Model-based Assistance for Making Time/Fidelity Trade-offs in Component Compositions

Vishal Dwivedi, David Garlan, Jürgen Pfeffer and Bradley Schmerl
Institute for Software Research
Carnegie Mellon University, Pittsburgh, PA
{vdwivedi, garlan, jpfeffer, schmerl}@cs.cmu.edu

***Abstract*—In many scientific fields, simulations and analyses require compositions of computational entities such as web-services, programs, and applications. In such fields, users may want various trade-offs between different qualities. Examples include: (i) performing a quick approximation vs. an accurate, but slower, experiment, (ii) using local slower execution environments vs. remote, but advanced, computing facilities, (iii) using quicker approximation algorithms vs. computationally expensive algorithms with smaller data. However, such trade-offs are difficult to make as many such decisions today are either (a) wired into a fixed configuration and cannot be changed, or (b) require detailed systems knowledge and experimentation to determine what configuration to use. In this paper we propose an approach that uses architectural models coupled with automated design space generation for making fidelity and timeliness trade-offs. We illustrate this approach through an example in the intelligence analysis domain.**

I. Introduction

When software engineers compose existing components into larger systems, they have to make decisions about component selection from component repositories. These decisions are often based on a fixed set of quality trade-offs, where engineers aim for an optimal choice in some trade-off space for some particular context. However, increasingly, compositions may be reused in different contexts where the original trade-off decisions may not make sense, and other choices of components may in fact be more suitable. This problem manifests itself most particularly in domains where compositions are shared with a large community of users, such as in intelligence analysis, medical informatics, or scientific computing.

As an example, consider an intelligence analysis composition that analyzes social-network data for interesting patterns. An analyst who is more concerned about accuracy than time may choose components that use complete information and use complex algorithms to get a nuanced and accurate analysis of the data being examined. Such a complete analysis may take on the order of days or weeks, but can lead to information about the roles, knowledge, locations, and relationships of important actors in that set of data.

Another analyst may want to perform a similar analysis in a situation where information is changing rapidly and timeliness of an answer is important. While the steps to perform the analysis might be the same, the choice of components to perform them in this case will likely differ from those that the first analyst chose. For example, the second analyst may favor quick, less detailed (or low fidelity) analyses over a long and complete (or high fidelity) response. Rather than reuse the original workflow, they are forced to create a new composition specifically for their context, even though the steps are similar. Ideally, the second analyst should be able to reuse the original composition but indicate his trade-off decisions to quickly tailor the composition to his context.

In fact, in domains where composition reuse is common, there may exist multiple (versions of) components with different fidelities or components may be have configuration options to provide different levels of response. Navigating this trade-off space and choosing components to use in a particular context is a complex problem with which software engineers have some difficulty. For domains such as life-sciences and bioinformatics — where composition is usually performed by "professional end-user developers" [23] — making such trade-offs is understandably even harder as the process involves significant analysis and coding skills.

In this paper we explore how architecture models can be used to automate the appropriate configuration of compositions to support trade-offs between different levels of fidelities. The approach takes advantage of the fact that in a given composition, there can be many component realizations having different fidelity and time properties that can be used to realize alternative compositions. We call these compositions *abstract compositions*. Together with component repositories and fidelity/time information we use the Alloy model generator [1] to explore concrete realizations of these abstract compositions that take into consideration a user's fidelity compromises, and then estimate the time it will take to execute the compositon. The user can then explore the compromises on fidelity, see the corresponding time estimates, and make appropriate choices for the given circumstance.

The main contributions of this paper are: (i) a composition approach that enables trade-offs between fidelity choices and execution time, (ii) estimation of time and fidelity using order information, and iii) using model checking to generate compositions that optimize the trade-offs.

The rest of the paper is organized as follows. In Section II, we expand on the problem and introduce the requirements for providing automated fidelity/time trade-off compositions. In Section III, we describe our approach, which involves passing architectural descriptions of abstract workflows to Alloy that generates concrete workflows matching the desired fidelity points. Section IV gives related work, and Section V provides some discussion and future work.

II. Problem

In many scientific fields, simulations and analyses require computations with varying fidelity expectations. For example,

scientists may perform a quick approximation using lesser data, or perform computations with various fidelity trade-offs. In many such domains, composing heterogeneous computational entities, usually in the form of workflows or component assemblies, allows scientists to execute their analyses. In these scenarios, often the fidelity selection of datasets, components, and their configurations determines the timeliness of queries (and vice-versa).

However, such component compositions are difficult to specify. There are often inter-component constraints that may lead to mismatches [15]. There may also exist multiple alternative configurations, with a different set of parameters that may provide results with different execution times and fidelities. It is even more difficult when a user has to make trade-offs between fidelity and timeliness choices to select a *component configuration* that meets expectations since the number of possible configurations grows combinatorially. Furthermore, user's fidelity selection may not directly match a configuration and the nearest approximation may be required. Existing composition approaches do not provide a good mechanism to support such fidelity and timeliness trade-offs [8].

As an illustration, consider an example in the field of military intelligence (also called edge analytics) where soldiers rely on analysis and simulations to guide their operations. Today, such analytic capabilities are provided by tools and mechanisms that can transform information sources captured as unstructured input (e.g., incident reports, news sources, miscellaneous geo-spatial data) into complex network models that aid sophisticated analysis such as situational awareness, key entities, fact identification, and what-if exploration [5].

A common querying scenario is when a soldier observes suspicious activity and sends an incident report (see example below) to an operating base, where analysts and other experts can analyze the incident and respond back with a report.

> **Incident Report:** Lt. Col. Liz Abreams (Date: 2/16/2011) Set up sensor alert at checkpoint zulu-1, border crossing between Talodi and Malakal. Position sensor picked up 15 vehicles. Darfur escapees. Overcast. Positive ID on LP 6VES512. Orange. Passengers were Hasim Makul, Hassan Sayid Deng, Jon Deng, and Mary Okulo. Visible knifes. Possible narcotics."
> **Query:** Should we detain? Will maintain position till 1800.

In order to assess the situation and to answer the soldier's question, an analyst in a forward operating base must decide whether the new information leads to any significant changes in the existing network structures in that geographical area. An illustrative computational workflow (shown in Figure 1) for this query involves: (i) processing this incident report along with the network data, (ii) converting it into a graph-based model and (iii) using network algorithms to create and visualize the impact of the new event.

In this domain, a typical network contains millions of nodes with information about people, events, and locations. Therefore, it may save significant time to pre-process and cache this data at the expense of using potentially stale information. Other fidelity variations include: (a) reducing the quantity of data based on dimensions such as time (e.g., only consider this year, vs. consider all years), (b) space (e.g., only consider sources associated with Darfur and Sudan), (c) source (e.g., only consider sources from local reports), and (d) using faster approximations vs. slower but accurate algorithms. These, and other fidelity choices, may lead to different component assemblies with different execution times.

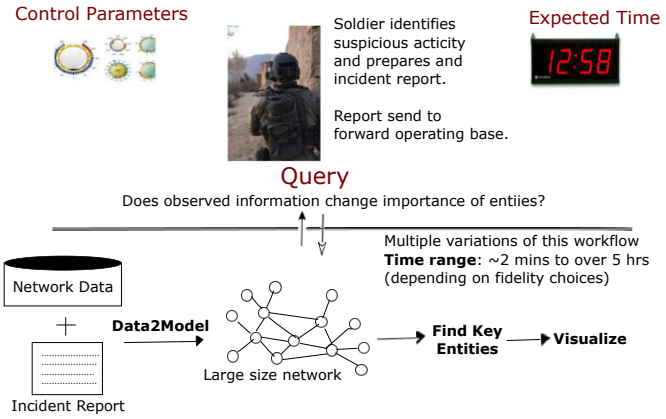

Fig. 1: Querying based on fidelity vs. timeliness.

A variation of the workflow from Figure 1 is illustrated in Figure 2 that has fidelity reductions in terms of using cached data with a faster approximation algorithm for computing key-entities. The end user (here, an analyst) provides the control parameters or fidelity expectations that can inform him about the expected execution time and help in the generation of the right computation assembly that serves his operational needs. While the workflow in Figure 1 takes more than 5 hours to execute, the one in Figure 2 takes about 2 minutes. This dramatic time saving is achieved by approximating the results by using a slightly older, cached network and a faster algorithm that uses a subset of the networks that deals with relationships between people from the Sudan network data (instead of using a collection of other relationships such as knowledge, resources, geospatial or temporal information, etc. that can provide a detailed, but slower, analysis).

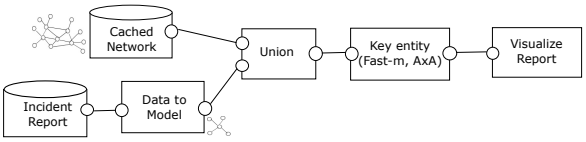

Fig. 2: A variation of the workflow in Figure 1.

The above example highlights interesting challenges in providing flexibility to the information analyst to process data and provide answers to questions in a timely manner. These challenges include:

1. Overwhelming component choice: How does the intelligence analyst map the abstract steps that must be followed to produce an answer to the actual program elements and services that are available? As we have reported previously [10], many domains including intelligence analysis have an abundance of components to choose from that provide similar functionality, and that can be parameterized for slightly different results. To map the abstract steps of the workflow to concrete elements, users in most cases resort to familiarity rather than applicability in the face of a lot of options.

2. Multiple fidelity dimensions: Analysts may want quick answers to questions, but they also need to understand what they are giving up for speed. In fact, there are typically multiple dimensions that need to be compromised for speed. In the example above, an analyst can choose to filter the data being used on several dimensions (time, geography), choose to use network analyses that do not follow potential paths in the network (and therefore cannot determine important attributes like grouping or connectedness), or choose to use only certain aspects of the networks (e.g., to focus on social networks and ignore knowledge or belief networks).

3. Inter-component dependencies: Further complicating the choice of components and fidelities is the fact that dependencies exist between the options in the composition. For example, choosing to use only the most recent data prohibits the use of trend analysis later in the workflow. Choosing a component that produces a certain restricted format of data may constrain the choice of downstream components that can be used.

III. Approach

An approach that helps to automate the above choices is required to simplify the composition of components for end users. In this section we describe an overview of our approach for providing a system that allows end users to explore the fidelity and time trade-offs in component compositions.

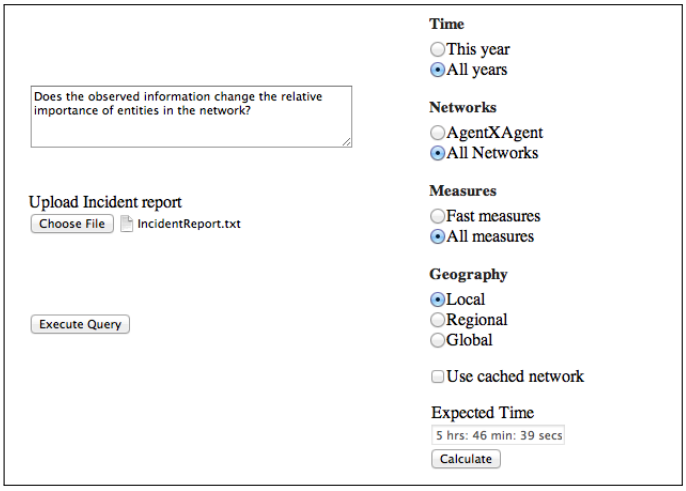

Fig. 3: A simple UI to perform fidelity time trade-offs.

A. Overview

To address the requirements above, we use an approach, illustrated in Figure 4, that allows a user to choose the fidelity options they are willing to make and receive information about the estimated time that the composition will take to perform a query. An expert analyst (who is a domain-expert) develops an abstract component composition describing the steps that are required to process a query. A component repository contains the instantiations of all components that provide concrete realizations of the steps in the composition, along with their fidelities and timing profile. To perform a query, a user fills in the fidelity options they are willing to compromise on. (See Figure 3).

We model the user's choices, the abstract compositions, and the concrete components in the Alloy modeling language [19].

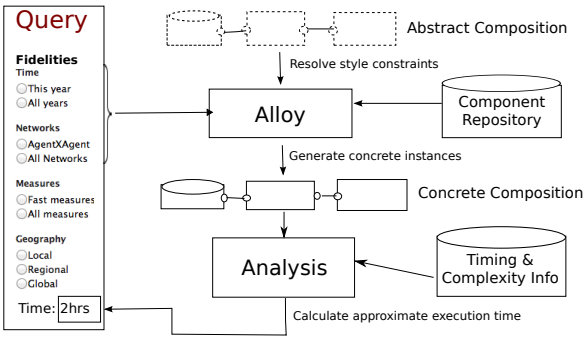

Fig. 4: Generating concrete workflow and timing.

An Alloy model is a collection of signatures and constraints that describes a set of structures, for example: all the possible configurations of a web application, or all the possible assemblies of a set of components that follow composition rules. The Alloy Analyzer is a solver that takes the constraints of the model and finds structures that satisfy them. It can be used both to explore the model by generating sample structures and to check properties of the model by generating counterexamples. Specifically, for our problem, we use Alloy's model-generation capability to generate compositions required to answer queries from analysts (described in more detail below).

In our approach, we consider two levels of compositions: (i) abstract compositions that are defined by an assembly of components that have some high-level properties, constraints and functions, and (ii) concrete compositions that are an assembly of computational elements that implement those functions. The high-level choices made by the end-user help in the selection of the abstract components. The Alloy model generator generates concrete compositions that satisfy composition and fidelity constraints. This concrete composition is then further analyzed (as explained later), using complexity and timing information, to generate an approximate execution time that can be given back to the analyst, who can then either choose different fidelity options, or run the concrete composition.

When the analyst chooses to execute a concrete composition, it is translated into a script that can be executed. In our current prototype, we generate a BPEL script that uses components derived from existing intelligence analysis tools deployed on the SORASCS platform [9].

B. Architecture representation of compositions

In our earlier work, we proposed compositions as end-user architectures [10] that can be explicitly represented as architectural models defined in a domain-specific architectural style. Such architecture models can be used not only for various analyses, but they can also generate executables. Such architectures form the basis of composition and fidelity analysis. For instance, here we represent end-user compositions in SCORE [16] — a dataflow based style that is customized for the intelligence analysis domain.

We adopt the architecture analysis approach from Kim et al. [20] where architectural types are specified in Alloy as signatures and constraints (based on fidelity and other architectural properties). These are analyzed for type checking and model generation. Compositions in SCORE are converted to the Alloy specification language that is based on first-order

and relational calculus well suited for representing abstract compositions.

While details about modeling architectures and their analysis in Alloy can be found in [20], we walk you through an example modeling scenario in Alloy. Listing 1 shows a snippet of the architecture model in Alloy (defined as a configuration of components and connectors). We extend this specification to define two types of components — abstract and concrete. A snippet of mapping between abstract and concrete components is shown in Listing 2.

Listing 1 : Architecture specification in Alloy.

```
...
abstract sig Architecture {
    comps: set Component,
    conns: comps lone -> lone comps /*connectors modeled as
        a relation between components */
}{
    no (iden & ^conns) /* no cycles */
    all c: comps | comps in c.*conns + c.*~conns /* fully
        connected*/
}
/* Data dependencies are satisfied */
pred WellFormedArch (arch: Architecture) {
    all c1, c2: arch.comps | (c1 -> c2) in arch.conns => c1.
        output.data = c2.input.data
}
...
```

Listing 2 : A mapping between abstract and concrete architectures.

```
...
sig RealizationMap {
    absArc: one AbstractArchitecture,
    concArc: one ConcreteArchitecture,
    impls: ConcreteComp one -> one AbstractComp
}
{all cc,ca: Component | (cc -> ca) in impls => (cc in
    concArc.comps) and (ca in absArc.comps) /*Mapping only
    existing components*/
    all ca:absArc.comps | some cc:concArc.comps | (cc->ca)
        in impls /*each abstract has a concrete one*/
    all cc:concArc.comps | some ca:absArc.comps | (cc->ca)
        in impls /*each concrete has an abstract one*/
}
...
```

Concrete models are constrained to satisfy both the structure of the abstract model and the fidelity properties selected by the end user, represented in Alloy as properties and predicates to be satisfied in the model generation phase. A simple example of fidelity properties is described in Listing 3.

Listing 3 : Specifying fidelities.

```
...
/*Network types*/
one sig AxA, AllNetworks extends NetworkTypeFidelity{}
/*Data Size*/
one sig ThisYear, AllYear extends DataSizeFidelity{}
/*Speeds*/
one sig FastOnly, AllSpeeds extends SpeedFidelity{}
/*Data Regions*/
one sig Local, Regional, Global extends DataRegionFidelity{}
...
/*Pred to check if types are satsified*/
pred satisfiesDimensionType() {
    all f: Fidelity | (f in Speed => satisfiesSpeedType[f])
        and (f in NetworkType => satisfiesNetworkType[f])
        and (f in DataSize => satisfiesDataSize[f]) and (f
        in DataRegion => satisfiesDataRegion[f])
}
```

C. Generate concrete compositions

As we discussed before, we use Alloy's model generator to create concrete models representing compositions. The concrete assembly is constrained to follow data-mismatch [15] and fidelity constraints. As an illustration, see Figure 5 where a high level workflow is mapped to a concrete composition and modeled using types (or signatures) comprising of components, interfaces, their properties and the constraints. We do this by representing the abstract and concrete component vocabulary, along with a mapping function in Alloy and using its model generation capability that generates concrete instances given a set of high-level components and their properties.

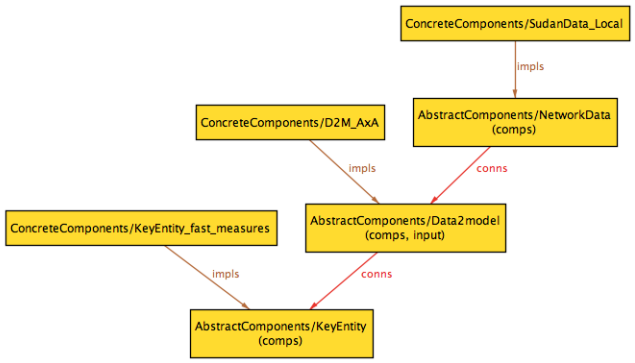

Fig. 5: An illustration of a mapping between abstract and concrete workflows in Alloy.

The abstract workflow in Figure 5 comprises three functions with varying input and output data requirements and fidelity constraints. This is mapped into a concrete workflow that includes services for individual functions and additional components for data translation and data fetching to generate a sound composition. The Alloy snippet in Listing 4 shows the function that is run by Alloy to create instances of concrete models given an abstract model within a specified scope of number of objects per signature (e.g., by using the "run for 15" command). A simplified concrete model is shown in Figure 5 (with additional details such as ports, properties and mappings turned off).

Listing 4 : Generating model instances in Alloy.

```
...
pred showConcreteFromAbstract{
    one RealizationMap
    one _AbstractArch
    _AbstractArch in RealizationMap.absArc
    one ConcArchitecture
    ConcArchitecture in RealizationMap.concArc
    Component in (ConcArchitecture.comps + AbsArchitecture.
        comps)
    SatisfiesFidelities[ConcArchitecture, fast, AxA,
        ThisYear, Local]
}
...
run showConcreteFromAbstract for 15 but 1
    AbstractArchitecture,
1 ConcreteArchitecture
```

D. Performance Analysis

Next, we perform an analysis that computes the execution time for the concrete compositions. We use properties of the networks and the complexity of the algorithms to approximate the execution times for components in the workflow, and aggregate them into the overall execution time. These approximations of execution time help end users to make an

TABLE I: A selection of centrality metrics

Metrics	Description	Reference	Complexity	Class
Degree Centrality	Number of neighbors	[25]	O(m)	fast
Hubs and Authorities	Power centers and connectors	[21]	O(m)	fast
Eigenvector Centrality	Power centers	[3]	$\sim$O(n^2)	medium
Clique Count	Part of dense groups	[4]	$\sim$O(nm)	medium
Cognitive Demand	Involvement in many activities	[6]	O(nm)	medium
Situation Awareness	Overview of activities	[18]	O(nm)	medium
Betweenness Centrality	Control communication flow	[17]	O(nm+ n^2log n)	slow

informed choice about the timeliness of compositions based on selected fidelities.

To illustrate the approach, consider "Generate Key Entities." *Key entities* refer to important agents in a network. Social network analysis has developed a wide array of metrics to describe the position of individuals within a graph-like structure consisting of nodes and edges [25]. As different metrics focus on different aspects of network importance (e.g., centrality), analysts often calculate a set of centrality metrics for identifying and ranking the important nodes in a network. Calculating these metrics and comparing the results with previous results can help the analyst to assess whether the newly observed situation creates significant change in the network.

Table I shows a selection of these metrics with their computational complexity. This enumeration is a selection to illustrate our approach; an actual application scenario consists of many more metrics.

The right column of Table I shows the complexity class that we use to filter the algorithms — n refers to the number of nodes and m to the number of edges. For estimating the calculation time, we first summarize the metrics complexities grouped by complexity class:

$$t_{fast} = O(m) \quad (1)$$
$$t_{medium} = O(nm + n^2) \quad (2)$$
$$t_{slow} = O(nm + n^2 logn) \quad (3)$$

Based on a user's explicit or implicit (by timeliness/fidelity options) selection of metrics we can determine the aggregated calculation complexity of all metrics that are included in the calculation. For instance, in order to calculate all metrics, the accumulated calculation complexity is:

$$t_{all} = t_{fast} + t_{medium} + t_{slow} \quad (4)$$
$$= O(m + nm + n^2 + n^2 logn) \quad (5)$$

As we know n and m (the number of nodes and edges in the network), we can estimate the actual calculation time in seconds of any combinatorial set of metrics after an initial "calibration" procedure for which we calculate a selected (small) set of metrics on a given network with a given machine, and measure the time in seconds needed for the calculations. It is to be noted that we ignored effects such as paging that may be evaluated for further optimization.

Using a small-world network [26] with 10,000 nodes and 50,000 edges we calculate betweenness centrality. This takes 80 seconds on a regular laptop computer. With this number we can estimate the timeliness of all combinations of metrics.

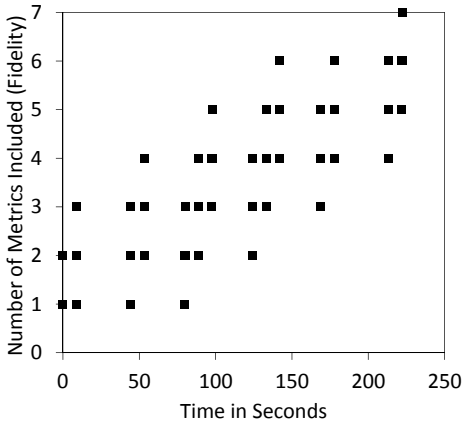

Fig. 6: Timeliness/Fidelity option space.

Even though our example consists of just seven metrics, there are 127 combinatorial options (2^7-1). The complete option space is visualized in Figure 6; some dots represent more than one metrics combination. In case we have no constraints or preferences on metrics this figure tells us that, for example, we can execute up to 5 metrics within 100 seconds.

E. Orchestration Generation

The concrete workflows generated by Alloy provide valid component compositions that obey configuration and respect fidelity preferences. They define a sequence of web-services provided by the SORASCS platform [9]. These concrete compositions are compiled into orchestration scripts and executed using a standard SOA infrastructure. The final result is the output of executing the service-composition. For the composition scenario discussed in the paper, it is a key-entity report that executes a set of centrality metrics (based on user selection) for the analysis query.

F. Implementation

To demonstrate our approach we implemented a prototype web-application (using the example scenario described in Section II) where users can specify their fidelity choices based on expected execution times. These fidelity choices are incorporated in an Alloy model that helps us to generate executable compositions. Our goal was to demonstrate (a) that the approach is feasible, and (b) it can scale to implement fidelity-timeliness trade-offs in realistic time.

For our prototype, we used data about terrorist activities in Sudan, consisting of about 10 years of news reports and incidents. Our data repository consists of about 300Mb-400Mb of plain text reports for each year, which when processed, consists of networks with 379,638 nodes and 15,373,115 edges (with information about people, events, and locations). The fidelity choices (such as caching, reduction in scope, etc.) therefore have a major impact on the processing time of the compositions.

We ran our prototype on a 3 GHz Intel Core 5 machine with 4GB RAM. Even for a simple composition scenario, our approach allows fidelity variations that lead to execution times ranging between 2 mins 23 secs to more than 5 hours. The total execution time for Alloy model-generation varies between 1600 milliseconds and 2400 milliseconds, which is almost

TABLE II: Fidelity vs. Timeliness results

Fidelity Choices			Execution Time
Data	**Meta-networks**	**Algorithm**	
Current year	Agent-Agent only	Fast metrics only	0 hrs: 2 min: 23 secs
Current year	Agent-Agent only	All metrics	0 hrs: 3 min: 54 secs
Current year	All relationships	All metrics	0 hrs: 4 min: 14 secs
All year	Agent-Agent only	Fast metrics only	2 hrs: 26 min: 49 secs
All year	All relationships	All metrics	5 hrs: 46 min: 39 secs

negligible given the much larger execution time for the entire composition. Table II shows the common variation points and their impact on execution times.

IV. Related Work

There has been some significant work towards analysis of compositions. Examples include: QoS (Quality of Service) based analysis for service compositions [12], analysis of time constraints based on Petri Nets [24] and soundness checks [22] for BPEL orchestrations. Furthermore, tools such eflow [11] by Fabio Casati have addressed issues with dynamic service compositions. However, most of these analytic approaches are geared towards static design choices where there is limited support for trade-offs in the fidelity and timeliness space.

A related domain where such trade-offs have been relevant is product-line engineering. There has been some work towards automatic prediction of execution time based on feature composition by Siegmund et al. [14]. Another related work in the product-line compositions has been the Clafer tool [13] that extends Alloy-based simulations with multi-objective calculations to determine the best set of features in a product line. In our work we have addressed trade-offs between (various dimensions of) fidelity and timeliness and have automated it to be able to synthesize executable code. Similar to Bagheri and Sullivan [2], we use architecture as a basis for such reasoning and code-generation.

V. Conclusion and Future work

We demonstrated an architecture-based approach to make fidelity vs. timeliness trade-offs in a realistic domain. As a first step, we presented a simple scenario where we generate compositions based on performance profiles of components. In this case study we used a small number of metrics resulting significant, but relatively small option space. However, we are confident that more complex trade-offs can be handled by this model generation technique. As a future step, we plan to expand this to other domains and other dimensions of fidelity trade-offs.

A more complicated problem, and a possible future work, is to do the reverse i.e., calculating the fidelity options given the execution time requirements, which is similar to solving multi-objective problems [7]. Also, for this prototype we have not considered all possible model instances generated by Alloy, which may be evaluated for more optimal solutions.

Acknowledgment

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Further support for this work came from Army Research Office under Award No. W911NF1310155. The authors would like to thank Ed Morris, Soumya Simanta, Kathleen Carley, Troy Mattern and Jeff Boelng for their valuable contributions. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute. This material has been approved for public release and unlimited distribution. DM-0000706

References

[1] Alloy analyzer. http://alloy.mit.edu/alloy4/.

[2] H. Bagheri and K. J. Sullivan. Pol: specification-driven synthesis of architectural code frameworks for platform-based applications. In *GPCE*, pages 93–102, 2012.

[3] P. Bonacich. Factoring and weighting approaches to status scores and clique identification. *Jour. of Math. Sociology*, pages 113–120, 1972.

[4] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[5] K. M. Carley and J. Pfeffer. *Dynamic Network Analysis (DNA) and ORA*. Advances in Design for Cross-Cultural Activities (Part 2), D. D. Schmorrow, D.M. Nicholson (eds), CRC Press, 2012.

[6] K. M. Carley, J. Pfeffer, J. Reminga, J. Storrick, and D. Columbus. Ora users guide 2013, 2013.

[7] K. Deb. Multi-objective optimization. *Multi-objective optimization using evolutionary algorithms*, pages 13–46, 2001.

[8] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528 – 540, 2009.

[9] B.R. Schmerl et al. SORASCS: a case study in SOA-based platform design for socio-cultural analysis. In *ICSE*, pages 643–652. ACM, 2011.

[10] D. Garlan et al. Foundations and tools for end-user architecting. In *Monterey Workshop 2012*, LNCS, pages 157–182. Springer, 2012.

[11] F. Casati et al. Adaptive and dynamic service composition in *eflow*. In *Seminal Contr. to Inf. Systems Eng.*, pages 215–233. 2013.

[12] J. Cardoso et al. Quality of service for workflows and web service processes. *J. Web Sem.*, 1(3):281–308, 2004.

[13] M. Antkiewicz et al. Clafer tools for product line engineering. In *SPLC Workshops*, pages 130–135, 2013.

[14] N. Siegmund et al. Predicting performance via automated feature-interaction detection. In *ICSE*, pages 167–177, 2012.

[15] P.V. Elizondo et al. Resolving data mismatches in end-user compositions. In *IS-EUD*, pages 120–136, 2013.

[16] V. Dwivedi et al. An architectural approach to end user orchestrations. In *ECSA*, pages 370–378. Springer-Verlag, 2011.

[17] L. C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40:35–41, 1977.

[18] J.M. Graham, M. Schneider, and C. Gonzalez. Report social network analysis of unit of action battle laboratory simulations (cmu-sds-ddml-04-01). carnegie mellon university, social and decision sciences., 2004.

[19] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2012.

[20] J.S. Kim and D. Garlan. Analyzing architectural styles. *Journal of Systems and Software*, 83:1216–235, July 2010.

[21] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[22] F. Puhlmann and M. Weske. Interaction soundness for service orchestrations. In *ICSOC*, pages 302–313, 2006.

[23] J. Segal. Some problems of professional end user developers. In *VL/HCC*, pages 111–118, 2007.

[24] W. M. P. van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *BPM*, pages 161–183, 2000.

[25] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[26] D. J. Watts and S. H. Strogatz. Collective dynamics of'small-world'networks. *Nature*, 393(6684):409–10, 1998.