

## Goal-Oriented System Semantics

Steve Ward, Chris Terman, and Umar Saif

**Introduction:** MIT's Oxygen project, and related projects elsewhere, aim to provide a computational environment that both responds to human needs and which dynamically adapts in real time as these needs change. The new-generation systems which these projects strive to construct immerse their users in a complex of invisible servers and mobile devices, and provide each user a continuously available, consistent, and user-customized repertoire of high-level services even as the user wanders about and available facilities change.

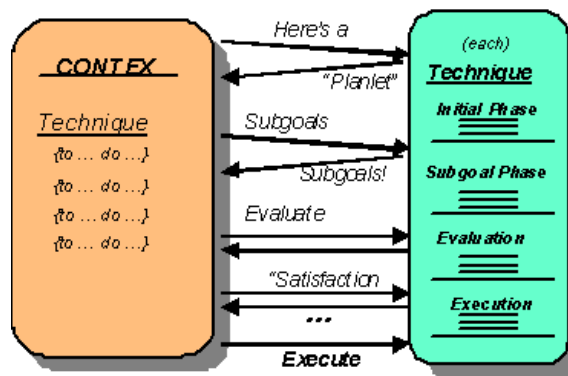
As a tiny example, a user might be involved in a videoconference with a colleague as he wanders about a well-equipped campus. As he moves from one room to another, his video may switch from the small LCD display of his handheld to a wall-mounted plasma screen as the latter comes into view; networking technologies might shift between 802.11b and CDMA depending on resource availability, and video may degrade or disappear altogether as communication bandwidth warrants.

Such behavior necessarily requires frequent reevaluation of available alternatives, as well as heuristic compromises to best address the user's needs with imperfect resources. Conventional techniques for constructing applications, in which top-level function is decomposed into statically-partitioned subfunctions each affixed to a particular API, makes such adaptation exceedingly difficult to program. If there is a change in available resources, it is often insufficient simply to reconsider how to implement the function specified at each API: it is necessary to reconsider the reason that API was selected, and whether an alternative function and API has now become more appropriate.

**Approach:** We propose a new system semantics in which *goals* are formalized as a language construct, and used to guide the automatic construction of a component-based system. Semantically, goals are similar to generic procedure calls: they involve a named generic service (the goal name) as well as an arbitrary number of typed parameters. Thus, **TeleConference(Victor, Steve)** might be a high-level goal whose satisfaction requires a teleconference link between **Victor** and **Steve**, each a parameter of type **Person**.

Unlike procedure calls, however, goals are disembodied from any block of code to be invoked during their execution. Rather, the system approaches the resolution of a goal by searching for one or more *techniques*, each of which constitutes a recipe for resolving a class of goals. Each technique specifies a pattern to be matched against a candidate goal, optional subgoals that must be satisfied for it to proceed, and code to be run in order to cause an incoming goal to be satisfied once the specified subgoals have been achieved. This semantic is reminiscent of Prolog and related logic languages of the 80s, although the implementation technology surrounding it is quite different.

During the resolution of a top-level goal (typically derived from a user command, either typed or uttered), many alternative techniques may be considered at both the top level and lower levels (corresponding to implied subgoals).



Each of the techniques is embodied as an object that plays an active role during the resolution process, in effect offering to satisfy the stated goal so long as certain constraints (expressed as subgoals) are met. The resolution process takes the form of a negotiation between a system object (called a *context*) and a set of applicable technique objects. Typically the context will create and explore a goal tree, heuristically choosing the most satisfactory branch of that tree to be implemented.

