Observable Behavior of Distributed Systems:
Component Reasoning for Concurrent Objects

Crystal Chang Din, Johan Dovland, Einar Broch Johnsen, and Olaf Owe
Department of Informatics, University of Oslo, Norway
{crystald, johand, einarj, olaf}@ifi.uio.no

Distributed systems play an essential role in the modern world, and the quality of such systems is crucial. Quality assurance is however non-trivial since distributed systems depend on unpredictable factors such as different processing speeds of independent components and network transmission speeds. Such systems are therefore hard to test under the different conditions, which calls for the need for precise modeling and analysis frameworks with suitable tool support. In particular, there is a need for compositional methods such that each component can be analyzed independently from its surrounding components.

Object orientation is the leading framework for concurrent and distributed systems, recommended by the RM-ODP [10]. Many distributed systems are today programmed in object-oriented, imperative languages like Java and C++. Programs written in these languages are in general difficult to analyze due to composition and alias problems, and the complexity of their concurrency, communication, and synchronization mechanisms. Rather than performing analysis at the level of the Java and C++ code, it may be easier to consider a model of the program at a suitable level. In this paper, we consider a more general OO language, ABS, which is inspired by the Creol language [11]. ABS provides a concurrent and communication model that is suitable for loosely coupled objects in the distributed setting. This language is imperative but avoids some of the mentioned difficulties of analyzing distributed systems.

In ABS, there is no access to the internal state variables of other objects, and a concurrent object has its own execution thread. Communication is only by method calls, allowing asynchronous communication in order to avoid undesirable waiting in the distributed setting, where one object need not depend on the responsiveness of other objects. Internally in an object, there is at most one process executing at a time, and intra-object synchronization is programmed by explicit release points and by conditional await statements. These mechanisms provide high-level constructs for process control, and in particular allow an object to change dynamically between active and reactive behavior. Concurrency problems inside the object are avoided since each region from a release point to another release point is performed as a critical region. The operational semantics of ABS has been worked out in [8].

The execution of a distributed system can be represented by its communication history or trace; i.e., the sequence of observable communication events between system components [3, 9]. At any point of time the communication history abstractly captures the system state [5, 4]. Therefore a system may be specified by the finite initial segments of its communication histories. A history invariant is a predicate over the communication history, which holds for all finite sequences in the prefix-closure of the set of possible histories, expressing safety properties [2].

The local history of an object reflects the communication between the object and its surroundings. We model a method call by four history events: (1) sending the invocation message, (2) reacting upon the invocation message, (3) sending the completion message, and (4) reacting upon the completion message. Here, events (1) and (4) are recorded on the local history of the caller, whereas (2) and (3) are recorded on the local history of the callee. In order to observe and reason about object creation using histories, we additionally let histories reveal information about object creation.

In this paper, we develop a partial correctness proof system for the ABS language. A class is specified by a class invariant, which ranges over the class variables and the local communication history of class instances. The proof system is derived from a standard sequential language by means of a semantic encoding. The reasoning inside a class is comparable to reasoning about a sequential while-language, and it amounts to proving that the class invariant is maintained from one release point to another. Since
remote access is restricted to method calls, the classical Hoare rule for assignment is sound, in which we can ignore the problem of undefined right hand side expressions.

For each object, a history invariant can be derived from the class invariant by hiding the local state of the object. Modularity is achieved since history invariants can be established independently for each object and composed at need. This results in behavioral specifications of dynamic system in an open environment. Such specifications allow objects to be specified independently of their internal implementation details, such as the internal state variables. In order to derive a global specification of a system composed of several components, one may compose the specifications of different components. Global specifications can then be provided by describing the observable communication history between each component and its environment.

The work is based on earlier work \cite{1,7}, but the semantic encoding is based on a different notion of locality. In \cite{7} message sending is visible on the local history of both the caller and the callee. Thus, message sending leads to restrictions on the local history of the receiver. Here we solved the problem by redefining message sending, which is considered as a local action of the sender, whereas reacting upon the message is considered as a local action of the receiver. When composed, each reaction by the receiver must match a sent message. In contrast to the earlier work, the current approach allows unrestricted use of assumptions on the environment. This is valuable when reasoning about objects in an open environment. The complete research result is presented in the research report \cite{6}.

References

\begin{enumerate}
\end{enumerate}