Flexible Architecture Conformance Assessment with ConQAT

Florian Deissenboeck, Lars Heinemann, Benjamin Hummel, Elmar Juergens
Technische Universität München, Garching b. München, Germany

ABSTRACT
The architecture of software systems is known to decay if no counter-measures are taken. In order to prevent this architectural erosion, the conformance of the actual system architecture to its intended architecture needs to be assessed and controlled; ideally in a continuous manner. To support this, we present the architecture conformance assessment capabilities of our quality analysis framework ConQAT. In contrast to other tools, ConQAT is not limited to the assessment of use-case-dependencies between software components. Its generic architectural model allows the assessment of various types of dependencies found between different kinds of artifacts. It thereby provides the necessary tool-support for flexible architecture conformance assessment in diverse contexts.

1. INTRODUCTION
For virtually all software systems of significant size, an architecture specification exists that describes the system’s components and their intended interactions. However, studies have shown that software systems undergo an architectural decay throughout their life-time if no counter-measures are taken, i.e., dependencies creep into the system that violate the interaction patterns that have been originally specified [3,4,9]. As the original intended architecture was usually designed to support important goals like portability or performance, such deviations are prone to increase the cost of maintenance as well as operation. A classic example of an unintended dependency is a direct call of a UI component (on the top of a three-layered architecture) to the data access layer although this call is meant to be mediated by the application logic layer. Similarly, many applications pool platform-dependent functionality in a special component to improve portability. Calls to platform-specific functions not routed via this dedicated component are therefore violations of the intended architecture.

Unfortunately, most of the programming languages used today provide no means to describe intended and non-intended dependencies beyond the level of classes and, possibly, packages. Hence, an analytic approach is required to assess the conformance of a software system’s architecture to its intended architecture.

Problem. Conformance assessment is challenging as the artifacts that comprise real-world software systems are of diverse nature. Among others, they include source code in different all-purpose programming languages as well as in domain-specific languages, libraries in binary form, various kinds of configuration files (e.g., for O/R-mappers), database artifacts like tables and views, plus a host of external resources like web-services. Crucially, a software system’s dependencies are not limited to dependencies within a specific artifact type but often cross these boundaries, e.g., a Java class querying a database table. An architecture conformance assessment tool must therefore be flexible enough to deal with the various artifact types and their dependencies.

Contribution. This paper presents a tool that addresses this challenge with a highly generic and flexible architecture conformance assessment approach. Our approach reduces the problem of architecture conformance assessment to the comparison of two hierarchical graphs; one that describes the intended architecture and one that describes the actual system’s dependencies. In contrast to other tools, conformance assessment is not limited to a certain language or particular type of dependency. The intended architecture is conveniently specified with a graphical editor that also allows to specify the mapping between architectural components and system artifacts. The analysis of the actual dependencies of the system is performed with the quality analysis toolkit ConQAT\(^1\) that provides the extraction of different dependency types for various languages. Moreover, new dependency extraction functionality can be added easily. The results of the conformance assessment can either be viewed in the graphical editor or included in a quality dashboard and thereby support the continuous control of architecture conformance. ConQAT’s architecture conformance assessment capabilities have been applied for architecture evaluations in multiple industrial projects and are used, among others, by the Munich Re and ABB in a continuous manner.

2. CONFORMANCE ASSESSMENT
To assess the architecture conformance of a system, three ingredients are required: (1) a machine readable specification of the intended architecture, (2) a mapping between the

\(^1\)Available as Open Source at http://www.conqat.org/
architecture's components and the system's artifacts, and
(3) knowledge of the dependencies between the system's ar-
tifacts (extracted automatically). Based on these inputs, the
actual assessment can be performed, which is usually embed-
ded in an iterative process of assessment and changes to both
the system and the architecture description. Our approach
corresponds in principle to the reflexion model technique [6].

2.1 Intended Architecture

ConQAT employs a simple hierarchical component model
for specifying the intended architecture of a software system.
It consists of the decomposition of the system into compo-
nents and dependency policies between them. A graphi-
cal editor based on the Eclipse framework and integrated
into ConQAT is used for the specification of the architec-
ture (Fig. 1).

There are three types of dependency policies: allow, deny
and tolerate. An allow policy defines that a component may
depend on another one. A deny policy defines that a com-
ponent must not depend on another one. A tolerate policy
is similar to a deny policy, but has attached a list of im-
plementation artifact dependencies that are tolerated. The
support for tolerated dependencies was introduced as a trib-
ute to industrial practice. Tolerations provide a mechanism
to mask existing violations until they can be resolved, while
ensuring that no new violations of the policy occur.

A component can have the PUBLIC stereotype indicating
that every other component within the same parent compo-
nent may depend on it. This is useful for library components
that are used by many other components.

2.2 Artifact Mapping

To relate the system to the architecture description, a
many-to-one mapping between system artifacts and the com-
ponents of the architecture has to be established. Which
level of detail is used when identifying artifacts can be con-
figured in ConQAT. To identify the artifacts of the system
we use the unique ID assigned by ConQAT to analyzed el-
ements. For Java classes this would be the fully qualified
class name including the package name, for C# classes this
is the full class name including any surrounding namespaces,
for files just the full file name is used.

The actual mapping is described by two sets of regular
expressions associated with each component of the architec-
ture. An artifact is mapped to the component if its ID is
matched by any expression in the first (include) set and not
matched by any expression in the second (exclude) set. As
Java packages and file names are organized in a hierarchy,
often the regular expressions are just used to select entire
packages or directories (e.g., package edu.tum.cs.conqat.core
maps to component core). So, if the system's structure re-
sembles the architecture's decomposition, the mapping is
fairly simple. However, the scheme is expressive enough to
capture more complex mappings.

2.3 Dependency Extraction

The extraction of dependencies is highly flexible as we can
utilize the full spectrum of the existing ConQAT framework.
An annotated example using ConQAT’s data-flow configu-
ration language is shown in Fig. 2, which illustrates the con-
furation used for architecture conformance assessment. As
the mapping and assessment works directly on plain depen-
dencies, for the dependency extraction part any ConQAT
analysis can be used that annotates an artifact with a list of
the ids of artifacts it depends on.

Probably the most common dependency types analyzed
are call or use dependencies. Call captures the invocation of
a function or method in another artifact, while use captures
general dependence which besides invocation also includes
inheritance or use of a type in attributes, parameters, or
local variables. Both can be extracted for Java and .NET
systems from the byte-code. ConQAT also allows to extract
the create dependency, i.e., object creation.

Another class of dependencies can be analyzed by replac-
ing the dependency extraction part by processing steps from
ConQAT’s clone detection sub-framework [5]. In one pos-
sible configuration code files could depend on each other if
they contain a piece of code which is (nearly) the same,
called a clone. This cloning dependency is relevant for two
reasons. First, duplicate code often needs to be changed
consistently, thus also the corresponding components require
consistent evolution. Second, cloning is sometimes used to
circumvent architectural constraints. For example, if the ar-
chitecture disallows to call a certain method from another
module, the developer could simply copy the method to
avoid an explicit dependency. As such behavior often in-
dicates either a problem with the architecture or a missing
understanding of the developer, finding these dependencies
can be relevant.

Beyond these artifact internal relations, often dependen-
cies can also be extracted from secondary systems and de-
scriptions, which slightly shifts the focus to assessment of the architecture itself. For example change management systems usually have a notion of a commit or change set. Thus these systems can be queried for files which have been changed together. By extracting this information and annotating the artifacts with it, ConQAT can also be used to analyze the change coupling [11] between artifacts and thus architecture components.

2.4 Assessment

The assessment compares the actual architecture to the specified intended architecture. Implementation artifacts are mapped to the components. Artifacts that cannot be mapped to any component—called orphans—are recorded and included in the assessment report. After the mapping, the dependency policies are assessed. In case there is no explicitly modeled policy between two components, the policies of the parents in the component hierarchy are considered. If no explicit parent policy is found, the dependency is considered as implicitly forbidden and therefore is treated as a deny policy. All components are implicitly allowed to access components that are transitive parents or children within the component hierarchy.

During the assessment, each policy is assigned a rating. An allow policy is rated valid if there is at least one implementation artifact dependency between the matched types of the components. Otherwise it is rated unnecessary. The notion of an unnecessary rating is useful for detecting superfluous dependency policies in the architecture specification. A deny policy—be it explicit or implicit—is rated valid, if either there are no corresponding implementation artifact dependencies or the target of the policy has the PUBLIC stereotype. Otherwise the rating invalid is assigned. A tolerate policy is rated valid, if all implementation dependencies are in the list of tolerations of the policy, invalid otherwise.

2.5 Process Support

We promote an iterative approach for introducing architecture conformance assessment in development projects. Our suggestion is to start with an initial architecture specification manually created from documentation (if available) and the implicit knowledge of team members elicited in an initial workshop. Once this version of the architecture specification is created, a first assessment is performed. Our experience shows that this first assessment will usually reveal many deviations between the intended and actual architecture. This is due to the loss of architectural knowledge in the project life cycle. The architecture specification is then refined in several iterations such that only those violations remain that are considered real problems and need to be addressed by realigning the implementation with the intended architecture. In case this cannot be done immediately, e. g., for organizational reasons, tolerations can be used to express this fact in the architecture specification. This iterative refinement is supported by the graphical editor which allows to interactively inspect the assessment. Upon opening the assessment report, the architecture definition is overlayed with icons indicating the ratings of the policies. The matched types for the components are displayed in a list. The assessment mode of the architecture editor allows to directly refine the architecture specification by menu actions for allowing or tolerating violations.

Once established, conformance assessment should be performed continuously in a development process [2, 8]. Ideally this is done in the context of a continuous quality control process. This can be accomplished by including the architecture assessment with ConQAT in a nightly build. For this purpose the assessment generates the report in HTML format that can be integrated in quality dashboards. This ensures that new architectural violations are detected early and can therefore be fixed with reasonable effort. Moreover the development team is forced to update the architectural specification in case the system’s architecture changes. This way, the architecture specification is turned into a “living” artifact. According to our experience, this is another valuable benefit of continuous architecture conformance assessment.

3. INDUSTRIAL EXPERIENCE

ConQAT is employed for architecture conformance assessment in several contexts.

Munich Re. Munich Re is one of the largest re-insurance companies in the world and employs more than 47,000 people in over 50 locations. Munich Re develops and maintains several business information systems to support its business processes. Continuous quality control of the development and maintenance efforts is supported through (ConQAT-based) quality dashboards that collect various quality indicators. These dashboards contain architecture conformance assessment results that are computed on a daily basis. This way, architecture violations can be discovered shortly after their creation, when their removal is still inexpensive. Consequently, ConQAT helps to reduce or even avoid architectural decay. Furthermore, we have found that conformance assessment can be a catalyst for architecture discussion—it helps to foster a common understanding of the intended architecture among developers. Due to positive experiences using ConQAT for quality assessment in three systems [4] over the course of two years, ConQAT architecture conformance assessment is currently introduced into the development process of further systems at Munich Re.

ABB. ABB is one of the world’s leading power and automation engineering companies. It employs about 115,000 employees in more than 100 countries. Similarly to Munich Re, ABB applies ConQAT to control the architectural conformance of a mid-sized C#-system that is used by ABB customers to configure the hardware products. Due to ABB’s worldwide operations, the development of the system is carried out, among others, at locations in Finland and India. ABB found that, particularly, in globally distributed software development, an automated mechanism that continuously controls architectural conformance and reports deviations to product managers in a timely manner, is of paramount importance. The company is, hence, currently working on integrating the ConQAT-based architecture-conformance assessment into their development tool-landscape as well as their processes.

ConQAT. The ConQAT code-base is used in lab courses, where 10 to 15 students spend several consecutive weeks extending ConQAT. Although some students stay on after a lab is finished, most move on to other tasks. This constellation causes a large turnover in the ConQAT development
team. As one measure to nevertheless achieve and preserve high maintainability of the source code, architecture conformance assessment is performed hourly to establish and persist a common architecture understanding among developers.

4. BEYOND CODE

The flexibility provided by the architecture description formalism and the open analysis platform allows application of architecture conformance assessment beyond source code. We successfully employed ConQAT to analyze conformance of the database architecture of a large industrial business information system at Munich Re.

The analyzed system has grown out of several previously independent systems. The database schema contains sub-schemas of its 11 constituent systems. These sub-schemas contain over 700 different database entities, e.g. tables and views. Rules govern access between entities located in different sub-schemas. For example, tables in other sub-schemas are encapsulated through views and must not be accessed from another sub-schema directly.

Each database sub-schema was modeled as a component. Sub-components were created for the tables and views. Policies were added between sub-schema components to allow access to views, but not to tables. Dependency extraction was performed via queries to the meta tables of the database management system—identifying over 1300 dependencies between database entities. Conformance assessment revealed multiple violations of the architecture rules. Examples include cross-subsystem accesses to tables instead of to the views that encapsulate them—making maintenance of these tables difficult. Prior to using ConQAT, an automatically laid out graph of the database entities and their dependencies was used to manually identify violations. Because of the size of the database schema, this graph grew to enormous size, significantly degrading its usefulness. In contrast, due to aggregation provided by the intended architecture, conformance assessment with ConQAT scales well.

5. RELATED WORK

Existing tools for architecture conformance assessment can be categorized into the three following approaches: source code query languages, dependency structure matrices and reflexion models [8].

The tools using source code query languages like .QL [1] and dependency structure matrices [10] specify the intended architecture directly on the basis of the system’s artifacts as they do not separate the architecture specification from the mapping to the system’s artifacts. Consequently, they cannot provide an architecture specification that is independent of the system’s implementation. However, such an artifact is vital as it serves documentation purposes and eases communication with stakeholders. Furthermore, experience shows that many stakeholders are most familiar with graphical architecture description and, hence, reluctant to deal with textual query languages or the matrix-based descriptions used by tools like XDepend and Lattix LDM.

In a comparison of architecture conformance checking techniques, Passos et al. recommend reflexion model based tools for integration of architecture conformance checking in a development process [8]. There are several tools that implement the reflexion model approach (Bauhaus, Depen-

6. CONCLUSION

In this paper we introduced ConQAT’s architecture conformance assessment capabilities. While none of the individual features are completely new from a research perspective, their combination and implementation in an analysis framework is in our opinion a valuable contribution to the community. The graphical editor simplifies the creation of architecture descriptions and the interpretation of assessment results. The combination with the flexible ConQAT framework allows the assessment regarding different types of dependency with low configuration efforts. Several industrial applications confirmed the practical relevance and applicability of our approach, but we also envision this tool platform as a basis for future experiments and case studies in architecture conformance and assessment.

7. REFERENCES


