Non-Functional Property Specifications for WRIGHT ADL

Basem Suleiman\(^1\), Vladimir Tosic\(^2\,\(^1\), Eldar Aliev\(^3\)

\(^1\) School of Computer Science and Engineering, University of New South Wales, Australia
\(^2\) Managing Complexity – Sydney ATP Group, NICTA*, Australia
\(^3\) Shared Services, SSH Australia, Australia
Basem.Suleiman@nicta.com.au

Abstract

Many architecture description languages (ADLs) were proposed for describing structure of software systems in terms of components and connectors. However, specification of non-functional properties is not addressed well in these ADLs, although it is crucial for building quality software architectures. We propose new extensions of WRIGHT ADL with capabilities to specify non-functional properties at the architectural level. They are based on ontological definitions of non-functional properties. To demonstrate their usefulness, we applied them to an e-health example.

1. Introduction

An architecture description language (ADL) \[1\] is a formal specification language for defining and analysing software architectures. It provides a set of precise notations for explicitly describing functionality and architectural structure of software components and their interactions \[2\]. This description is beneficial for early design decisions and their analysis, as well as for generating executable code (e.g., prototypes). Many ADLs have been proposed, including Darwin, Koala, UniCon, Rapid, WRIGHT, C2 and ACME \[3, 4\]. Among them, WRIGHT is a powerful and promising ADL that is widely adopted by the software engineering community. It allows description of not only architectural styles (i.e., types of software systems), but also architectural instances (i.e., individual software systems). However, most ADLs, including WRIGHT, do not describe non-functional properties, such as performance or security. We present WRIGHT+NF, a WRIGHT extension with optional specification of non-functional properties.

While functional properties determine what a system is supposed to do, non-functional properties specify conditions under which a functionally correct operation is appropriate. For example, a functionally correctly bank system that allows unauthorized users to access any information is not appropriate. While achieving desired functionality is always essential, attaining desired non-functional properties is often also a crucial aspect of software success. Descriptions of non-functional properties in software architecture can be used for design analysis and refinement decisions, as well as automatic generation of code and test cases. Unfortunately, specifying non-functional properties at any level of abstraction and at any time in the lifecycle is difficult for many reasons \[5\]. Further, specification of non-functional properties that cannot be checked during design-time and/or runtime is not very useful, but high precision is necessary for such checks. For example, it is not enough to say only “response time of operation X should be less than 1 second”. There are at least 2 definitions of the non-functional property “response time” \[6\]: (1) “time from the end of request submission till the beginning of response” and (2) “time from the end of request submission till the end of response” Both are correct definitions, but if 2 interacting components assume different definitions, this will lead to semantic mismatches and reduction of interoperability. Furthermore, the example is not clear on where response time is measured – there can be a huge difference between its measurement at client side and its measurement at server side, due to network delays.

Our proposed WRIGHT+NF constructs are based on ontological definitions of non-functional properties, specification of additional information necessary for runtime monitoring, and the Communicating Sequential Processes (CSP) notations used in WRIGHT. The use of ontological definitions increases precision of defining system properties and reduces chances for semantic mismatches. This is a crucial feature, especially in heterogeneous distributed systems. The specification of additional run-time monitoring information (e.g., when,

---

\* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.
how, and by which party is monitoring done) gives precision necessary for semi-automatic refinement of architectural descriptions into deployable implementations. The adoption of CSP notations (e.g., logical operations) enhances consistency of the new constructs with WRIGHT.

Section 2 provides a quick review of related work. Section 3 describes the basic constructs of WRIGHT ADL and introduces a realistic example that illustrates WRIGHT constructs and its shortcomings in specifying non-functional properties. The proposed WRIGHT+NF notations are explained in Section 4 and their capabilities of specifying non-functionality are illustrated in Section 5, using the previous example. A discussion of the benefits and limitations of the proposed constructs and other conclusions are presented in Section 6.

2. Related work

Due to the importance of non-functional properties, there were a huge body of work on their specification at different levels of abstraction and different stages of the software lifecycle. To specify non-functional properties at the programming language level, many code annotations [7] and variations of aspect-oriented programming [8] were proposed. Further, many languages for specification of non-functional properties, particularly performance, of distributed objects [9] and Web services [5] were developed and used. While the best among these works provide useful specifications of non-functional properties, they are intended for late, not early phases in software lifecycle, and do no support architecture analysis and refinement. Thus, only some principles (and partial solutions) are reusable at the architectural level.

Specifications of non-functional properties in early software lifecycle stages are still not widely used. Medvidovic and Taylor [4] surveyed and classified various ADLs and their properties, structures, specifications and support to particular modeling or programming languages. They gave deep insights on the ADLs to enable software community to identify key features of each ADL and provide a framework for selecting and extending a particular ADL based on software requirements. Eenoo et al. [10] have extended WRIGHT to capture non-functionality. They mainly supported new constructs derived from CSP to express non-functional properties in terms of ‘required’ and ‘ensured’ features. Although the proposed notations are capable of expressing some non-functionality at the architectural level, they use a string-like representation that is not precise enough and leads to some redundancies. The ‘NonFun’ notations [11] present an approach for addressing non-functional requirements of software systems in software architectures. The approach introduces various notations for describing non-functional requirements and behavior of software components. Although ‘NonFun’ allows definition of a wide range of non-functional requirements, its concepts are not precise enough and are not smoothly integrated with ADL concepts, which reduces its usability. In another study [12], ‘Con Moto’ is the ADL proposed to allow early management of particular non-functional (security and availability) requirements on the architectural level of mobile software applications. Apart from precision, a limitation of this work is that it is intended only for mobile distributed systems. To conclude, the past ADL extensions are not precise enough for run-time checks. Precise answers to questions about which non-functional properties are measured and when, where, how, and by which entity these measurements are done are possible during early software lifecycle. Unfortunately, ADL authors focus only on design-time issues for which simplistic notations without such precision might seem to some authors as adequate. Works on specifying performance properties on software designs [13] and applying aspect-oriented programming principles in requirements engineering and architecture design [8] are examples of other recent achievements in the area of specification of non-functional properties in the early lifecycle stages, but they also lack precision.

3. Overview of WRIGHT and an example

In this section, we summarize fundamentals of WRIGHT [1] and show how WRIGHT notations are limited to only functional descriptions. WRIGHT syntax is mainly based on the notations of Communicating Sequential Processes (CSP) [14] that enable a concise description of events, processes and interactions among them. WRIGHT consists of three major software building blocks: components, connectors and configurations. A component is an independent entity that performs particular functionality based on input data from its environment and it produces output results to other components in its environment. It has two essential specifications: the interface and the computation. An interface consists of one or more ports through which a component can interact with its environment. Ports are described in terms of events that represent actions of behavioural specification. The computation part describes complete component’s behavior and components’ expectations about its environment [1]. A connector realizes the interaction between a set of software components. It can be a method invocation, shared variable, remote procedure call (RPC), network connection, database protocol, or other mechanism. WRIGHT connector description consists of two basic elements: a set of roles and glue. Each role specifies interaction behavior of participant
connector never allows simultaneous read and write of patient test data from the shared variable. This patient, a specialist’s software component (SP) reads the accepting them. To prepare the diagnosis report on a the identity of provided patient medical data before authenticating itself and by writing the data to a shared (GP) passes patient medical data to the environment by Figure 1. A general practitioner’s software component similar to the one from [10]. It is an e-health proposed extensions, we introduce an example attachments part, by attaching a particular component’s topology of the configuration (i.e. which component components. Similarly to component’s computation, the glue provides the complete behavioral specification of a connector. It also describes how participant components work together to achieve an interaction [1]. A configuration is a set of components’ and connectors’ instances attached together to provide functionality. The topology of the configuration (i.e. which component participates in which interaction) is described in the attachments part, by attaching a particular component’s port with appropriate connector’s role [1].

For further explanation of WRIGHT and our proposed extensions, we introduce an example application, similar to the one from [10]. It is an e-health system in which confidential clinical information is passed between systems’ stakeholders, including general practitioners, specialists, patients, and pharmacies. The system has several components and connectors, shown in Figure 1. A general practitioner’s software component (GP) passes patient medical data to the environment by authenticating itself and by writing the data to a shared variable connector. The shared variable connector checks the identity of provided patient medical data before accepting them. To prepare the diagnosis report on a patient, a specialist’s software component (SP) reads the patient test data from the shared variable connector. This connector never allows simultaneous read and write of test data from/to the shared variable. The same component then should provide a diagnosis report to a pipe connector that must have at least 50KB buffer space and 1 MB/second reading speed. A pharmacy component (PH) then reads the provided diagnosis report from its pipe environment and writes prescriptions information to its output port. The prescription information has to be encrypted before being provided to a remote procedure call (RPC) connector. To successfully end the process of patient information processing, the GP component requires receiving from its RPC environment all prescriptions information encrypted with its public key.

Behavioral specification of the system in WRIGHT is outlined in Figure 2. It is available in full at [15]. All system functional specifications can be stated well in WRIGHT. However, because of WRIGHT shortcomings in expressing systems’ non-functional properties, there is no possibility to describe authentication and encryption issues, buffer size, read and write speed. This could cause choosing functionally correct architectural solutions that are not appropriate in practice (because they do not satisfy non-functional properties), potentially leading to a huge amount of effort to correct these problems in later stages of the software lifecycle. To avoid such situations, non-functional properties should be explicitly stated in software architecture.

4. WRIGHT+NF

WRIGHT+NF is our extension of WRIGHT with specification of non-functional properties at the architecture level. Detailed discussion of syntax and semantics of WRIGHT+NF constructs is available at [15]. Due to limited space, we will focus this paper on explaining its 3 main elements, depicted in Figure 3. They are specified within a WRIGHT configuration, after all standard WRIGHT constructs. Note that the underlined parentheses ↓ and ↓ indicate that a particular construct is optional (if followed by underlined indices ↓ or can occur multiple times (if followed by underlined indices ↓). These parentheses and indices are not parts of the WRIGHT+NF syntax.

The first element is NF-ATTRIBUTES. We use the term ‘measured attribute’ to indicate an aspect of a non-functional system property (such as performance) that can be related to a particular value (and measurement unit, if applicable). Examples are response time or read speed. A measured attribute can be used in arithmetic or logic expressions. We require that measured attributes and measurement units are precisely defined in external ontologies [6]. This is essential to avoid semantic mismatches that could occur when systems’ components/connectors interact. An ontology formally defines a
The NF-ATTRIBUTES part. Measured attributes are expressions for systems' measured attributes introduced in SPECIFICATION. It allows specification of a set of expressions for systems' measured attributes introduced in the NF-ATTRIBUTES part. Measured attributes are grouped under generic property name that they belong to. For example, Authentication and Encryption are measured attributes of the Security property and BufferSize and ReadSpeed are attributes of the Performance property. NF-SPECIFICATION is a named element – PropertyName should be replaced with a non-functional property under which the specification of a number of measured attributes can be grouped and evaluated. Examples are security and performance. There can be 1 or many parameters (local variables) for an NF-SPECIFICATION. They abstract architectural elements (e.g., ports, roles) to which the NF-SPECIFICATION is applied (in the NF-FILTERS construct). The parameters are used within expressions. In this way, the same NF-SPECIFICATION can be applied to different instances of architectural elements. Each parameter is in the format variableName:TypeName, where variableName is the local name of the parameter and TypeName is the name of its WRIGHT ADL type (e.g., port-role). Multiple expressions of measured attributes of a particular property can be given inside one NF-SPECIFICATION. Our extensions allow specification of different arithmetic and logic expressions expressed in CSP notations. In order for an NF-SPECIFICATION to be satisfied (e.g., at runtime), all contained expressions must evaluate to ‘true’. Each non-functional property needs to be specified in a separate NF-SPECIFICATION construct, so there can be 1 or many such constructs.

The third element is NF-FILTERS. It is used to constrain operation of instances of architectural elements according to systems’ qualities given in NF-SPECIFICATION constructs. There can be only 1 NF-FILTERS construct within a WRIGHT Configuration element. It lists property names (each of which must be previously defined in an NF-SPECIFICATION construct) applied to concrete instances (1 or more parameters, corresponding in number and type to the NF-SPECIFICATION). It is possible to apply the whole property or only expressions related to a particular measured attribute. For example, the construct Performance.ResponseTime[P] means that only expressions of the Performance property related to the measured attribute ResponseTime are applied to the instance (e.g., port) P. On the other hand, the construct Performance[C] means that all expressions from the Performance property are applied to the instance C.

Figure 3. The new WRIGHT+NF constructs for specification of non-functional properties

Figure 3 shows the new WRIGHT+NF constructs for specification of non-functional properties.

A WRIGHT+NF example

In this section, we use WRIGHT+NF constructs to describe non-functional properties of the e-health system that are not captured in the WRIGHT configuration given in Figure 2. Figure 4 lists the new e-health configu-
In this example, there are 3 NF-SPECIFICATION constructs: Security, Performance, and Concurrency. Security has 1 parameter (PR) that can be either a port or a role. It is used in expressions and will be instantiated (3 times) in the NF-FILTERS construct. For example, the expression “$PR.Encryption.Type = \text{RSA}$” means that the encryption type that must be applied to the instance port or role is RSA. The value of Length measured attribute of Encryption is a choice between 128 bit or 256 bit (the `#` symbol is a CSP notation that means a choice). Note that a numeric value of a constant must be followed by a measurement unit when applicable (in this case: bit). This reduces inconsistency due to different measurement units adopted by different components and connectors, especially in distributed heterogeneous systems. The measurement units used for a particular measured attribute, as well as conversions of these measurement units, are defined in ontologies of measured attributes. The measured attributes of Authentication specify that the size of the security token must be within 4 bytes inclusive and 8 bytes inclusive. The NF-SPECIFICATION for Performance also has 1 (port or role) parameter (PR). The contained expression “$PR.BufferSize \geq \text{50KB}$” and “$PR.ReadSpeed \geq \text{1MB}$” constrain the BufferSize and ReadSpeed measured attributes, respectively. For example, if it is observed during run-time that ReadSpeed dropped under 1 MB, the monitoring system should raise a performance error event. The Concurrency property requires two port/role parameters to ensure that Read and Write conditions for ports/roles cannot occur at once.

The NF-FILTERS construct lists all instances from the e-health example to which these non-functional specifications have to be applied. For example, “$Security.Encryption[Ph1.provide_descp]$” means that the provide_descp port of Ph1 component instance (of type PH) has to encrypt data according to the expressions in NF-SPECIFICATION for Security. Another example is that the role of the $P1$ connector instance (of type Pipe) has to ensure that all performance attributes are met (i.e., $BufferSize \geq \text{50KB}$ and $ReadSpeed \geq \text{1MB}$).

6. Conclusions

The paper has introduced extensions to WRIGHT ADL that enable incorporation of non-functional properties in an architectural description. The NF-ATTRIBUTES construct contain references to precise, structured, ontological definitions of measured attributes (and also their measurement units and various relationships), as additional information (e.g., when, how, and by which party is monitoring done) necessary for run-time monitoring. Expressions limiting values of measured attributes are specified in the NF-SPECIFICATION con-
struct. The NF-FILTERS construct enables application of specified expressions to various instances of architectural element (e.g., component ports and/or connector roles).

The key innovation of WRIGHT+NF is in the precision supported by the NF-ATTRIBUTES construct. Use of external ontologies enhances precision and reduces chances for semantic mismatches that could occur during component system interactions. In addition, such ontologies are scalable—they can be expanded with new properties, measured attributes, and/or measurement units relatively easily. The additional precision that comes both from ontological definitions and specification of details necessary for run-time monitoring eases automation at design and code generation levels, e.g., towards semi-automatic refinement of architectural descriptions into deployable implementations.

Additionally, our constructs are designed to increase specification reusability and decrease redundancy of similar non-functional property specifications. This is achieved through the use of ontologies and NF-SPECIFICATION parameters. In ontologies, measured attributes are grouped into logically related blocks so that their specifications can be reused.

Further, our constructs are divided into logically-related blocks that are relatively easy to maintain. Maintainability is also facilitated by independency (separation of concerns) between functional and non-functional descriptions. Non-functional properties usually change more often than functional properties and our constructs support that a change of non-functional properties does not require a change in functional properties. (However, a change in functional properties might require a change in corresponding non-functional properties.)

Although some ontology languages (including OWL) are complex and require non-trivial processing overhead, our approach allows use of simpler domain-specific ontology languages (e.g., see [6]). In any case, there is some additional overhead that reduces run-time performance (efficiency) of systems supporting WRIGHT+NF. However, the experience from other areas (e.g., [5, 6]) shows that the benefits of enhanced precision usually overweight this additional runtime overhead. Further, our constructs extend the popular WRIGHT ADL and use the CSP notation, which significantly reduces time and effort needed to learn them.

We have validated our solutions on detailed examples. However, the main weakness of our work is that we have not yet finished appropriate language processing tools for WRIGHT+NF. They can be implemented by extending the existing WRIGHT tools [16] as they are consistent with WRIGHT syntax and CSP notations. This is the most important item for our future work. The extensions will provide: (1) parsing of our specifications; (2) performing different kinds of syntax checks leveraging the FDR [1] (Failures, Divergence, Refinement) model checker; (3) interpretation of our specifications into code for system configuration and their run-time checking (this step is not yet complete even for WRIGHT). Further, we plan other tools that will check correctness of run-time behavior of the new constructs.

Acknowledgment

Dr. Khaleed Khan from University of Western Sydney has provided valuable information in the course Software Architecture that contributed to achieving this work.

References