This corresponds to providing an abstract configuration for generating an abstract workflow, and a concrete configuration for generating a concrete workflow. When there is the need to generate a new workflow, the workflow that coordinates the selection of a configuration notifies the workflow generator. In this way, whenever there is, for example, a failure during the execution of the generated workflow, a new concrete workflow is generated using updated information from the reconfiguration process, and if there is not enough resources for generating a concrete workflow (i.e., it is not possible to identify a concrete configuration), a new abstract workflow is generated after the selection of a new abstract configuration. In the following, we present the workflows defined for controlling these two processes.

The workflows in our approach are modelled using the YAWL workflow modeling language [23]. YAWL is based on Petri nets, extended with additional features for modeling complex workflows. A workflow in YAWL is a set of nets that form a hierarchy in a tree-like structure. Each net has got tasks, and one input condition and one output condition. This notation can represent different coordination patterns, and supports atomic tasks or composite tasks. As a reference for the workflows to be presented in this paper, Figure 3 presents a sub-set of the symbols of the YAWL language.

In this work, we are focused on generating the workflow associated with the Connect concrete configuration activity, which is associated with the tactics level of the reconfiguration process, shown in Figure 2. We proceed now to present the different workflows associated with this level. Figure 4 depicts the workflow associated with the Connect configuration activity. It receives as input the selected abstract configuration and the abstract workflow, both defined at the strategy level. After that, the set of available instances is obtained (Obtain available instances), and used for identifying all the possible concrete configurations (Identify concrete configurations). In the sequence, a concrete configuration is selected (Select concrete configuration), and then used for generating the concrete workflow (Generate concrete workflow) that will be executed in the Connect concrete configuration task. If this task finishes with a failure, the workflow is restarted, considering the abstract configuration and the abstract workflow received. Once a new concrete configuration has been selected, a new concrete workflow is generated and executed. If it is not possible to find a concrete configuration for the
selected feasible configuration with the available instances, this workflow finishes with a failure, returning the process to the strategy level, where a new abstract configuration will be selected, and a new abstract workflow will be generated. If the configuration is established, this workflow commits.

Figure 4. Tactics level workflow.

Figure 5 presents the workflow associated with the Connect Concrete Configuration task. This workflow controls the execution of the generated concrete workflow. The generated concrete workflow is used as sub-workflow for the Connect instances tasks, where the system configuration is effectively altered. For now, all components involved in a concrete architectural configuration are disconnected, and the blocking of the involved components is part of the Connect Concrete Configuration workflow and not the generated concrete workflow. If there is a failure while connecting the components, all active connections are disconnected and the components unblocked, before workflow finishes with a failure. Otherwise the workflow commits.

Figure 5. Workflow associated with the Connect Concrete Configuration task.

For now, the workflow associated with the disconnection of the active connections is not generated using the defined model. This is done by using a simple undo scheme, in which the log of the workflow execution is used to identify the executed tasks, and the information about the tasks that undo the ones executed is part of the task library.

4.3. Workflow Generation for Architectural Reconfiguration

In our approach, we are restricting architectural reconfiguration to the connection/disconnection of components and connectors, not considering the change of component parameters as part of the reconfiguration process. Thus the tasks templates available are related to operations necessary for changing a configuration according to the underlying component model used (i.e. connect/disconnect, block/unblock, start/stop operations). To deal with components failures, the defined task templates are also structured according to the strategies based on atomic action identified in [8].

Figure 6 presents the sub-workflow defined as a task template for the connection of two components. This is an example of a task template with an exception handling path in it. After the verification of the pre-condition, the components can be connected. The connection is dependent on the communication technology used. If there is a failure during the connection of the components, the connection is aborted and this workflow finishes with a failure. In case there is a failure in the post-condition associated with this workflow, the components are disconnected, and it finishes with a failure. The workflow commits when the components are successfully connected and the post-condition met.

Figure 6. Task template for connecting two components.

The workflow initial state and objective are represented using, respectively, pre- and post-conditions. They are expressed in terms of the elements of an abstract architectural configuration, where the involved components are identified using logical names. These logical names are also used in pre- and post-conditions associated with the tasks templates of the generated abstract workflow. In this context, the pre-condition states that all components are disconnected, and the post-condition states that all components are connected.

The mapping of an abstract workflow into a concrete one uses the logical names to identify the component instances in the concrete configuration. At this point, the components instances are known, and their state are queried in order to identify the necessity of any additional tasks in the concrete workflow.
5. Case Study Application

For demonstrating the feasibility of using dynamic workflows for coordinating software adaptation, we have developed a web service application that relies on the dynamic reconfiguration of its architecture for the provision of dependable services. This application addresses the problem of obtaining stock quotes from the Web. The stock quotes are collected from different sources, and different fault-tolerance strategies applied in order to obtain correct stock quotes. These different strategies are captured by alternative architectural configurations. The coordination of the architectural reconfiguration is obtained through the usage of workflows that are dynamically modified during run-time depending on the availability of resources.

The software architecture of this case study consists of a FrontEnd component, two different components that represent the different fault-tolerance strategies (Voter and Comparator), and different sources of stock quotes (for example, BridgeYahoo and BridgeGoogle - a bridge handles architectural mismatches between the online sources and our system, and provides a well defined interface for accessing different online sources of stock quotes). The Voter component requires three different sources of stock quotes, and performs majority voting on the values received, while the Comparator component applies comparison between two sources of stock quotes for detecting inconsistencies.

For implementing this application, we have used the infrastructure presented in Figure 1. All architectural elements are implemented using Java language, and the communication among them uses web services technology. For our approach, we have developed an extension of xADL 2.0 that allows the architectural elements and the workflow engine to communicate through web services. Our extension to xADL 2.0 adds some tags to represent the information used by the WIMS, such as, the WSDL address, and the attributes of the resources. Our extension of xADL allows also to represent architectural configurations, resources, and system requirements. For now the requirements are expressed in terms of the attributes and the respective values that must be met by the system.

For evaluating the current approach, we have conducted some experiments using the Workflow generator for building the workflow during run-time according to the selected architectural configuration. In these experiments, faults were injected in the system for crashing different components at different stages of the reconfiguration process in order to investigate its robustness in the presence of failures. The architectural configuration was successfully modified during run-time based on the system requirements, the alternative configuration, and the attributes of the available resources. Based on the selected configuration, the workflows are successfully generated for changing the system.

Figure 7 presents an example of an abstract and concrete architectural configurations. The abstract configuration identifies the functional and non-functional requirements associated with each component using its logical name. For example, C1:FrontEnd indicates that the component with logical name C1 has got functional requirements associated with components of type FrontEnd.

In the concrete configuration we have the instances associated with each component identified, and their respective attributes. C1:FrontEnd:Gui indicates that the component with logical name C1 of type FrontEnd has got the component instance identified by Gui associated with it.

Figure 8 presents an abstract workflow generated based on the abstract configuration of Figure 7. This workflow identifies the tasks templates to be used, and uses the logical names of the components to populate the templates.

A concrete workflow that connects the instances is dynamically defined by populating the abstract workflow with the components instances that are part of the concrete configuration. Figure 9 presents an example of a concrete workflow for the presented configuration.

In this case study, faults were injected during the process for evaluating the fault-tolerance of the proposed approach. Although the workflows are successfully generated during run-time based on the available resources, and the system is successfully configured when there are enough resources, experiments in more complex scenarios must be
realised for a better evaluation of the presented approach. These experiments may include a different number of task templates and different components states, reconfiguration of systems where the selected components are already connected, or the replacement of one component in an existent configuration. Moreover, this example has also shown how the proposed approach support the handling of faults during the reconfiguration.

6. Related Work

Existential approaches for self-adaptation [7] [10] have focused on the selection of an adaptation plan using rule-based mechanisms, where each available plan is defined at design-time for each condition that triggers an adaptation. In a such approach all possible actions and associated activation rules are kept in a repository. Sykes et al. [21] present an approach for the generation of reactive plans of autonomous systems from high level goals. A reactive plan is a set of condition-action rules that specify the behaviour of the system, where actions are mapped into component interfaces that define the functional requirements of the system. Different from these approaches, our approach aims to build automatically an adaptation plan during run-time, thus providing the means for dealing with uncertainty associated with the self-adaptation process.

Our approach is not the first to employ workflows management technology for coordinating software (self-) adaptation [20] [22]. However, these approaches either restrict the support to software adaptation [20], or deal with the coordination of mobile agents responsible for executing the adaptation [22]. Moreover, both approaches suffer from the same limitations of having to define adaptation plans during design-time. In our approach, we focus on building and modifying workflows during run-time, providing the means for handling failures during the execution of an adaptation plan, and do not restrict adaptation since workflows are dynamically defined according to the available resources.

Autili et al. [5] present an approach for context-aware self-adaptation. In their approach, applications are adapted at deployment time for the target environment. Their focus is on the selection of the best adaptation alternative based on the current environment state and the required resources of the application. In our approach, we focus on the dynamic modification of the adaptation process, assuming the existence of mechanisms responsible for selecting an architectural configuration for the system.

Concerning workflow generation, Shankar and Campbell [19] present a policy-based approach using event-condition-action (ECA) rules, where a workflow is built based on pre- and post-condition of different rules actions that are triggered by a single event. This approach requires the definition of rules in a manner that the right actions will be activated by a single event. Moreover, this approach does not deal with issues like the variability in the resources availability. Another approach for automatic workflow generation is presented by Lu et al [16]. They have developed a formal model in which workflows are assumed to be constructed from a library of parameterised tasks with pre- and post-conditions, and algorithms that synthesise a workflow based on the pre- and post-conditions of workflows. Their approach considers workflow generation as search problem in a search space graph, and assumes a particular task model, where tasks can have alternative outputs with different post-conditions. Our approach uses ideas from [19] and [16], and is very similar to the ideas presented by Lee et al. [15], in which an abstract workflow is mapped into a concrete one and then executed in a computing grid. However, in their approach they are focused on optimising the mapping between the abstract and the concrete workflows. In our approach, we are not restricted to a pre-defined abstract workflow, as the abstract workflow can also change.

Several approaches apply AI planning techniques for generating workflows in different domains, such as, grid computing [9], web service composition [2], and pervasive computing [18]. Arshad et al [4] have used planning technology for reconfiguration planning. However, their approach can only deal with a fixed initial set of components, and the generated plan is not executed, as the generation infrastructure is isolated, with the execution of the generated plan indicated as future work. In our approach, we are not restricted to a fixed set of resources for generating adaptation plans, and we are able to modify the adaptation plan to deal with failures during its execution.

7. Conclusions

This paper has proposed an approach that employs dynamic workflows for controlling and coordinating the self-adaptation of software systems. The main focus of this approach is towards mechanisms for automatic generation of workflows. These generated workflows have been applied in the context of architectural reconfiguration of software systems, and in order to handle faults during the adaptation, atomic actions were used as a means for structuring the workflows.
The feasibility of the proposed approach was demonstrated in the context of a web service application for obtaining dependable stock quotes. In this application, the workflows for coordinating the reconfiguration of the system changed depending on the availability of resources. For evaluating the fault-tolerance of the approach, faults were injected in the system components at different stages of the reconfiguration process, and the system has been successfully configured when there are enough resources for establishing a configuration. Although the preliminary results are encouraging, more detailed studies, involving more complex scenarios and other kinds of applications, have to be performed for a better evaluation of the proposed approach.

Currently, we are investigating existing techniques for workflow generation in order to optimise the generation process, and for that we intend to make better use of techniques based on workflow meta-models. Our long term plan is the application of the mechanisms for workflow generation in order to optimise the generation performed for a better evaluation of the proposed approach.

References


