Presented at The Fifth International Conference on Autonomic and Autonomous Systems (ICAS'09)

Resource-Aware Contracts for Addressing Feature Interaction in Dynamic Adaptive Systems

Yu Liu and René Meier
Lero @ TCD
Distributed Systems Group
Department of Computer Science
Trinity College Dublin
{yuliu, rmeier}@cs.tcd.ie

Abstract

Dynamic adaptive systems are becoming increasingly popular due to their ability to adapt to heterogeneous and changing environments. Such systems must avert adverse feature interaction where the adaptation of an existing feature or the introduction of a novel feature may result in unexpected and possibly adverse system behavior. This paper proposes resource-aware contracts for addressing adverse feature interaction in dynamically adaptable systems resulting from resource constraints. Resource-aware contracts explicitly capture the resource requirements of the individual components comprising a system. They are considered a fundamental means towards detecting and ultimately resolving adverse feature interaction and a key enabler of dynamic system adaptation.

1. Introduction

Dynamic adaptive systems are becoming increasingly popular with a number of recent trends contributing to this popularity, including, expanding industrial use of wireless technologies, reduced human intervention, greater autonomy of software system, and componentization of software system. Dynamic adaptation enables systems to handle variations to their operational conditions, for example, as a result of changes to the system context, due to fluctuations to the available communication resources, or as a result of new user requirements.

Systems may adapt to such conditions by updating existing system components or by introducing novel components to the system. Such updated or new software components are typically thoroughly tested prior to their introduction to a system. However, due to the ever-increasing scale and complexity of today's systems, where potentially vast sets of components, parameters, and interactions are modeled, adaptation may cause two (possibly independent) components to disrupt each other's behavior. This is referred to as adverse feature interaction. Adverse feature interaction describes a situation where the combination of two or more components, or features, each of which individually performs correctly results in adverse behavior. Adverse Feature Interaction was first identified in telecommunication systems [4] where applications provided by independent third-party developers are expected to collaborate to implement a call protocol. Interaction between software components have been mainly dealt with at structural and behavioral level [1]. Structural level [5] interaction is concerned with the violation of architectural rules when updating existing components or adding new components. Behavioral level [5] interaction is concerned with the violation of the protocol specification of a system in terms of pre-condition and post-condition invariants. However, these approaches fall short in addressing adverse feature interaction arising from resource constraints. Resources-based adverse feature interaction occurs when a new component is added and the system cannot cater for its resources requirements, for instance, the available network bandwidth is insufficient. Moreover, other

components may not be able to adapt to the new component's resource usage pattern, for example, where a component requires exclusive control over a certain resource while other components are unable to relinquish their control of same resource.

This paper proposes resource-aware contracts for addressing adverse feature interaction in dynamic adaptive systems resulting from resource constraints. Such resources can be categorized as Exclusive Resources, Fixed-Capacity Resources, Varying-Capacity Resources and Shared Resources. Exclusive resources, such as a single-core CPU, can only be used by a single component at any given time and are shared between components in a sequential manner. Fixed-Capacity Resources, such as memory and thread pools, can be used by multiple components in parallel, as long as total use does not exceed resource capacity. Network bandwidth and battery power may change over time, and thus, are termed Varying-Capacity Resources. Shared Resources, such as actuators, can receive requests from two or more components, possibly at different times. Without coordination, a shared resource might receive conflicting requests and multiple shared resources may attempt to affect the environment in a contradictory manner [6]. The usage patterns of such resources are typically implicit to component implementations.

The concept of contracts [2] can be used to explicitly capture the resource requirements of the individual components in a component-based system. Resource-aware contracts model the semantics of components in terms of their resource consumption and separate the computation and coordination constraints of components. Dynamic adaptation necessitates explicit coordination between components, as system behavior can be changed through the addition of new components, the replacement of existing components or the reconnection of existing components. While contracts have been used for resolving behavioral component conflicts [3], the novel resource-aware contracts described in this paper are essential to explicitly describe the requirements of component's resource needs and ultimately, to address adverse feature interaction that may result from such resource usage.

The remainder of this paper is structured as follows: related work is described in Section 2; Section 3 discusses adaptation scenarios where resource based adverse feature interaction arises; Section 4 describes our resource-aware contracts; Section 5 describes an initial approach for using resource-aware contracts in dynamic adaptive systems; Section 6 presents conclusions and future work.

2. Related Work

Different approaches have been proposed to address adverse feature interaction. These approaches can be grouped into two classes [4], namely, *off-line approaches* and *on-line approaches*, according to the stage of the software lifecycle to which the approach is applied.

Off-line approaches rely mostly on formal models describing features [10]. Many formal notations have been proposed, including LOTOs, CSP, Promela [4]. The underlying assumption of this approach is that formal models of features are obtainable and conflicts can be detected at the design time and in an automated fashion. Off-line approaches focus on dealing with behavioral feature conflicts and do not consider resource conflicts.

On-line approaches [7] depend on observable behavior of features at runtime to analyze and reason about potential adverse interactions. Observable behavior of features can be in the form of either exchanged messages among features or negotiation proposals among communicating agents. Many on-line approaches do not consider resource as a potential source of adverse feature interaction. [8] is the first work that shifted the focus of research from behavioral conflict to resource conflict; it uses feature-resource relationships to define resource consumption of individual features, and describes the resource constraints of a set of composed features through goals. However, the resource specification proposed only deals with Fixed-Capacity Resources and Varying-Capacity Resources, and does not describe the usage pattern of Exclusive Resources and Shared Resources. Moreover, the resource specification does not address resource requirements at component assembly level. In contrast, this paper adopts resource-aware contracts to explicitly define the resource needs of individual components and of component assemblies as well as of the resource constraints of the system.

3. Adaptation Scenarios

Dynamic adaptation not only changes the configuration of a component based system, by adding or replacing components, but also has an impact on the resources consumed. Dynamic adaptive systems employ various adaptation strategies, which are fundamentally driven by the resource constraints of a system. A component needs a set of required resources to perform its task. As a result of the execution of the task, the properties of not only the required resources

but also of other resources can be changed. Different components can share resources; however, un-coordinated access can be problematic at times. Resource-based feature interaction can be attributed either to conflicting usage patterns upon the same resource or to compromised resource constraints. The following scenarios illustrate how resource-based feature interaction emerges in the context of dynamic adaptation.

Adaptation to Limited Resources: Consider an in-vehicle entertainment system that allows backseat passengers to play video games and browse the Internet. A backseat passenger might be invited to join a multiplayer network game, possibly by a passenger in a car traveling in the same direction. As a result, a new video game component might be downloaded via roadside infrastructure and activated with the consent of the passenger. Such an adaptation is likely to impact another passenger, who happens to browse the Internet. These two application components compete for network bandwidth and there is possibility that one application, for example, the video game, consumes most of the bandwidth and thus interrupts Internet browsing. One application is designed to adapt to changes to the resources, in this case the available bandwidth, without considering the consequence of such adaptation upon other applications.

Adaptation to Conflicting Resource Constraints: Consider a component that adapts to low battery level by suspending activities that use network bandwidth [9]. As a result, more bandwidth becomes available, which in turn might trigger bandwidth adaptation that avails of unused bandwidth by activating activities that will use bandwidth. This bandwidth adaptation conflicts with such a battery management policy as the additional activities are likely to increase battery power consumption.

4. Defining Resource-Aware Contracts

Resource-aware contracts express the resource consumption of components without the need to refer to their implementation. Resource-aware contracts are considered an abstract part of components that can be automatically processed during adaptation time. Resource-aware contracts are used to describe the resource usage patterns of components and component assemblies, as well as inherent constraints of system resources. Adverse feature interaction arises if resource usage patterns of different components comprise each other's goal, or resource constraints are violated. To define resource-aware contracts in a dynamic adaptive system, the following concepts are used.

Resource [8] is required by software component in order to execute its task. The execution of a task can change certain properties of a resource. Each resource has a set of constraints defining correct use of resource.

Component [11] is a basic unit in component-based system and typically consists of required and provided interfaces. There are one or more operations in an interface. Each operation is seen as a single task.

Component Assembly is a way of structuring an application from a set of independently developed components.

System is a collection of component assemblies and resources.

Two essential concepts, namely resource-aware routine assertion and resource invariant, are used in resource-aware contracts. Resource-aware routine assertion comprises pre and post conditions associated with each operation. An assertion defines the semantics of the execution of either a single operation or a group of operations. Resource invariant can be thought of as common law regarding resource use and consists of a set of rules prescribing correct use of a resource.

```
Component = {
                componentName,
                (RoutineName, RoutineAssertion) *
}
Connector = {
                 (Component.RoutineName,
                 Component.RoutineName)
}
RoutineAssertion = {
                 RequiredResources = (Resource)*
                 Pre-Condition = P(RequiredResources)
                 AffectedResources =  (Resource)*
                 Post-Condition = Q(AffectedResources)
}
```

The resource needs of a single component can be defined through a set of routine assertions. As is seen from the above definition, *RequiredResources* defines the set of resources needed for an operation. *Pre-Condition* is a boolean function over the required set of resources, defining entry conditions for this operation. Typically, entry conditions can be the desired quantity or state of each required resources. *AffectedResources* identifies the set of resources being influenced by the execution of an operation, and *AffectedResources* does not necessarily overlap with *RequiredResources*. *Post-Condition* is also a boolean function over *AffectedResources*, defining the resource impact of the operation execution, for instance, the amount of

bandwidth consumed. Only when a *Pre-Condition* is satisfied an operation is allowed to be executed, the implication of this is that valid resources are available. *Post-Condition* should always hold immediately after the execution of an operation.

comp1 = {VGameComp, (activate, assertion1), (execute, assertion2)}
comp2 = {WBrowerComp, (activate, assertion3), (execute, assertion4)}

assertion1 = {(memory, bandwidth), (memory > 100M && bandwidth >= 5 M/s), (memory), (allocated(memory) == 25 M)}
assertion2 = {(memory, bandwidth), (memory > 50M && bandwidth >= 5M/s), (memory, bandwidth), (allocated(memory) == 25M && allocated(bandwidth) == 5M/s)}
assertion3 = {(memory, bandwidth), (memory > 50M && bandwidth >= 2.5 M/s), (memory), (allocated(memory) == 10 M)}
assertion4 = {(memory, bandwidth), (memory > 50M && bandwidth >= 2.5M/s), (memory, bandwidth), (allocated(memory) == 15M && allocated(bandwidth) == 2.5M/s)}

For example, consider possible resource-contracts for the two components, for video gaming and for Internet browsing, of the proposed in-vehicle entertainment system scenario. From the routine assertions used for this scenario, we can derive the bandwidth each operation is expected to consume. If both components are executed concurrently, the total bandwidth available must exceed 7.5M/s. The proposed adaptation of the video game component should be cancelled (or postponed) if these bandwidth requirements cannot be met.

```
ComponentAssembly = {
        LocalComponents = (Component)*
        LocalConnectors = {Connector)*
        LocalResources = (Resource)*
        AdaptationStrategies = (AdaptationStrategy)*
}
AdaptationStrategy = {
        AdaptationTrigger = T (LocalResources)
        AdaptationPre-Condition =
    P(LocalResourcs, LocalComponents, LocalConnectors)
        AdaptationPost-Condition =
    Q(LocalResources, LocalComponents, LocalConnectors)
}
```

The resource contract for a Component Assembly includes contracts for participating components and for overall resource consumption. A component assembly essentially defines an adaptive application. The adaptive application is structured from a set of components, *LocalComponents.* The way components are interconnected is defined by *LocalConnectors*, and each connector connects two operations in different local components. A set of resources, *LocalResources*, is assigned to the adaptive application and accessible by local components. Adaptation strategies are used to constrain adaptation. An adaptation is triggered by changes taking place at local resources, as indicated by a boolean function defined over *LocalResources*. Adaptation typically has an impact on components, connectors and resources. The impact of adaptation is constrained by *AdaptationPre-Condition* and *AdaptationPost-Condition*. *AdaptationPost-Condition* can be seen as the goal of adaptation.

An important category of adverse feature interaction is due to conflicting goals of adaptation strategies manifested through conflicts at resource level; therefore, an explicit description of the adaptation goal supports the detection of adverse feature interaction at a subsequent stage. For example, consider the resource management where two conflicting adaptation strategies are employed: battery power adaptation and bandwidth adaptation. Battery power adaptation is triggered initially. The goal of battery power adaptation is to keep power consumption under a certain threshold. As more bandwidth becomes available, conditions in *AdaptationTrigger* of bandwidth adaptation will eventually hold. The intention of bandwidth adaptation is to avail of unused bandwidth, and as a side effect, more power will be consumed. Both the intention and the side effect are described in the *AdaptationPost-Condition* of the adaptation strategy. In this case, detection of conflicting adaptation strategies is conducted through examination of the conflicts in the *AdaptationPost-Condition*.

```
Resource = {
        ResourceID
        ResourceType =
  Enum {Exclusive, Fixed-Capacity, Varying-Capacity, Shared}
        ResourceInvariants(ResourceType)
}
ResourceInvariants(ResourceType) = {
        If ResourceType == Exclusive:
                AtMostOneClient
        If ResourceType == Fixed-Capacity:
                NeverExceedingFixedCapacity
        If ResourceType == Varying-Capacity:
                NeverExceedingCurrentCapacity
        If ResourceType == Shared:
                NeverAcceptConflictingValues
}
```

System wide resource invariants are rules associated with resources that must always be

enforced. The rational of having resource invariants is to provide a system view of a resource profile which otherwise would be implicit. Resource can be accessed from a component and a component assembly. Resource invariants apply wherever a resource is accessed. As identified earlier, four types of resources are available, each of which has unique invariants. Abstract functions can be used to denote resource invariant. Due to space limitation, only a brief description of each of these abstract functions is provided here.

For exclusive resources, *AtMostOneClient* means this type of resources can only be locked and used by one client exclusively. Typically, some scheduling mechanism is used to serialize the use of exclusive resources. For Fixed-Capacity Resources, *NeverExceedingFixedCapacity* means the capacity of this type of resources is fixed and it must be guaranteed that the allocation of resource to different components will never exceed its capacity. For Varying-Capacity Resources, *NeverExceedingCurrentCapacity* means the capacity of resources is determined from the current operational context, and total use of this resource should not surpass current capacity. For Shared Resources, concurrent control requests can be in the form of either boolean values or numerical values [6]. *NeverAcceptConflictingValues* means that conflicting requests to change the shared resource should not be allowed.

5. Enforcing Resource-Aware Contracts

Resource-aware contracts can be used at adaptation time to ensure safe transition from the existing component-based configuration to a new component-based system configuration. Adaptation, in response to an adaptation request, is expressed in terms of adaptation strategies. An adaptation strategy specifies the new configuration as well as the resource goal of the adaptation. The resource goal of an adaptation can then be verified against the resource requirements of the components of the new configuration as expressed in their respective resource contracts. An adaptation request is considered invalid if the adaptation would result in a violation of the "resource invariants" that describe the constraints of the resources. Such adaptation requests can then be denied and adaptation to invalid compositions can be prevented. An adaptation based on an alternative adaptation strategy with a similar adaptation goal might be considered instead.

6. Conclusion and Future Work

This paper describes our initial approach to using resource-aware contracts for addressing adverse feature interaction in dynamic adaptive system. We explicitly capture the resource consumption of features in the form of resource contracts, and apply these contracts to different aspects in dynamic adaptive systems. From the resource point of view, adverse feature interaction can happen at either the same resource or different but conflicting resources. From adaptation point of view, adverse feature interaction takes place when either a new component is integrated into an assembly or multiple adaptation strategies are applied. Our future work will refine and further evaluate what constitutes a pair of conflicting resources and conflicting resource usage patterns, and extend contracts accordingly. We then intend to develop an approach for enforcing resource-aware contracts.

Acknowledgments. The work described in this paper was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] A. Nhlabatsi, R. Laney, and B. Nuseibeh, "Feature Interaction: the security threat from within software systems", *Progress in Informatics*, No.5, pp.75-89, 2008

[2] B.Meyer, "Appying Design by Contract", *IEEE Computer*, pp.40-51, Oct.1992

[3] L.F. Andrade and J.L. Fiadeiro, "Feature modeling and composition with coordination contracts", *Proceedings Feature Interaction in Composed System (ECOOP 2001)*, pp.49-54, 2001

[4] M. Calder, M. Kolberg, E.H. Magill and S.R. Marganiec, "Feature Interaction: A Critical Review and Considered Forecast", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, v.41 n.1, p.115-141, 15 January 2003

[5] A.Leicher, "Analysis of Compositional Conflicts in Component-Based Systems", *PhD Thesis*, Technischen Universitat Berlin, 2005

[6] A.L.Juarez-Dominguez, N.A.Day and J.J.Joyce, "Modelling Feature Interaction in the Automotive Domain", *Proceedings of the 2008 international workshop on Models in software engineering*, pp.45-50, 2008

[7] S.R. Marganiec, "Runtime Resolution of Feature Interactions in Evolving Telecommunication Systems", *PhD Thesis*, Department of Computer Science, University of Glasgow, 2002

[8] J. Bisbal and B.H. Cheng, "Resource-based Approach to Feature Interaction in Adaptive Software", *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pp.23-27, 2004

[9] L. Blair, G. Blair, J. Pang and C. Efstratiou, "Feature Interaction Outside a Telecom Domain", *Proceedings of Workshop on Feature Interactions in Composed Systems (ECOOP'2001)*, 2001

[10] A.P. Felty and K.S. Namjoshi, "Feature specification and automated conflict detection", *ACM Transactions on Software Engineering and Methodology*, v.12 n.1, p.3-27, January 2003

[11] Kung-Kiu Lau and Zheng Wang, "Software Component Models", *IEEE Transactions on Software Engineering*, 33(10): 709-724, 2007