Component Technologies and Fundamental Research in Interoperability

Michael W. Mislove[1]
Tulane University

Joy N. Reed[1]
Armstrong Atlantic State University

White Paper submitted to
Workshop for New Visions for Software Design and Productivity:
Research and Applications

Abstract

The complexity of the software infrastructure of the information age is a significant issue affecting the security and economic viability of the U.S. Despite being of fundamental importance to the proper functioning of so much of our everyday life, these complex systems often exhibit bugs that cause system crashes and vulnerabilities to cyber attacks, such as Internet viruses and worms. As ever more complex systems are adopted for use in safety critical applications, it is important to seek methodologies that can assure these systems function as intended, and are invulnerable to interference from hackers. Formal methods have long been viewed as a promising approach to providing such technologies, but the complexities of industrial-strength systems have proven too much for the homogeneous approach most often associated with formal methods analysis. In this white paper we outline an alternative, heterogeneous approach which seeks to apply formal methods in a component-wise fashion that combines varying approaches. We motivate this approach with an example that includes a formal analysis of an event-based system, combined with a similar analysis for a state-based system. Our goal is to devise a wide range of component methods that can be applied where appropriate, and combined under the umbrella an overall formal analysis system.

1 Introduction

Complex computer systems pervade every facet of modern life: they form the underlying infrastructure for our economy, they are the backbone of the US defense infrastructure, and even our communication networks are essentially computer systems. Despite being of fundamental importance to the proper functioning of so much of our everyday life, these complex systems often exhibit bugs that cause system crashes. In addition, unexpected vulnerabilities to cyber attacks – including Internet viruses and worms – are a fact of life. But do they have to be? As ever more complex systems are adopted for use in safety critical applications, from embedded systems to automated control units, it is important to seek technologies that provide methods to assure these systems function as intended and are invulnerable to interference from hackers.

[1]Work supported by the US Office of Naval Research.

The fragility of software systems is well-known, and it is due in large part to the complexity of systems, and the *ad hoc* methods with which they are developed. Increasingly software systems are designed and implemented as network-centric distributed systems, made up of components which must interoperate in a desirable and well-understood manner. System development often requires integrating legacy components with newly designed ones. Open systems philosophy is an accepted mode of development, bringing with it a greater need for effective interface specifications which support independently developed and adapted components. Middleware addresses the integration of components by defining interface standards that allow diverse components to interoperate, but in itself does not address the problem of achieving reliable and secure interoperation. While these approaches all point to a need to interoperate and to support independent development, there are few tools available that support analysis of the various components that can arise by employing these approaches – tools that provide assurance that the individual components meet their required specifications.

Providing methods for assuring components meet their specifications has been the traditional province of formal methods, and, more specifically, semantics. Early semantics work from the 60s and 70s has had a lasting and beneficial effect on classical program development. Structured programming, invariant relationships, OO paradigms, separation of specification and implementation – all now accepted as best-practice methods have their origins in fundamental semantics research. This approach tends to view a system as composed of homogeneous components that are largely similar to one another, and, in particular, that are amenable to the same analysis tools.

Component technologies, on the other hand, are heterogeneous in nature, with various components of the overall system differing markedly from one another. Further semantics research is required to develop a sound semantics framework for supporting component technologies. Such a framework should specifically address integration of different techniques and formalisms, including combining well-understood formal methods with informal best-practices. The aims of the research are to identify and elucidate fundamental semantics issues of development methods, tools, and domain specific interfaces and programming languages for component object interoperability. We now outline an approach we are developing that addresses these issues.

Our approach is to take a relational view of components, with one serving as a potential plug-in to another. Obligations are placed on both so that collectively they achieve the desired behavior. Existing composition and integration methods often preserve safety properties, e.g., restricting access to authorized users, but liveness properties such as freedom from deadlock are significantly more problematic. We have identified specific behavioral, machine-checkable paradigms for client-server models whereby components can be developed independently, in such a way that desired liveness properties are preserved under separate refinement. Our objective is to generalize the semantic context of these paradigms, so that we can derive fundamental relationships of disciplined design and interface specification for distributed components. Essential to our aims is the ability to use heterogeneous techniques when appropriate. For example, consider a subscription-based data base. State-based techniques such as Action Systems or UML are well-suited for specifying functional properties, while process algebraic techniques such as CSP have proved extremely useful for specifying and analyzing security concerns.

2 Component Technologies: Fundamental Research in Interoperability

One approach to managing the complexity of a large system is to break the system into modules, an approach that also supports separation of concerns. If formal methods are to be applied in the process, a number of options are available. A unified notation may be applied throughout, but possibly at the expense of prohibitive complexity and the lack of optimality for individual parts of the system. Alternatively, specifying different system aspects in different notations is attractive, particularly for greater flexibility in incorporating COTS components which ideally come with some guarantee of their behavior. In a distributed system it can be the case that a transaction or service requires the interoperation of a chain of components and services collectively intended to achieve a desired result. Various components may be selected as off-the-shelf products to plug into a particular application. The correct operation of the application depends not only on the integrity of its own functions, but also on its interactions with other, separately developed components.

Our approach is to develop a framework in which components of a system can be specified and developed independently, with constraints on their interfaces which ensure that the specification of each is satisfied. The goal is to devise appropriate languages and semantic models to derive paradigms and best-practice techniques for component technologies, and to understand and demonstrate the efficacy of these methods.

Concurrency and refinement Our interest is in concurrent systems, which are composed of several component processes. We describe a framework in which components of a concurrent system can be specified and developed independently, with constraints on their interfaces ensuring that the specification of each is satisfied. One of the fundamental problems in concurrent systems is avoiding *deadlock*, a situation in which no component can make progress. Our notion of one component serving as a plug-in to another is that the plug-in component does not increase the possibility of deadlock of the overall system. This view has the advantage of offering a mechanism to specify minimal interface properties required of a plug-in component, for example, for a server providing specified services to a client. However, in general such relational properties lack *stability*, by which we mean they are preserved under refinement. Refinement is understood as a semantic way of assuring that a specification for a given application can be maintained through further development: a process P *refines* a process Q if all of the behaviors of P meet the specification for Q. If relationally defined properties lack stability, this accepted notion of formal refinement is fatally undermined: originally specified components might themselves always interact in a desirable fashion, but their refinements are not guaranteed to do so. An answer lies in identifying stronger semantic conditions which ensure compatibility of interface specifications, guaranteeing not only that components interoperate satisfactorily, but also that they are stable. We have formulated such conditions in machine-checkable CSP[1]. However, the issues transcend specific notations, and the goal is to generalize these results in order to integrate heterogeneous specification techniques.

CSP and Specification Process algebras such as CSP provide a method for defining and analyzing concurrent processes. CSP has many applications, among them specification and verification

[1]CSP is a process algebra developed at the University of Oxford, and which also has a model-checker, FDR that supports automated analysis of finite (and some infinite) CSP processes.

of systems composed of concurrently running processes. As with other process algebras, CSP is defined by its *syntax* and its *semantics* The former includes the usual operations of concurrency – sequential and parallel composition, nondeterministic choice and recursion – as well as some individual to CSP, most notably deterministic choice and the hiding operator. The latter is built upon a notion of *behavior* of a process. This is described in terms of the atomic actions – each representing a communication event between processes – a process can or cannot participate in. A *failure* of a process P is a pair (s, X) where s is a finite string of atomic actions P can participate in, and X is a set of atomic actions P could refuse after participating in the string s. Similarly, a *divergence* of P is a string s of actions that P can participate in and after which P could *diverge* – engage in infinitely many internal actions and never respond to its environment. There are a number of healthiness conditions that must be satisfied by a set $A = (F, D)$ of failures-divergence pairs for A to represent a process.[2] Using the *failures-divergences* $(F(P), D(P))$ of a process P, we can define a partial order on processes:

$$Q \sqsubseteq P \qquad \Leftrightarrow \qquad F(P) \subseteq F(Q) \; \& \; D(P) \subseteq D(Q).$$

This means that every failure of P is also a failure of Q, and every divergence of P is also a divergence of Q. This partial order on processes is sometimes called the *order of nondeterminism*, since clearly Q is more nondeterministic than P. In case that $Q \sqsubseteq P$ we also say P *refines* Q.

CSP also allows specifications to be encoded as failures and divergences, so we can view the lower process Q in the relation $Q \sqsubseteq P$ as a *specification* for the process P. Thus, P *satisfies* Q iff $Q \sqsubseteq P$. This effectively reduces checking that a process meets its specification to checking a set containment. The problem identified above – under what conditions can a refining component can be substituted for another one in a complex, heterogeneous system? – is an issue of fundamental importance for system development and ongoing operation.

Plug-in components A concurrent system consists of components running in parallel, and in such a system, we may wish to replace one component P with another, P'. An obvious assumption is that $P \sqsubseteq P'$, but it turns out that this is not enough. Indeed, we *might* think that, assuming that $P \sqsubseteq P'$ and $Q \sqsubseteq Q'$, then it would follow that $P \| Q \sqsubseteq P' \| Q'$. The problem is that reasoning does not take proper account of deadlock: there are simple examples (cf. [2]) where $P' \| Q'$ can deadlock when both components are deadlock free.

These ideas are explored in [2], and the result is the notion of a *plug-in* component. Basically, P' is a *plug-in* for P if P' does not allow any deadlock in a parallel composition $P' \| Q$ that is not already present in $P \| Q$. From the above remarks, this is a stronger relation than refinement, and in fact, the principle motivation for [2] was to find conditions when $P' \| Q'$ would continue to satisfy a specification that was satisfied by $P \| Q$, assuming that $P \sqsubseteq P'$ and $Q \sqsubseteq Q'$. The requirement that was found in [2] was that of *plug-in*, which requires not only that P' refine P, but also that P' be as live as P with respect to Q. The precise definition of this last involves the failures and divergences of P, P' and Q, and hence is tied closely to the failures-divergences model for CSP.

Order theoretic ideas The notion of a plug-in developed in [2] provides an example of how one can reason about refining a parallel composition by refining each component, but a more general theory also is needed. In fact, the semantics of CSP, and of many process algebras like it, are given

[2]For a comprehensive presentation, see [3].

by an order theory, such as the order of nondeterminism given above. There are standard results from order theory that need to be investigated in this setting. Among these are the notions of *Galois adjunctions* (cf. [1]) which generalize and make precise such notions as the refinement order. Galois adjunctions also provide an approach to characterizing the *least* plug-in for a component P of a parallel composition, Since the structure of the failures-divergences model is given by the order of nondeterminism, the goal is to use Galois adjunctions to describe precisely the plug-ins for a component P in a parallel composition $P \| Q$.

Dually, Galois adjunctions also hold the promise for deriving results going the other way: how coarse can $P \sqsubseteq P'$ be so that $P \| Q$ satisfies a specification which already is satisfied by $P' \| Q$? The process of software development is cyclic in nature – devising components, then revising them to suit further needs. In such a process, one needs as much flexibility as possible, and having methods that allow one to move both ways – up or down with respect to refinement – clearly could be useful.

3 Summary

We have described work in [2] which addresses issues arising in a process algebraic setting where one would like to replace a component of a concurrent system with another component, while still preserving the specification of the overall system. This problem is harder than simple refinement, because of the possibility of introducing deadlock into the system, and the result is the notion of a plug-in for a component of a concurrent system. The arena in which this work is carried out is basically order theoretic, and we have described potential applications of standard order theoretic ideas that may prove useful in providing a methodology for reasoning about plug-ins, and coarser processes that can be substituted into parallel systems without introducing deadlocks.

The work in [2] also considers Action Systems, which are state based, and how they relate to the process algebraic setting of CSP. The arguments are still order theoretic, and the same ideas that can be applied in the process algebraic setting also can be applied there. Thus, our overall approach will include both state-based and event-based systems, which will provide a richer environment for development than either setting alone can support.

Finally, the presentation at the workshop will include specific examples from areas such as security which space limitations prevent us from describing here.

References

[1] Gierz, G., *et al*, "A Compendium of Continuous Lattices," Springer-Verlag, 1980.

[2] Reed, J. N. and J. Sinclair, *Refinement preserving lug-in components,* preprint, 19pp.

[3] Roscoe, A. W., "The Theory and Practice of Concurrency," Prentice Hall, 2000.