Synthesizing ASMs from Natural Language Requirements

Vincenzo Gervasi
Dipartimento di Informatica, Università di Pisa
e-mail: gervasi@di.unipi.it

Introduction

Abstract State Machines (ASMs) have been successfully used in a number of studies (see [2] for a survey) to specify and model various software-intensive systems. Among the properties that make use of ASMs desirable for this purpose, the ease with which an ASM can be inspected by the customer or final user of the system plays an important role. In fact, the clear operational semantics of ASMs makes it easy even for non specialists to understand the behavior described by an ASM model.

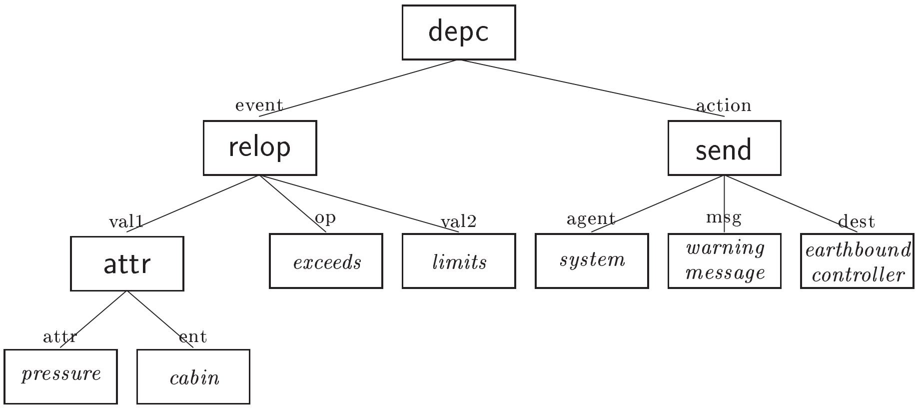
In this paper, we report on the dual approach, that is, synthesizing an ASM model from a natural language description of a system. We show how relevant information can be extracted from informal requirements in natural language, and how the information obtained can be used to produce an ASM model on a level of abstraction comparable to that of the original requirements. The initial model can then be refined in a controlled way to add further details, and possibly obtain the executable code for the system in the end.

The entire process, including parsing of natural language, synthesis and consistency checking of internal models, and generation of ASM models, is supported by tools in our requirements environment Circe [3, 4].

From NL to ASM

To extract information from natural language requirements, we use the domain-based parser Cico, a shallow parser implementing domain-based parsing [1]. Cico can also be described as an engine for the application of a fuzzy rewriting system to a text, using backtracking and heuristic optimization strategies to determine an optimal parse tree for a statement. For the purpose of this paper, it suffices to say that the parsing process transforms natural language requirements into abstract *parse trees* whose nodes represent the (domain-specific) semantic links between the constituent terms (see Figure 1).

From a forest of such parse trees, representing the information extracted from the requirements, a set of *modelers* (software agents running on the parse trees)



"Whenever the cabin pressure exceeds the limits, the system shall send a warning message to the earthbound controller."

Figure 1: A sample requirement and its domain-based parse tree, as produced by Cico.

builds a number of models of the system described by the requirements, e.g. data flow, entity-relationship, finite state automata, event-condition-action models. These models form the basis for the translation into Abstract State Machines.

Both the parser Cico and the modelers interpret the description of the system given in the requirements according to a precise model of a software system. In this view, a software system is modeled as a set of communicating *agents*. Each agent can send, receive, and perform computations of various sorts on typed packets of *information*; an agent can also optionally have a current *mode* of operation (or *state*) from a given set of modes. Agents' behavior can be specified in terms of observable *events* and *conditions*, and of *actions* to be performed; events include causal and temporal dependencies.

Any entity in the domain can have *attributes*[1], and relationships among entities can be declared and used to select specific instances from a generic class, e.g. "If a workshop is canceled, the system shall delete *all the room reservations of the workshop* from its database" (implying that there is a "reserved_for" relation between rooms and workshops).

Once all the different bit of knowledge about the system to specify and its domain are collected from the requirements and consolidated in uniform models by Circe's modelers, the system can be specified as an ASM through systematic (and automatic) translation.

- **System.** The whole system is modeled as a distributed ASM, with agents corresponding to sequential ASM modules. The remarkably general defini-

[1] This includes both real-world entities, and software-only entities, e.g. internal representations that the system maintains. The value of such attributes is considered to be an information packet.

Whenever the cabin pressure exceeds the limits, the system shall set the Cabin Pressure Alarm to TRUE, send a warning message to the earth-bound controller that is controlling the CPM, and switch to emergency state.	**if** Exceeds(Pressure(Cabin),Limits) **then** PressureAlarm(Cabin):=TRUE $\textsc{Send}$(WarningMessage,Controlling(CPM)) State(*self*):=Emergency where $\textsc{Send}(msg, dst) \equiv$ $\mathrm{Channel}(self, dst) := \mathrm{Channel}(self, dst) \uplus \{msg\}$

Table 1: An example translation.

tion given in [5] avoids imposing a particular concurrency mechanism, in keeping with the generality of NL.

- **Relationships and attributes.** Entities are mapped to members of an *ENT* universe. Relations among them are mapped to functions from sets of entities to sets of entities[2] Attributes of entities are modeled as functions on the universe of the entities with the given attribute (a subset of *ENT*).

- **Data flows and computations.** Each sequential ASM communicates with other ASMs through a set of shared functions modeling abstract communication channels. The specific, problem-dependent properties of such channels (e.g., lossyness, in-order delivery, synchronicity, etc.) are not specified in the model. In the same vein, computations are translated as assignments to the 0-ary or 1-ary (in the case of attributes) functions representing the resulting information packet.

- **Dependencies.** Causal, temporal, and data dependencies extracted or inferred from the requirements are directly used to connect events, conditions and actions of our agents to form ASM rules for the appropriate agent.

While we do not have space in this abstract for the details (reported in the full paper), the general flavor of the translation can be observed in the example in Table 1.

The abstraction mechanisms of ASMs are a good match to the analogous abstractions we use everyday in natural language. For example, Table 2 shows how progressively more refined characterization of the constituents of a natural language statement yields correspondent refinements in the translated ASM. Being able to faithfully modulate the level of detail in the ASM version according to the precision of the NL definitions is an important feature that ensures that no arbitrary and unnecessary decisions are made in the translation process.

[2] In practice, one-to-one and one-to-many relations results in singletons that are treated as single elements for the sake of conciseness.

If $\underbrace{\text{the cabin pressure exceeds the limits}}_{\text{EVENT}}$, ...	**if** CabinPressureExceedsLimits() **then** ...
If $\underbrace{\text{the cabin pressure}}_{\text{INFORMATION}}$ $\underbrace{\text{exceeds the limits}}_{\text{UNARYOP}}$, ...	**if** ExceedsLimits(CabinPressure) **then** ...
If $\underbrace{\text{the cabin pressure}}_{\text{INFORMATION}}$ $\underbrace{\text{exceeds}}_{\text{RELOP}}$ $\underbrace{\text{the limits}}_{\text{INFORMATION}}$, ...	**if** Exceeds(CabinPressure,Limits) **then** ...
If $\underbrace{\text{the cabin}}_{\text{ENTITY}}$ $\underbrace{\text{pressure}}_{\text{ATTRIBUTE}}$ $\underbrace{\text{exceeds}}_{\text{RELOP}}$ $\underbrace{\text{the limits}}_{\text{INFORMATION}}$, ...	**if** Exceeds(Pressure(Cabin),Limits) **then** ...

Table 2: An example of parallel refinement in Natural Language and in ASM.

Conclusions

Exhibiting a systematic translation from NL statements to ASMs adds experimental evidence to the thesis that ASMs constitute a good common language for analyst-customer communications. Moreover, since our translation comes from a set of classical software models (E-R, DFD, etc.) that are validated by automatic tools for intra- and inter-model consistency, the analyst can rest assured that the generated ASM will share analogous consistency properties, and can concentrate on more pregnant issues rather than on details. Finally, since the CIRCE environment can generate other representations extracted from the same requirements (including, for example, UML diagrams), translation from a common requirements source to ASMs provides a good opportunity for developing multi-view specifications with reasonable effort.

References

[1] V. Ambriola and V. Gervasi. Experiences with domain-based parsing of natural language requirements. In G. Fliedl and H. C. Mayr, editors, *Proceedings of the Fourth International Conference on Applications of Natural Language to Information Systems*, number 129 in OCG Schriftenreihe (Lecture Notes), pages 145–148, 1999.

[2] E. Börger and J. Huggins. Abstract State Machines 1988-1998: Commented ASM bibliography. *Formal Specification Column, EATCS Bulletin*, (64):105–127, Feb. 1998.

[3] V. Gervasi. *Environment Support for Requirements Writing and Analysis*. PhD thesis, University of Pisa, February 2000.

[4] V. Gervasi and B. Nuseibeh. Lightweight validation of natural language requirements. In *Proc. of the 4th International Conference on Requirements Engineering*, pages 140–148, June 2000.

[5] Y. Gurevich. Evolving algebra 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.