Computational Complexity of Selecting Components for Composition

Mikel D. Petty[1], *Eric W. Weisel*[2], *and Roland R. Mielke*[1]

[1]Virginia Modeling, Analysis, and Simulation Center
Old Dominion University, Norfolk VA 23529
mpetty@vmasc.odu.edu, rmielke@odu.edu

[2]WernerAnderson, Inc.
6655 Main Street, Gloucester VA 23061
eweisel@werneranderson.com

Keywords:
Composability, component selection, NP-completeness

ABSTRACT: Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements. The defining characteristic of composability is the ability to combine and recombine components. Composability exists in two forms, syntactic and semantic (also known as engineering and modeling). Syntactic composability is the implementation of components so that they can be connected. Semantic composability is the question of whether the models implemented in the composed components can be meaningfully composed, i.e., is their combined computation valid? A theory of semantic composability has been developed that examines the semantic composability of models using formal definitions and reasoning.

The computational complexity of the problem of selecting a set of components that meet a set of objectives is examined. In earlier work, Page and Opper defined four variants of this component selection problem based on two forms of objectives decidability (bounded and unbounded) and two forms of composition (emergent and non-emergent). They gave a proof that the bounded non-emergent variant of the component selection problem is NP-complete. In this paper an additional form of composition (anti-emergent) is defined, leading to two additional variants of the problem. Then a general form of the component selection problem that subsumes all six variants is defined. The general component selection problem is proven to be NP-complete even if the objectives met by a component or composition are known. Several related but different problems, including determining the objectives met by a component and determining the validity of a proposed composition are defined, and conjectures for their complexity are given.

1. Introduction

The ability to compose simulation systems from repositories of reusable components, i.e., composability, has lately been a highly sought after goal among modeling and simulation developers. The expected benefits of robust, general composability include reduced simulation development cost and time, increased validity and reliability of simulation results, and increased involvement of simulation users in the process. Consequently, composability is an active research area, with both software engineering and theoretical approaches being developed.

Composability, considered as a process, has a number of computational problems embedded within it that are important enough to warrant formal study. Here one of the most basic questions of composability is examined: given a set of components and a set of objectives, how difficult is it to select a subset of the components that meet the objectives? The computational complexity of the component selection problem is established, following and generalizing earlier work done on the problem by Page and Opper. Several other computational problems within composability are defined and their complexity is considered.

Following this introduction, the paper begins with background information on composability and NP-completeness theory. Then the component selection problem is defined formally, first in several variants, then in a general form. Next, the computational complexity of the general component selection problem is established. Following that, several other problems inherent in composability are defined. Finally, conclusions and ideas for future work conclude the paper.

2. Background

This section provides background information on composability, functions, and NP-completeness theory.

2.1 Composability

Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements [1]. The components to be composed may be drawn from a library or repository of components. That library might include multiple network interfaces, different user interfaces, a range of classes of implemented entity models, a variety of implemented physical models at different levels of fidelity, and so on. Different sets of components from the repository may be composed into different simulation systems. The components may be reused in multiple simulation systems. Indeed, to a certain extent reuse depends on composability [2].

The defining characteristic of composability is that different simulation systems can be composed in a variety of ways, each suited to some distinct purpose, and the different possible compositions will be usefully valid. Composability is more than just the ability to assemble simulations from parts; it is the ability to combine and recombine, to configure and reconfigure, sets of components into different simulation systems to meet different needs.

Composability exists in two forms, *syntactic* and *semantic*. Syntactic composability is the actual engineering implementation of composability. Syntactic composability is concerned with the compatibility of implementation details, such as parameter passing mechanisms, external data accesses, and timing mechanisms. It is the question of whether a set of components can be combined. Semantic composability, on the other hand, is the different question of whether the models that make up the composed simulation system can be composed and remain valid. Modeling issues such as domains of validity and consistent assumptions are central to semantic composability. It is the question of whether a composition of models is meaningful.

2.2 Functions

A *function* is a rule or mapping that relates elements of the function's domain (a set) to elements of its codomain (also a set), with the constraint that no element of the domain may be mapped to more than one element of the codomain [3]. One way to specify the mapping for a function is as a subset of the cross product (or Cartesian product) of the domain and the codomain. For example, suppose function f is defined from A to B, written $f: A \to B$. Then the mapping is a subset of $A \times B$, where $A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$, such that no two pairs (a, b) in the mapping of f have the same a. If a function is defined for all elements of its domain, i.e., if every $a \in A$ is mapped to some $b \in B$, it is a *total* function; otherwise, it is a *partial* function.

2.3 NP-completeness theory

NP-completeness theory is concerned with the computational complexity of decision problems [4]. A decision problem is defined in two parts. The first part is a formal specification of the information (such as sets, graphs, matrices, or numbers) that is the subject of the problem. The specification is given in precise yet general terms, for example, calling for "a graph of n vertices" rather than some specific graph. An instance of a decision problem is a specific set of information that complies with the information specification. The second part of a decision problem is a question, which can be answered "yes" or "no" (hence "decision problem"), about the properties of an instance. A solution to a decision problem is with respect to a specific instance; the solution is "yes" if the instance satisfies the question, "no" if it does not. For decision problem U, the set Y_U is the set of all instances of U for which the solution is "yes". For an instance I of problem U, $I \in Y_U$ if and only if the solution to instance I is "yes".[1]

In computational complexity theory, problems are categorized based on their upper bound on time $O(f(n))$, where n is the size of the instance. Those problems where $f(n)$ is a polynomial function on n (e.g., $f(n) = n^2$) on a deterministic computer belong to set P. Problems for which the time function $f(n)$ of the best known algorithm is an exponential function on n (e.g., $f(n) = 2^n$) belong to set NP. (Problems in NP can be solved in polynomial time on a nondeterministic computer.) Though it remains unproven, it is widely assumed that P $\neq$ NP.[2] NP-complete problems[3] are those problems in NP such that

[1] Problems of other types, such as search or optimization, can generally be shown to be no easier than their corresponding decision problems, so proofs about the difficulty of the decision problems apply to the other types as a lower bound. Surprisingly, problems of the seemingly more difficult types can also be shown in many cases to be no harder than their corresponding decision problems [11].

[2] Settling the question of whether P = NP has been called "the most important open question of either mathematics or computer science" [12].

[3] Because NP-complete is a set of problems it is perhaps more precise to say "problems in NP-complete" than "NP-complete problems". However, the latter formulation is ubiquitous and we follow the convention.

every instance of any NP-complete problem can be transformed into an equivalent instance of any other NP-complete problem by some process that requires $O(f(n))$ time, where $f(n)$ is polynomial on n.

Given a decision problem V, V can be shown to be NP-complete using a two-step procedure. The first shows that the problem is in NP. The second shows that it is at least as difficult as other NP-complete problems. The steps together imply that a problem is NP-complete.

1. Show that V is in NP, by giving a polynomial time algorithm to check a solution for V.
2. Transform a known NP-complete problem U to V, as follows:
   - 2a. Define a transformation function h from an instance I of U to an instance $h(I)$ of V.[4]
   - 2b. Show that h requires polynomial time in n, the size of I.
   - 2c. Show that $I \in Y_U$ if and only if $h(I) \in Y_V$.

Showing a problem to be NP complete has practical value. Once a problem is proven to be NP-complete it is known to be as hard as all other NP-complete problems. The failure, to date, to find a polynomial algorithm for any NP-complete problem suggests that finding one for the given problem may be problematic.

As mentioned earlier, to be NP-complete a decision problem must be in NP (step 1 in the basic proof procedure). NP-completeness theory may also provide information about decision problems that are not in NP. Suppose decision problem W is not in NP, but a known NP-complete problem U can be transformed into W so that a solution to W is a solution to U (step 2 in the basic proof procedure). Then W is at least as hard as the NP-complete problems, and is not solvable in polynomial time unless P = NP. Such problems are referred to as NP-hard [4].

NP-completeness theory includes the idea of oracles (also known as oracle Turing machines or oracle functions) [4]. An oracle is essentially a hypothetical or notional computational procedure that can perform an arbitrary computation in one computational or time step. Oracles can be used to study separately the computational complexity of problems that may have other computational problems embedded within them or connected to them. For example, if problem U has problem V embedded within it, then an oracle to solve V could be assumed and the complexity of U studied. This allows the determination of how difficult U might be even if the related problem V were solved.

[4] The transformation is more commonly known as f, rather than h [4]. We use the latter to avoid name conflict with the time function in the conventional order notation $O(f(n))$.

3. Component selection problem definitions

In this section the computational problem that is the focus of this paper, namely, component selection, is defined. Informally, component selection is the problem of selecting components to compose in order to meet a simulation's objectives. Somewhat less informally, component selection is the problem of selecting from a repository containing a given set of components a subset of those components to be composed such that the composition will meet a given set of objectives. In this section the problem is defined formally. In 1999, Page and Opper formally defined four variants of the component selection[5] problem based on two forms of objectives decidability and two forms of composition [5]. First, those definitions are examined and two additional variants of the problem, based on a third form of composability, are defined. Then a general form of the component selection problem that subsumes all six variants is defined.

Let $O = \{o_1, o_2, \ldots, o_n\}$ be a set of objectives.[6] Let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of components. A simulation system S is a subset of C, i.e., $S \subseteq C$. If $|S| > 1$ then S is a composition. Let $\circ$ denote composition of components, e.g., $(c_j \circ c_k)$ is a composition of two components. Let $\Rightarrow$ denote satisfying an objective, i.e., $c_j \Rightarrow o_i$ means component c_j satisfies objective o_i, and c_j *not*$\Rightarrow$ o_i means that it does not.[7] The $\Rightarrow$ and *not*$\Rightarrow$ operators may also be applied to compositions and sets of objectives, e.g., $(c_j \circ c_k) \Rightarrow o_i$ and S *not*$\Rightarrow$ O have the expected meanings. A simulation system $S \Rightarrow O$ if and only if $S \Rightarrow o_i$ for every $o_i \in O$.

In their definitions, Page and Opper first considered the decidability of objectives, defining two forms: *bounded* and *unbounded* [5]. If, for every objective o_i in a given set of objectives O, it is possible to decide in polynomial time that o_i is satisfied, then O is bounded.

[5] They actually referred to the problem as "composability", rather than "component selection". Because there are other computational problems inherent in composability, we use the latter term.

[6] The notation introduced here follows that of Page and Opper [5]; for the most part it is the same, but there have been some changes and additions.

[7] The symbol "$\Rightarrow$" normally means "logically implies", as in $A \Rightarrow B$. To avoid confusion, we will not use it in that sense in this paper.

Decidability of objective	**Page and Opper [5]**	**Alternate**
Decidable in polynomial time	Bounded	Decidable
Decidable but not in polynomial time	Unbounded	Decidable
Undecidable	Unbounded	Undecidable

Table 1. Alternate classes of objective decidability.

If O contains at least one objective o_i for which polynomial time is not sufficient to decide if o_i is satisfied, then O is unbounded. The question of whether a component halts is undecidable [5] [6] and so is an example of the latter case. Note that the decidability of the objectives is defined independently of any particular component or composition.

These definitions are helpful and draw attention to the potential difficulty of deciding if an objective is satisfied. Two comments regarding them should be noted. First, as given the definitions refer to deciding in polynomial time if an objective is satisfied, but they do not state explicitly which problem parameter the decision time must be polynomial in; should it be polynomial in the length of the objective's encoding, in the size of the objective set, or something else? The former is assumed. Second, it is possible to partition the objectives into three decidability classes, instead of two: decidable in polynomial time, decidable but not in polynomial time, and undecidable. Page and Opper group the second and third of these three into the "unbounded" class in their definitions; an alternate classification would be to group the first and second into "decidable", leaving the third as "undecidable". These classifications are summarized in Table 1. Though the alternate grouping has some uses, for the remainder of this paper the Page and Opper terms and definitions for objectives decidability are retained.

Page and Opper second considered forms of composition, also defining two forms: *emergent* and *non-emergent* [5]. The idea is that the objectives met by a composition of components may not necessary be simply the union of the objectives met by the components individually. Suppose $c_j, c_k \in C$ are components and $o_i \in O$ is an objective. If c_j *not*$\Rightarrow$ o_i and c_k *not*$\Rightarrow$ o_i and $(c_j \circ c_k)$ *not*$\Rightarrow$ o_i, then the composition if non-emergent. If c_j *not*$\Rightarrow$ o_i and c_k *not*$\Rightarrow$ o_i but $(c_j \circ c_k) \Rightarrow o_i$, then the composition is emergent. In the latter case, the components combine to satisfy some objective that neither satisfies separately.

Based on their two forms of objectives decidability (bounded and unbounded) and two forms of composition (emergent and non-emergent), Page and Opper defined four variants of the component selection problem: bounded emergent, unbounded emergent, bounded non-emergent, and unbounded non-emergent [5].

We define an additional form of composition: *anti-emergent*. If $c_j \Rightarrow o_i$ or $c_k \Rightarrow o_i$ but $(c_j \circ c_k)$ *not*$\Rightarrow$ o_i, then the composition is anti-emergent. The idea is that the components in a composition could interfere with each other in such a way that the composition fails to satisfy objectives that the components satisfy separately. It is easy to imagine examples of anti-emergence; a terrain database component and an intervisibility determination component may separately satisfy objectives, but if they are based on different terrain database formats, their composition will likely be anti-emergent.

The three possible combinations already listed do not exhaust the possibilities. Table 2 lists all eight possible logical combinations of c_j $\Rightarrow$/*not*$\Rightarrow$ o_i, c_k $\Rightarrow$/*not*$\Rightarrow$ o_i, and $(c_j \circ c_k)$ $\Rightarrow$/*not*$\Rightarrow$ o_i. (Those due to Page and Opper are so noted.) The combinations all fall into one of the three composition forms defined (non-emergent, emergent, anti-emergent).

The anti-emergent form of composition adds two new variants of the component selection problem to the four defined earlier: bounded anti-emergent and unbounded anti-emergent. The computational complexity of each problem variant could be studied separately. Indeed, Page and Opper considered one problem variant, giving a proof that the bounded non-emergent variant of the component selection problem is NP-complete [5]. However, we prefer to separate the problem of determining which objectives a component or composition satisfies from the problem of selecting a set of components to meet the objectives; we will do so by assuming an oracle for the former problem. We also prefer to consider the component selection problem in general, rather than in terms of variants.

$c_j \Rightarrow o_i$	$c_k \Rightarrow o_i$	$(c_j \circ c_k) \Rightarrow o_i$	Non-emergent
c_j *not*$\Rightarrow o_i$	$c_k \Rightarrow o_i$	$(c_j \circ c_k) \Rightarrow o_i$	Non-emergent
$c_j \Rightarrow o_i$	c_k *not*$\Rightarrow o_i$	$(c_j \circ c_k) \Rightarrow o_i$	Non-emergent
c_j *not*$\Rightarrow o_i$	c_k *not*$\Rightarrow o_i$	$(c_j \circ c_k) \Rightarrow o_i$	Emergent [5]
$c_j \Rightarrow o_i$	$c_k \Rightarrow o_i$	$(c_j \circ c_k)$ *not*$\Rightarrow o_i$	Anti-emergent
c_j *not*$\Rightarrow o_i$	$c_k \Rightarrow o_i$	$(c_j \circ c_k)$ *not*$\Rightarrow o_i$	Anti-emergent
$c_j \Rightarrow o_i$	c_k *not*$\Rightarrow o_i$	$(c_j \circ c_k)$ *not*$\Rightarrow o_i$	Anti-emergent
c_j *not*$\Rightarrow o_i$	c_k *not*$\Rightarrow o_i$	$(c_j \circ c_k)$ *not*$\Rightarrow o_i$	Non-emergent [5]

Table 2. Forms of composition.

The general component selection problem is defined as follows, following the problem definition conventions of NP-completeness theory [4]:

COMPONENT SELECTION
INSTANCE: Set $C = \{c_1, c_2, \ldots, c_m\}$ of components, set $O = \{o_1, o_2, \ldots, o_n\}$ of objectives, oracle function σ: *power*$(C) \rightarrow$ *power*(O), positive integer $K \leq |C|$.
QUESTION: Does C contain a composition S that satisfies O of size K or less, i.e., a subset $S \subseteq C$ with $|S| \leq K$ such that $O \subseteq \sigma(S)$?

In this problem definition, the purpose of C as the set of components and O as the set of objectives in the instance is straightforward. Oracle function σ determines what objectives any component or composition of components satisfies. To understand the definition of σ, recall that the power set of a set is the set of all its subsets, so a function from the power set of C (written *power*(C)) to the power set of O (written *power*(O)) can specify for any subset of C, i.e., any component or composition of components, what subset of objectives from among O that subset of C satisfies. Because σ is an oracle function, it can be assumed to answer the question in one step, allowing examination of the difficulty of component selection independent of the difficulty of objectives decidability. Integer K in the instance is the maximum size of the composition allowed; the limit is required to make this a decision problem. The deceptively similar-sounding problem "What is the smallest subset of C that satisfies O?" is an optimization problem and not within the bounds of NP-completeness theory. However, note that the decision problem subsumes the also similar-sounding existence problem; if the question is "Does there exist a subset of C that satisfies O?" then setting K to $|C|$ expresses that existence problem as a decision problem.

Finally, note that the condition $O \subseteq \sigma(S)$ in the question could equivalently have been written using the notation introduced earlier as $S \Rightarrow O$; the former form was chosen to have the problem definition use only standard set notation.

The important thing to observe about this problem is that it subsumes all six variants of the component selection problem defined earlier. The emergent, non-emergent, and anti-emergent forms of composability are covered by specifying different mappings for function σ in the instance. The bounded and unbounded forms of objectives decidability are covered by making σ an oracle function.

4. Computational complexity of component selection

In this section the computational complexity of the general component selection problem, defined in the previous section, is established. As mentioned, Page and Opper gave a proof that that one variant of the component selection problem is NP-complete [5]. It would be reasonable to assume that the general problem, which includes the variants, is also NP-complete (or NP-hard). As expected, it will be shown that the general problem, even given the oracle function to determine the objectives satisfied by a composition, remains NP-complete.

The proof of NP-completeness will use the MINIMUM COVER (MC) problem, known to be NP-complete [4] [7]. That problem is defined as follows:

MINIMUM COVER[8]
INSTANCE: Collection C of subsets of a finite set O, positive integer $K \leq |C|$.
QUESTION: Does C contain a cover for O of size K or less, i.e., a subset $C' \subseteq C$ with $|C'| \leq K$ such that every element of O belongs to at least member of C'?

The definition of COMPONENT SELECTION (CS) is repeated here for convenience:

COMPONENT SELECTION
INSTANCE: Set $C = \{c_1, c_2, \ldots, c_m\}$ of components, set $O = \{o_1, o_2, \ldots, o_n\}$ of objectives, oracle function σ: $power(C) \rightarrow power(O)$, positive integer $K \leq |C|$.
QUESTION: Does C contain a composition S that satisfies O of size K or less, i.e., a subset $S \subseteq C$ with $|S| \leq K$ such that $O \subseteq \sigma(S)$?

We now prove the main result.

Theorem. COMPONENT SELECTION is NP-complete.

Proof. By transformation from MINIMUM COVER. First it must be shown that CS is in NP. Given a subset S of C, determining if $O \subseteq \sigma(S)$ can be done by searching in $\sigma(S)$ for each element of O. A naïve algorithm to do so requires $O(mn)$ time (recall that computing $\sigma(S)$ requires only one step), which is clearly polynomial in the length of the instance, thus CS is in NP.

The transformation function h from any instance I of MC to an instance $h(I)$ of CS is now defined. Because the instances of MC and CS have like-named items, subscripts will be used where needed to distinguish them, e.g., C_{MC} is the set C from instance I of MC, whereas C_{CS} is the set C from the instance $h(I)$ of CS. The function σ in $h(I)$ will be defined by generating its mapping as ordered pairs. The transformation h is as follows:

1. For every $c_i = \{o_{i,1}, o_{i,2}, \ldots\} \in C_{\text{MC}}$, generate $c_i \in C_{\text{CS}}$, where the elements of C_{MC} are sets and those of C_{CS} are simply elements.
2. For every $c_i = \{o_{i,1}, o_{i,2}, \ldots\} \in C_{\text{MC}}$, generate $(c_i, \{o_{i,1}, o_{i,2}, \ldots\}) \in \sigma$, where first element of each ordered pair is an element of C_{CS} and the second element the corresponding element of C_{MC}, i.e., a subset of O_{MC}.
3. Copy O_{MC} to O_{CS}.
4. Copy K_{MC} to K_{CS}.

Step 1 requires time $\in O(m)$, step two $\in O(mn)$, step 3 $\in O(n)$, and step 4 $\in O(1)$, so the overall time complexity of transformation h is $\in O(mn)$, which is polynomial in the length of the input.

It must now be shown that $I \in Y_{\text{MC}}$ if and only if $h(I) \in Y_{\text{CS}}$.

Only if. Assume $I \in Y_{\text{MC}}$. Then there exists subset $C' \subseteq C_{\text{MC}}$ such that $|C'| \leq K_{\text{MC}}$ and $O_{\text{MC}} \subseteq \bigcup_{c \in C'} c$.

Let $S = \{ c_i \in C_{\text{CS}} \mid c_i \in C' \}$. Then $\sigma(S) = \bigcup_{c \in C'} \sigma(c) = \bigcup_{c \in C'} c$ by h. Because $O_{\text{CS}} = O_{\text{MC}}$ by h and $O_{\text{MC}} \subseteq \bigcup_{c \in C'} c = \sigma(S)$, then $O_{\text{CS}} \subseteq \sigma(S)$. Because $K_{\text{CS}} = K_{\text{MC}}$ by h and $|S| = |C'| \leq K_{\text{MC}}$, then $|S| \leq K_{\text{CS}}$. Therefore $h(I)$ in Y_{CS}.

If. Assume $h(I)$ in Y_{CS}. Then there exists subset $S \subseteq C_{\text{CS}}$ such that $|S| \leq K_{\text{CS}}$ and $O_{\text{CS}} \subseteq \sigma(S)$. Let $C' = \{ \sigma(c_i) \mid c_i \in S \}$. Then $\bigcup_{c \in C'} c = \bigcup_{c \in S} \sigma(c) = \sigma(S)$ by h. Because $O_{\text{MC}} = O_{\text{CS}}$ by h and $O_{\text{CS}} \subseteq \sigma(S) = \bigcup_{c \in C'} c$, then $O_{\text{MC}} \subseteq \bigcup_{c \in C'} c$. Because $K_{\text{MC}} = K_{\text{CS}}$ by h and $|C'| = |S| \leq K_{\text{CS}}$, then $|C'| \leq K_{\text{MC}}$. Therefore I in Y_{MC}.

Thus $I \in Y_{\text{MC}}$ if and only if $h(I)$ in Y_{CS}, and therefore CS is NP-complete. ■

This result shows that general component selection problem is NP-complete even if the objectives met by each component and composition are known. If the objectives met by the components and composition are not known and must be determined as part of assembling a simulation system by composition, then the combined problem is at least as hard component selection, i.e., the combined problem of component selection and objectives determination is NP-hard.

Figure 1 summarizes the complexity results of this paper. It shows the six variants of the component selection problem previously defined, the general component selection problem, and their computational complexities.

[8] In [4], the definition for MINIMUM COVER uses S for the set to be covered, rather than O. Because S is already used for the subset in COMPONENT SELECTION and the set to be covered in MINIMUM COVER corresponds to O, not S, in that problem, using O here makes the notation of the proof less confusing.

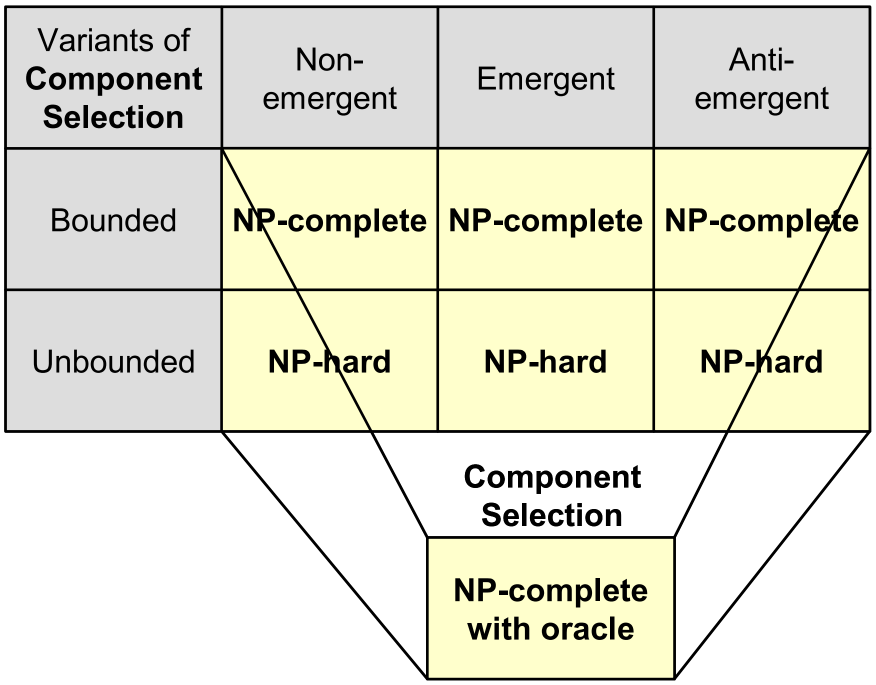

Figure 1. Component selection problems and complexities.

5. Other computational problems in composability

Component selection is not the only computational problem inherent in composability. In this section several related but different problems are informally defined and conjectures for their complexity are discussed.

Component objectives. What objectives are satisfied by a component? This problem has already been mentioned in previous sections. Knowing the objectives met by a component is a necessary precursor to solving the component selection problem. To determine what objectives a component satisfies the component itself must be analyzed; that might be done by processing the component as programming language source code or by processing a meta-model specification of the component's semantics. Unfortunately, a procedure to determine in general which objectives a component satisfies may not be in NP or even decidable, as observed by Page and Opper [5], because one objective could be that the component halt for arbitrary input and the halting problem is undecidable [3] [6]. Even the simpler-seeming problem of determining if a given query objective is satisfied by a component may be likewise undecidable for the same reason. However, it will be necessary to make determinations of the objectives satisfied by components for composability to be a practical reality. Such determinations will either have to be made by non-algorithmic methods and/or only for decidable objectives.

Composition objectives. What objectives are satisfied by a composition? This problem was also mentioned previously, but was conflated with the component objectives problem. In fact, it may be separable. If the objectives met by each component in a composition are already known, and the form of composition is known, then that information could be used as the basis for an algorithm to determine what objectives are satisfied by a composition, or if a given query objective is satisfied by a composition. We conjecture that this question, given the information mentioned, is decidable.

Validity determination. Is a component valid? In related work a formal definition for validity has been developed; there validity is defined as a special case of quantifiable consistency between models, where one of the models is defined as "perfect" and the other may or may not be valid with respect to it [8]. Though the theory applies to models, not components, the validity determination may be applicable to components that implement a model. Using that definition, we conjecture that determining the validity of a model is decidable but NP-complete or NP-hard.

Composition complexity. Given the computational complexities (time and/or space) of the components, what is the computational complexity of their composition? We conjecture that this problem is not only decidable, but in P, solvable by analysis of a formal specification of the component, such as programming language source code or formal semantic meta-model. However, a practical algorithm to make the determination could be difficult to develop.

6. Summary and conclusions

The computational complexity of the component selection problem was examined. Page and Opper defined four variants of the problem and gave a proof that the one variant is NP-complete. Two additional variants of the problem were defined. Then a general form of the component selection problem that subsumes all six variants was given. The general component selection problem was proven to be NP-complete even if the objectives met by a component or composition are available. Several other computational problems inherent in composability, including component objectives, composition objectives, validity determination, and composition complexity were informally defined.

The main result, that component selection is NP-complete even if the objectives satisfied by a component or composition are available, suggests the underlying computational complexity of composability. The component selection problem is conceptually simple compared to issues of actually implementing composability in either syntactic or semantic forms.

Of course, an NP-complete problem is not unsolvable; rather, if a problem is NP-complete there is no efficient (polynomial time) algorithm to solve it, unless P = NP. However, establishing a problem as NP-complete does not free developers from the need to solve it in practical applications. NP-complete problems are regularly solved by algorithms that are not efficient but have run-times that can be tolerated, or by heuristics that are not guaranteed to find an optimum solution but give a reasonable approximation for a useful range of cases. Clearly, if composability is to become a practical reality, the component selection problem will have to dealt with in some way.

7. Related and future work

Though composability is widely understood at the conceptual level, many different specific meanings and levels have been associated with the term; those levels have been clarified, composability has been distinguished from related ideas such as interoperability, and composability research has been briefly surveyed [1]. Formal definitions for model, simulation, and validity have been given as the basis for a theory of semantic composability [9], and some of the definitions have been updated and made more general, especially validity [10]. Classes of models and validity relations are defined and the question of whether validity is preserved under composition for those classes has been studied [8].

One line of future work that follows directly from the results presented here is to study the computational complexity of the computational problems in composability other than component selection, i.e., the problems identified earlier. We also anticipate working towards the development of formal semantic meta-models and algorithms to process them for validity determination.

8. Acknowledgements

The Office of Naval Research and the Defense Modeling and Simulation Office sponsored this research under contract N00014-03-1-0089. That support is gratefully acknowledged.

9. References

[1] M. D. Petty and E. W. Weisel, "A Composability Lexicon", *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Orlando FL, March 30-April 4 2003, pp. 181-187.

[2] J. Igarza and C. Sautereau, "Distribution, Use and Reuse: Questioning the Cost Effectiveness of Re-using Simulations With and Without HLA", *Proceedings of the Fall 2001 Simulation Interoperability Workshop*, Orlando FL, September 9-14 2001.

[3] J. L. Hein, *Discrete Structures, Logic, and Computability, Second Edition*, Jones and Bartlett, Sudbury MA, 2002.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, New York NY, 1979.

[5] E. H. Page and J. M. Opper, "Observations on the Complexity of Composable Simulation", *Proceedings of the 1999 Winter Simulation Conference*, Phoenix AZ, December 5-8 1999, pp. 553-560.

[6] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungs-problem", *Proceedings of the London Mathematical Society*, Ser. 2, Vol. 42, 1936, pp. 230-265; Correction, *ibid.*, Vol. 43, pp. 544-546, 1937.

[7] R. M. Karp, "Reducibility among combinatorial problems", in R. E. Miller and J. W. Thatcher (Editors), *Complexity of Computer Computations*, Plenum Press, New York NY, pp. 85-103, 1972.

[8] E. W. Weisel, R. R. Mielke, and M. D. Petty, "Validity of Models and Classes of Models in Semantic Composability", *Proceedings of the Fall 2003 Simulation Interoperability Workshop*, Orlando FL, September 14-19 2003.

[9] M. D. Petty and E. W. Weisel, "A Formal Basis for a Theory of Semantic Composability", *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Orlando FL, March 30-April 4 2003, pp. 416-423.

[10] M. D. Petty, Eric W. Weisel, and Roland R. Mielke, "A Formal Approach to Composability", *Proceedings of the 2003 Interservice/Industry Training, Simulation, and Education Conference*, Orlando FL, December 1-4 2003.

[11] F. P. Preparata and M. I. Shamos, *Computational Geometry, An Introduction*, Springer-Verlag, New York NY, 1988.

[12] S. Homer and A. L. Selman, *Computability and Complexity Theory*, Springer-Verlag, New York NY, 2001.

10. Author biographies

Mikel D. Petty is Chief Scientist of the Virginia Modeling, Analysis & Simulation Center at Old Dominion University. He received a Ph.D. in Computer Science from the University of Central Florida. Dr. Petty has worked in modeling and simulation research and development since 1990 in the areas of simulation interoperability, computer generated forces, multi-resolution simulation, and applications of theory to simulation. He previously worked for the Institute for Simulation and Training at the University of Central Florida. He has served as a member of a National Research Council committee on modeling and simulation and is currently an editor of the journal *SIMULATION: Transactions of the Society for Modeling and Simulation International*.

Eric W. Weisel is a Project Scientist at the Virginia Modeling, Analysis & Simulation Center at Old Dominion University. He received a M.S. in Operations Research from the Florida Institute of Technology in 1995 and a B.S. in Mathematics from the United States Naval Academy in 1988. He is currently a Ph.D. candidate in Modeling and Simulation at Old Dominion University. His research interests include simulation formalism and composability. He is also President of WernerAnderson, Inc., a private veteran-owned small business that performs research and development in the areas of defense-related operations research and simulation. He previously served as a U.S. Navy submarine officer with experience in nuclear engineering, operations management, navigation, and joint operations.

Roland R. Mielke earned B.S., M.S., and Ph.D. degrees, all in Electrical Engineering, from the University of Wisconsin – Madison. Since 1975 he has been on the faculty of Old Dominion University where he is currently Professor of Electrical and Computer Engineering and holds the designation University Professor. Dr. Mielke also serves as Technical Director for the Virginia Modeling, Analysis and Simulation Center. His research interests are in the areas of systems theory and simulation and include simulation methodologies, system modeling, composability theory, discrete event simulation, and the application of simulation to the development of enterprise decision support tools.