Choosing an appropriate computation model

The GLIF language includes several constructs that make the computation model far more complex than a simple sequential system. These are:

1) Branch steps that enable parallel paths
2) Triggering events and exceptions that should be monitored, and on their occurrence, create an interrupt and cause some guideline step to be executed.

The interaction of branch and triggering events and exceptions must be made clear. For example, if there is an exception that causes an action to terminate, and that action has been executed in parallel, then are both parallel paths terminated? There is also the issue of real-time constraints: constraints on the duration of a process, and on the reaction time of the system to a triggering event. There are many things that we haven't defined about the event-triggered model and that we should address.

There are several computational models that are possible for concurrent systems with real-time constraints (Real-Time systems). The one that we should use should be formal, in order to allow verification. The different models are covered in (Ostroff 1992; Rhalibi, Crestani et al. 1995) (The second reference could be found in: http://iel.ihs.com:80/cgi-bin/iel_cgi?sess=280864755&prod=IEL&page=%2fiel3%2f3576%2f10687%2f00496671%2epdf).

Of the different possible formal models, I am most familiar with Statecharts (Harel, Lachover et al. 1990), and have used its validation tools. I am somewhat familiar with Petri-Nets (Peterson 1981) and several temporal logics. I am also familiar with LOTOS (Ardis 1991) (a process algebra) and with Z (Coombes and McDermid 1993) (Abstract data type, based on set theory), but not with the Z variants that allow specifying real-time constraints.

On verification and validation

Verification: mathematically proving that the specification is correct, complete, feasible, unambiguous, and free from omissions and contradictions.

Validation: demonstrating, by simulating the system's behavior (transitions through states) that certain properties hold.

Theorem Provers and Model Checkers: can prove properties by using mathematical methods.

Desired properties:
Safety: certain invariants always hold (e.g., daily dose is never exceeded)
Response: If something holds at a certain point in time, then eventually some other formula will hold (e.g., if patient scheduled a visit, then he sill be seen by the physician within 30 days).

Reachability: is state s2 reachable from sate s1
Liveness: the program can potentially run forever without crashing
Boundedness: finite state space
Reversibility: initial state recovery
Mutual exclusion: sharing of common resources: when a resource is being used it sometimes needs to be forbidden for another resource to use it. For example, only one resource can update a database entry at one timestamp.

There are many more properties. Basically, anything that can be expressed using temporal logic (the modal operators Always, and Eventually [in the future and in the past]).

Verification and validation **tools** are available for many formal methods. However, they are not at all easy to use, and are not all automated, but require a lot of user input.

Validation Tasks (taken from the InterMed Army proposal)

(a) Identify reachability problems (i.e., is node B reachable from node A?).

This task can be done by tools or methods that analyze Petri Nets. This would mean that a GLIF guideline specification would have to be translated to a Petri Net. Other properties, like mutual exclusion, safety, and response can also be proved by translating the GLIF specification into a formal language. We should decide whether we want to provide this verification functionality or not.

(b) Find deadlock situations, or missing "elses" (e.g., a branch with only one branch destination or a decision with only one next step)

Finding a branch with only one branching path or a decision step with only one decision option should not be hard to do.

We could also look at all the decision options of one decision and see if (1) they are mutually exclusive or not, and (2) cover all cases (e.g., cover ages 20-30, 30-50, 50+, but not "<20")

Deadlock is the opposite of liveness, see (a).

(c) Assess logical inconsistencies or redundancies in criteria

If the guideline contains a branch (Mor: should be decision?) criterion that utilizes certain data elements, the checker can make sure that:

(1) Branch (Mor: Decision?) criteria are not redundant, and do not result in situations where the logic of the criteria prohibits traversal to certain parts of the guideline)

This is covered by the last point in item (b)

(2) Actions exist at or before that point in the guideline through all possible traversal paths, which result in the collection of those data elements

What is meant here is that if there needs to be a decision then we should invoke the Get_Data_Action beforehand. The validation task is to check that all the data items that are referenced by a criterion are listed as data items that the guideline contains, and that the data is acquired beforehand by the Get_Data_Action for variable data items.

(d) Analyze decision steps to ensure that alternatives are mutually exclusive, non-redundant or conflicting, nor missing, or at very least to document such limitations.

This is covered by the last point in item (b)

(e) There is also a Data and Logic, that is described in the document that Dongwen prepared. The checking include: 1) data type (Mor: is this what Qing calls “semantic checks”?) and range checking, and 2) logic completeness and consistency checking. (Mor: I think that this function includes the Arden parser that Lola is working on. Also, it has to do with the last paragraph in item (b))

Ardis, M. (1991). “Formal Methods for Telecommunication System Requirements: a Survey of Standardized Languages.” Annals of Software Engineering **3**: 157-187.

Coombes, A. and J. McDermid (1993). “Specifying Temporal Requirements for Distributed Real-Time Systems in Z.” Software Engineering Journal(September): 273-283.

Harel, D., H. Lachover, et al. (1990). “STATEMATE: A Working Environment for the Development of Complex Systems.” IEEE Transactions on Software Engineering **16**(4): 403-414.

Ostroff, J. (1992). “Formal methods for the psecification and design of real-time safety-critical systems.” The Journal of Systems and Software: 33-60.

Peterson, J. (1981). Petri Net Theory and the Modeling of Systems. Englewood Cliffs, NJ, Prentice-Hall.

Rhalibi, A. E., D. Crestani, et al. (1995). Petri nets, FCCS, and synchronous languages to specify discrete events systems: a comparative synthesis on validation power. Proceedings., 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, 1995. ETFA '95,.