Abstract—Software architects principally leverage successful architectural design practices systematized in terms of architectural styles and tactics. While architectural tactics focus on particular non-functional properties, styles are typical solutions that generally cover several aspects. The problem is that we do not yet have a formal account of how styles and tactics interact in a way that is sufficient to enable automated synthesis of architectures from application models and the combination of architectural styles and tactics specifications. The contribution of this paper is an extension of our previous work on formal architectural maps that makes this fundamental relationship clear, and a demonstration that it enables automated formal derivation of architectures.

Keywords—Software Architecture; Architectural Styles; Architectural Tactics; Architectural Maps; Alloy Language.

I. INTRODUCTION

Software engineering researchers and practitioners have recognized software architecture as a promising means of managing the complexity of software systems [20], [22]. Other studies have further shown its significant role in achieving systems’ non-functional properties [7], [13]. Developing a sound and appropriate architecture, however, remains a significant and intellectually challenging activity.

Notwithstanding the growing body of research and significant progress in architecture-based software development, the success of architecture construction still mainly depends on the experience of software architects [14]. As software systems unrelentingly grow in complexity and size, designing such systems manually in this way becomes more costly and labor-intensive, and, once completed, hard to change. Moreover, failure to meet the requirements, which might be later discovered, causes the need to backtrack, which is expensive especially in the case of architectural decisions.

To address this issue, we proposed a model-based approach to automated synthesis of architectural descriptions from formal, abstract application models and separate choices of architectural style specifications [3]. While our previous work deals with issues of architectural structure, it does not address support for non-functional properties.

Architectural tactics [5] have emerged as architectural decisions that codify and record best practice knowledge for achieving a desired quality attribute. However, despite years of research and practice, we have little formal account of how styles and tactics interact. Furthermore, the reusability of architectural tactics is hampered by a number of factors, among which is the absence of formalization, as they are typically represented as a set of documents. In fact, unlike architectural styles, characteristics of an architectural tactic are not understood in such a formal manner, making it difficult for architects to make informed choices.

In this paper, we present a novel approach based on our formal notion of architectural maps that makes this fundamental relationship clear. In particular, we show the feasibility of incorporating the notion of tactics with that of styles to enhance the specification of architectural maps, which provides the capability of formally applying refinement tactics to improve quality attributes.

This paper makes three principal contributions in this area. First, we formalize and automate architectural tactics by means of constraint predicates parameterized by choices of application type and architectural style. Second, we show that our ideas for formal and reusable representation of tactics parameterized in this way can be realized in practice. More specifically, starting with an application model and using the formal definition of the architectural maps as well as the architectural tactics provided in this work, it is possible to arrive to the set of compliant architectural descriptions. Third, we have developed the technique as an extension of our tool for model-based development of software architecture, Monarch [1], which leverages Alloy as the basis for the specification of mappings’ semantics and the automatic analyzer [15].

The rest of the paper is organized as follows. Section II explains the background context of our work. While Section III presents our approach for incorporation of architectural tactics into the formal synthesis of software architecture, Section IV demonstrates that it can be realized in practice. In Section V, we report on experiences applying our approach and tool set and provide a discussion. Section VI briefly surveys related work. Section VII finally concludes.
II. BACKGROUND: ARCHITECTURAL MAPS

In this section, we provide an overview of the notion of architectural maps, introduced and formalized in our previous work [3] based on the idea that choices of architectural style are made separately from choices of essential application properties, and that architects in essence implement a mapping that takes an application model and an architectural style to an architectural model. Knowledge of this mapping is crucial to expertise in software design. Given an application description of some type, the experienced designer knows both what architectural style to pick, and how to map an application description of the given type to an architectural model in the chosen style. In some sense an architectural map embodies knowledge of how to realize different types of applications in different architectural styles.

Figure 1, which is an extension of our previous work, represents the fundamental elements of our model on the basis of the architectural maps and their relationships: $s$, an architectural style specification; $\{a_i\}$, a set of architectural models; a binary relation, conforms, on the cross product, that encodes the conformance of the set of architectural models, $\{a_i\}$, to an architectural style, $s$; $t$, an application type specification; $m$, an application model; an analogous binary relation, conforms encoding the conformance of a given application model, $m$, to a given application type, $t$; and a relation, refines, encoding the notion that a set of application-specific architectural models, $\{a_i\}$, refines, or implements, the application model, $m$. Finally, we have $\text{map}(t,s)$, which takes an application model, $m$ of type $t$, and an architectural style, $s$, to the set of architectural models, $\{a_i\}$, such that $\text{refines}(a_i,m)$ and $\text{conforms}(a_i,s)$.

More recently, we have shown that the proposed separation of concerns supports a model-based development and tools approach to architectural-style-independent application modeling, and architecture synthesis with style as a separate design variable [4].

III. APPROACH

Above and beyond providing a proof of concept of the feasibility of the proposed formal architectural maps in an automated way, previous works revealed deficiencies in current specifications of architectural styles. That is, applying architectural maps leads not to a single plausible architecture, but to a set of architectures, where not any instances in that set necessarily satisfies all required non-functional properties. In fact, as can be seen in the diagram, one more optimizing search step is required within that space to find the subset of plausible instances especially with respect to quality attributes. This is mainly because style specifications to which application models were being mapped are still underspecified, and in turn their corresponding architectural maps leave overly large architectural spaces.

As such, the systematic process of getting from an application model to an architecture involves two steps: (1) constraining the formal architectural space; (2) performing optimizing search over the derived space for properties of interest. The more we can do in the mapping, the less we have to do in the search. Therefore, to achieve a practical automated design tool for software architecture, there is a pressing need to formally specify relationships between architectural design decisions and quality attributes [7].

Architectural tactics are architectural decisions concentrating on non-functional properties that shape the system’s architecture but in a smaller extent compared to the style decisions [5]. Several architectural tactics have been proposed for various quality attributes such as reliability, performance and modifiability. As architectural styles have extensive use in the architectural modeling, relating styles to tactics provides a basis for making rational design decisions [5]. That is, tactics can systematically be leveraged to adapt styles for the purpose of improving their properties with respect to particular quality attributes. As such, the structure of the tactic must fit within the rules implied by the style. To effectively apply architectural tactics along with the styles the architect needs to realize their side effects and how they relate each other [14].

This paper develops the notion that architectural tactics are not independent of, but rather are parameterized by, choices of application type and architectural style, and it is possible to represent tactics parameterized in this way in a formal and reusable form. Architectural styles and tactics then can be considered as complementary techniques, subsequently allowing for the automated reuse of architectural best practices. Indeed, the effect of a tactic refines and extends the architectural map.

$$\text{map}(\text{AppType, ArchStyle}) \xrightarrow{\text{extends}} \text{tactic}(\text{AppType, ArchStyle})$$
In our approach, one expresses an application model as an instance of an application type, then selects an architectural style in which to synthesize an architecture for the application. The combination of application type and architectural style selects a mapping from applications of that type into architectural instances in that style. The decision to use a particular tactic in mapping an application model to an architecture then involves the explicit inclusion of such a reusable tactic specification along with a more generic architectural map. Applying this compound map to the application model yields a family of correct-by-construction architectural descriptions for the given application in the given style supporting the given tactics. In the next section, we show that our ideas can be reduced to practice.

IV. Automation

To demonstrate the viability of implementing tools that support automated formal synthesis of software architectures complying with the rules implied by choice architectural styles and tactics, we present a proof of concept leveraging the Alloy language for formalization and the Alloy Analyzer for automation.

Alloy is a lightweight specification language based on the first-order relational logic [15]. Signature is an essential construct of the Alloy language that represents the basic type of elements and the relationships between them. Facts can be used to define constraints over model instances. Alloy further provides Predicates to be used in defining parameterized reusable constraints always evaluated to be either true or false. A Function similar to a predicate can be invoked by instantiating its parameters, but what it returns is either a true/false or a relational value instead.

Its SAT-based analyzer makes automatic analysis of Alloy models possible. The Alloy Analyzer can be used either to find satisfying solutions with respect to the constraints of a given model, or to find any possible counterexample violating constraints in a given model. The former is used in our approach to compute architectural models.

Five pieces of Alloy specifications are conjoined in the process of synthesizing compliant architectural models: (1) an application type represented in an Alloy module; (2) an architectural style specification module; (3) an application model, comprising an instance of an application type, again represented in an Alloy module; (4) an architectural map specifying the relationships required to hold between an application of the given type and an architecture in the given style, represented in an Alloy predicate; and (5) a tactic predicate specified for a given pair of an application type and an architectural style to incorporate a given tactic into the software architecture. The Alloy Analyzer computes satisfying solutions to the compound specification, leading to the synthesized architectures.

V. Evaluation

Our claim we make in this paper is that it is feasible to represent architectural tactics parameterized by choices of application type and architectural style in a formal and reusable form. In this section, we report and interpret data from experimental testing of our approach and hypothesis. More specifically, we first show that our approach supports formal specifications of architectural tactics with respect to the architectural maps developed for each pair of an application type and an architectural style. We then show that it also supports automatic synthesis of architectural instances which conform to the rules implied by both the application model and the architectural style as well as supporting the given architectural tactic.

In support of our approach, we have extended our tool suite, Monarch, which is available for download and inspection [1], and applied it to several case studies. This paper cannot accommodate detailed presentations of all of our experiments. Rather, we report one case in more detail, which is about the employment of a fault detection tactic in the process of formal synthesis of Lunar Lander architecture [22] in the implicit-invocation (II) style from its abstract application description in the sense-compute-control (SCC) application type. This section is followed by presenting discussion over the experiments.

A. Application Model: Lunar Lander\textsubscript{SCC}

Our application case study is inspired by Taylor et al. [22]. In their new textbook on software architecture, they illustrate the structuring of an embedded control system called Lunar Lander in different architectural styles. The FlightController analyzes the information provided by various sensors, i.e. Altimeter, Gyroscope, Fuel level indicator and Engine control switch, to maintain the state of a spacecraft. Then, it provides updated data to the various actuators: Descent engine controller, Attitude control thruster and Display.

Figure 2. Lunar Lander application modeled within GME using the generated modeling environment for our SCC metamodel.
Figure 2 illustrates Lunar Lander application modeled as an instance of our SCC metamodel within Generic Modeling Environment (GME). We customize GME such that application types are realized concretely as GME metamodels, providing architecture-independent modeling languages (AIML). We specify a formal model of the SCC type to model applications in which sensors and actuators are connected to controllers that cycle through the steps of fetching sensor values, computing function values, and sending outputs to actuators.

```plaintext
module LunarLander
open SCC

one sig Altimeter extends Sensor

one sig Gyro extends Sensor

one sig EngineControlSwitch extends Sensor

one sig AttitudeControlThruster extends Actuator

one sig Display extends Actuator

one sig FlightControl extends Controller

sensors= FuelLevel+ EngineControlSwitch+ Gyro+ Attimeter
actuators= DescentEngController+ Display+
AttitudeControlThruster

caller = periodic

caller = fast

program = controller_code

...
```

Listing 1. Lunar Lander application description represented in Alloy

Listing 1 partially outlines the Alloy representation of the Lunar Lander application description, automatically synthesized from its concrete representation. The application model (an Alloy module) imports the SCC module, and defines four sensors and three actuators using signature extension to subtype the Sensor and Actuator types. The only controller of the application is then synthesized, while its properties are specified as Alloy facts.

B. Tactic: PingEcho(SCC,II)

PingEcho and Heartbeat are two architectural tactics proposed for Fault Detection, which is one of the four design concerns for reliability [5]. In this case study we employ the PingEcho tactic, although the structures of these two tactics are very close. For detecting a fault using the aforementioned tactic, a monitor component regularly sends ping messages to receivers. The receiver is then supposed to respond with an echo message within a certain time period, otherwise it is considered to have failed.

Listing 2. The Alloy predicate for Ping/Echo tactic developed for the architectural map of the pair of SCC application type and II architectural style

```plaintext
module PingEcho_SCC_II
open SCC

open II

pred pingEcho_SCC_II [s, set: needHandle] {
  one o: IObject | o. handle = Pinger & &
  no handled: needHandle | handled in o. handle & &
  one p: Procedure | p in o. ports & &
  all n: needHandle | n in s => {
    one c1: procedureCall | c1 in n. handle | procedure[ attachments ], ran. connector& &
    c1 in o. call[ attachments ], ran. connector
  }
}

...
```

Listing 3. The Alloy module for incorporation of Ping/Echo tactic

After modeling the application and specifying the mappings and tactics predicates, we need to define a compound module incorporating tactics’ specifications into the process of synthesizing satisfying architectural models. Here, we have leveraged the architectural map for the SCC application type and the implicit-invocation architectural style specified in earlier work [4].

```plaintext
module LunarLander_SCC_II
open LunarLander as AppDesc
open SCC_II as map_SCC_II
open PingEcho_SCC_II

pred execute {
  map_SCC_II ()
  PingEcho_SCC_II [FlightControl] }
```

D. Satisfying Architectural Models

Given all the specifications and mapping constraints, Monarch using the Alloy Analyzer computes a set of architectural models that refine the application description in conformance with the fully formal definitions of the implicit invocation architectural style and the Ping/Echo tactic.
represented as Alloy solutions. To make those low-level, XML formatted outputs human-readable, it then using our Alloy2ADL transformer [4] converts them to an architecture description language (ADL).

A considerable number of ADLs have been proposed during the past several years to model specific application domains or as general purpose architectural modeling tools. We use Acme [11] in our tool, which is a mature general purpose ADL, with a particular support for architectural styles. It is also designed to work as an interchange between wide varieties of architecture description languages. Figure 3 shows an instance of the computed architectural models in Acme. In general, there is more than one satisfying solution, and the result is a set of formally derived architectural models for the given application with respect to the enforced architectural constraints.

E. Discussion

This work shows that architectural tactics can be formalized and implemented as executable specifications. By incorporating both the architectural styles and tactics during the synthesis, our approach formally derives narrower architectural spaces, which reduces the difficulty of design space exploration. We note that automated search could occur not only within one architectural style but across styles incorporating a set of tactics – leading to a tool able to automatically find appropriate architectural models for a given application model.

It is also worth mentioning that the incorporation of a tactic within a style might result in either the alteration of the style or even addition of a style to the architecture [14]. As a case in point, by application of the Ping/Echo tactic to an architecture developed upon the Pipe and Filter style, the derived architecture contains an added component in charge of performing monitoring process and having connections with the pinged components (filters), which consequently imposes Broker style into the architecture. The current version of Monarch avoids such mismatches by deriving an empty set of architectural instances, as there is no such architectural model that thoroughly complies with rules implied by the Pipe and Filter style and embodies an instance of the Ping/Echo tactic simultaneously. As a future work, we plan to investigate ways of handling such cases.

Another interesting avenue for future work is to provide more rigorous representation of application models especially with respect to non-functional requirements (NFRs). The more complete the specifications of application models, and the more tightly constrained the mapping specifications, the narrower the outcome architectural spaces, and the slighter the required postprocessing optimizing search. Work by Rosa et al. [19] and more recently by Jackson et al. [16], proposing that many forms of NFRs can be considered as constraints over the space of models, suggest interesting possibilities in this regard, which we intend to explore.

Finally, the upshot is that those who are doing research in software architecture, patterns, and tactics should recognize the need to specialize tactics to the particular settings induced by a choice of both application type and architectural style, and could consider the function of tactics as extending and refining architectural maps.

VI. RELATED WORK

We can identify in the literature three categories of works that are related to our research. The first one concerns works that deal with formal modeling of architecture using Alloy. The second encompasses researches on leveraging architectural best practices to satisfy quality attributes. Finally, the last area of related work focuses on formal approaches to the evolution of architectures under the guidance of architectural styles.

Focusing on the first category, numerous researchers have applied Alloy to formal specification of software architecture. Among others, Kim et al. [17] proposed an approach to translate architectural styles described formally in an architecture description language to Alloy language, and in turn, to verify properties implied by architectural styles. Georgiadis et al. [12] similarly specified structural architectural styles as a set of constraints in order to control runtime reconfiguration by means of constraints evaluation. While these works use formal modeling and verification for detecting architectural mismatch [9], we focus on preventing such mismatches with a formal synthesis approach. That is, we use Alloy not only to specify the application model or architectural constraints, but also to model the spaces of mappings consistent with both given application models as well as target architectural styles and tactics, and to automate the mapping process.

Bucchiarone and Galeotti [6] also proposed an approach based on graph grammars and DynAlloy [8] to verify programmed dynamic software architectures in which all
possible architectural changes are defined before run-time. Although this work, like other work we have studied, lacks an explicit notion of separating application description from other design decisions, it appears to have the potential to help us extend application models to include richer semantics.

Focusing on the second category, a number of approaches explored relating architectural styles and tactics stimulated by them regarding various quality attributes (e.g. modifiability [2], reliability [14]). Extending the same line of work, Kumar et al. [18] recently proposed a more general pattern-oriented knowledge model composed of four dimensions, including the pattern to tactic relationship. Our work is different from these works in several ways. First, our work is geared towards the systematic specification of architectural models from abstract application models to the architectural styles and the associated tactics. Second, our approach has a formal basis which is missing in theirs.

Regarding the last area of related work, Tamzalit and Mens [21] recently proposed an approach for the evolution of an architecture description under the guidance of architectural style. To this extent, they rely on a formalism based on graph transformation. The other similar work is the recent work of Garlan et al. [10] on the evolution of programs with respect to architectural style. The premise of this work is that it is sometimes necessary to change a program written in one architectural style into a related program in another style. These works share with ours an emphasis on formal application of architectural best practices, but our work focuses on formal mappings of architecture-independent application models to a diversity of realized architectural models in practical architectural styles and tactics.

VII. Conclusion

While a wealth of research has been done on software architecture, architectural styles and architectural tactics, very little has been done on automated support for the derivation of software architecture with respect to styles and tactics. We presented an approach based on the notion of architectural maps, that formally supports automatic synthesis of architectural models from abstract application models in conformance with the architectural styles. In this paper, we showed that architectural maps can also incorporate architectural tactics in a formal and reusable form.

References


