Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 115 (2005) 59–68
www.elsevier.com/locate/entcs

Integrating the Semantics of Deterministic Multi-layered Communication Systems

Rui Gustavo Crespo[1]

Department of Electrical Engineering and Computers
Technical University of Lisbon
Av. Rovisco Pais, 1049-001 Lisboa, Portugal

Abstract

Communication systems are now developed and tailored by different classes of participants, each one focusing his/her attention on particular aspects of the system.
We analyse how Phone and Internet services can be described by event and state representation schemes. We propose to use the denotational semantics to express the interpretation of different representation schemes for the development and personalisation of deterministic communication systems.

Keywords: Denotational semantics, Event and State viewpoints, Viewpoint integration

1 Introduction

Communicating systems are event driven, because they are distributed and reactive. Therefore, the event viewpoints are the first, the most important, and frequently the only viewpoints identified. Also, the third characteristic of the event driven systems, the concurrency, is not always present in the communication systems, such as the private phone exchange systems-PBX.

Lately, the market started to require the possibility of users to tailor the systems to their specific needs, such as the Internet telephony [10]. However, many users only understand state driven viewpoints. Hence questions arise, such as how can we interface the separated event and state viewpoints, and can the different representation schemes have one single formal semantics.

[1] R.G.Crespo@digitais.ist.utl.pt

doi:10.1016/j.entcs.2004.09.031

1.1 Representation schemes

Because there are different participant classes in the development and use of communication systems, the event and the state viewpoints must use different representation schemes. Because the participants must be able to access to all information they need, the representation schemes must be powerful enough.

To reason about the whole system and guarantee the coherence between the viewpoints, the two representation schemes must share the same semantics. Formal semantics of event driven representation schemes include traces [7] and LTS [5]. For state driven programming languages, analysts may adopt operational [11], denotational [12] and axiomatic semantics [6].

The integration of event and state driven semantics reveals to be difficult (see SDL [4]). However, the use of a limited part of the representation scheme simplifies the integration of the semantic formalisms. Being nondeterminism absent, we adopt the denotational semantics and interpret events as deterministic state changes. Reason for the choice include the user's familiarity to the state driven viewpoints, the tools availability and the easier integration of semantics due to the interpretation of the syntactic elements by functions. We note that the reverse approach is also possible, by modelling states as equivalence classes of event sequences.

1.2 System example

We are developing a PBX with many features, where users choose the features they wish to subscribe and define policies for feature selection upon an event arrival. We adopted a three-layer architecture. Starting from the lowest layer,

(i) Call processing layer (CP), with a stand-alone definition of features.

(ii) Context update layer (CU), tailors the features to the individual needs.

(iii) Selection and execution layer (SE), which describes how to choose the feature to be executed.

2 Event driven scheme

FET-Feature Execution Trees [2] depicts the control logic of features in terms of subtrees that can succeed or fail, causing other subtrees to be evaluated.

2.1 FET language

A telephone feature must have the form shown in (1), with a trigger event $\varepsilon T \lambda$ and three conditions: pre-condition, execution body and post-condition.

(1) $seq(event \wedge pre-condition, seq(body, post-condition))$

The pre-condition states which predicates must be satisfied, in order the system be able to execute the feature. The body depicts the events that the subscriber (L) and the other features (R) must launch, and the predicates that must be satisfied. The post-condition indicates the new status of the subscriber and which events are launched, after the execution of the body.

The figure 1 depicts the abstract syntax of the FET language. τ, ε and π are the syntactical elements of terminal, event and predicate identifiers.

$$FET : \Phi ::= \varepsilon T\lambda, X \rightarrow X \rightarrow X$$

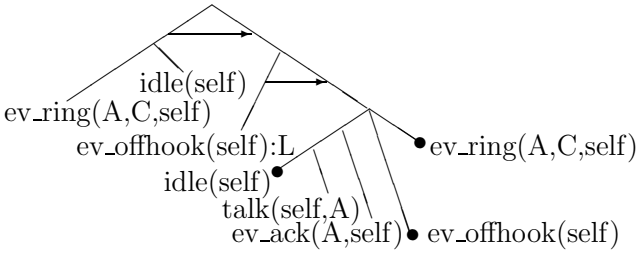
$$Cond_F : X ::= X\Omega_\phi X \mid not\ X \mid true \mid \Sigma\ T\lambda \mid \Phi$$

$$Oper : \Omega_\phi ::= seq \mid and \mid or, \quad Selector : \Sigma ::= \varepsilon \mid \varepsilon : \Delta_F \mid \pi$$

$$Direction : \Delta_\phi ::= launched \mid required, \quad TermList : T\lambda ::= \tau, T\lambda \mid \tau$$

Fig. 1. Abstract syntax of FET language

Example 2.1 The CA service is triggered by the launch of the *ev_ring* event, and requires the subscriber to be idle. The body requires the subscriber to launch the $ev\_offhook$ event. After the execution of the CA service, one event is launched (*ev_ack*), two events are cleared ($ev\_offhook$ and *ev_ring*), the predicate *talk* becomes satisfied and the predicate *idle* becomes not satisfied.



2.2 FET Semantics

Being nondeterminism absent in FET, the denotational semantics is powerful enough to identify the FET semantics and the events are interpreted, likewise predicates, by status functions.

Figure 2 depicts the semantic domains. The syntactic categories for FET are $Term = \{ann, \ldots\}$, $ID_p = \{idle, \ldots\}$ and $ID_e = \{ev\_ring, \ldots\}$. $Status$ is, simply, the union of $Status_\pi$ and $Status_\varepsilon$. The semantics of a condition and the feature execution are a pair, made of the three-value logic result and the new system status. $\perp$ represents an undefined execution of the feature.

$$Status_\pi : \sigma_\pi ::= Term \rightarrow ID_p \rightarrow seq\ Term \rightarrow Bool_\perp$$

$$Status_\varepsilon : \sigma_\varepsilon ::= ID_\varepsilon \rightarrow seq\ Term \rightarrow Bool_\perp$$

$$C_\phi : Cond_F \rightarrow Term \rightarrow S \rightarrow Bool_\perp \times Status \bigcup$$

$$Cond_F \rightarrow Term \rightarrow \Delta_\phi \rightarrow Status \rightarrow Bool_\perp \times Status$$

$$F : Term \rightarrow Status_\varepsilon \rightarrow seq\ Term \rightarrow Cond_F \rightarrow Cond_F^2 \rightarrow Status \rightarrow Bool_\perp \times Status$$

Fig. 2. FET semantic domains

Figure 3 shows the valuation functions for *seq* and *event* constructs. The valuation functions for *true* and the *not, or, and* constructs are the classical. The valuation function of *seq* evaluates the second subtree, if and only if, the first subtree has the valuation value of *true*. The value function of the required events and predicates is, simply, the value of the corresponding *Status* member. $build : TermList \rightarrow seq\ Term$ converts a set into a sequence of terminals.

$$C_\phi[\![X_1\ seq\ X_2]\!] self\ \sigma = (if\ C_\phi[\![X_1]\!] self\ \sigma\downarrow 1 \neq 1\ then < C_\phi[\![X_1]\!] self\ \sigma\downarrow 1, \sigma >,$$
$$else\ (if\ C_\phi[\![X_2]\!] self\ \sigma\downarrow 1 = \perp\ then < \perp, \sigma >,$$
$$else\ (< C[\![X_2]\!] self\ \sigma\downarrow 1, C_\phi[\![X_2]\!] self\ \sigma\downarrow 2 \oplus C_\phi[\![X_1]\!] self\ \sigma\downarrow 2 >)))$$

$$C_\phi[\![\varepsilon\ T\lambda]\!] self\ \delta\sigma\ = (if\ \delta = required\ then < S_\varepsilon\ build(T\lambda), \sigma >,$$
$$else < 1, \sigma \oplus \varepsilon \mapsto build[\![T\lambda]\!] \mapsto 1 >)$$

Fig. 3. Valuation of seq and event C constructs

Figure 4 shows the valuation function of one individual feature. The valuation function says that the status is updated, if and only if the following results hold in succession: the event occurs, the pre-condition is evaluated to *true* and the execution body is also evaluated to *true*. $conv : Cond_F \rightarrow Term \rightarrow S$ generates a status function from a post-condition.

$$F[\![\varepsilon\ T\lambda, X_1 \rightarrow X_2 \rightarrow X_3]\!] self\ \sigma =$$
$$(if\ C_\phi[\![\varepsilon T\lambda]\!] self\ required\ \sigma\downarrow 1 \neq 1\ then < C_\phi[\![\varepsilon\ T\lambda]\!] self\ required\ \sigma\downarrow 1, \sigma >,$$
$$else\ (if\ C_\phi[\![X_1]\!] self\ \sigma\downarrow 1 \neq 1\ then < C_\phi[\![X_1]\!] self\ w\sigma\downarrow 1, \sigma >,$$
$$else\ (if\ C_\phi[\![X_2]\!] self\ \sigma\downarrow 1 \neq 1\ then < C_\phi[\![X_2]\!] self\ \sigma\downarrow 1, \sigma >,$$
$$else < 1, \sigma \oplus conv[\![X_3]\!] self\ \sigma >)))$$

Fig. 4. Valuation function of F

3 State driven automata

The features are personalized with CPL-Call Processing Language [8] scripts.

3.1 CPL language

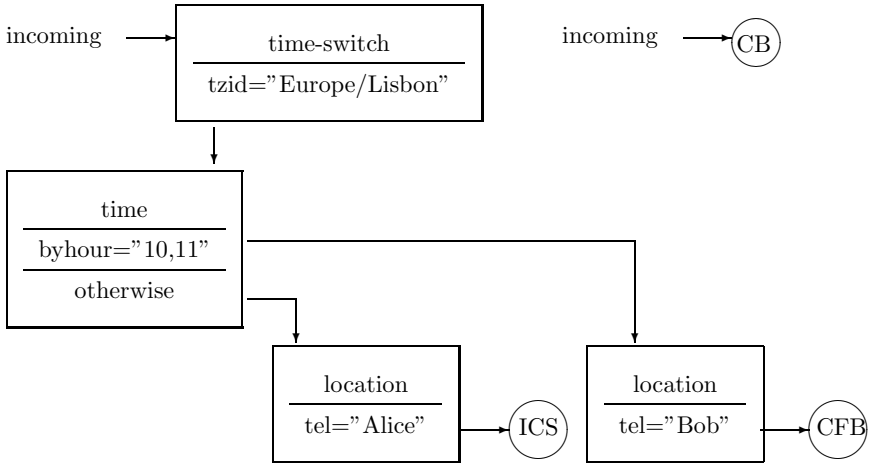
CPL scripts are trees of two kinds, *incoming*-for the arrival event and *outgoing*-for subscriber launched events.

The intermediate nodes are conditions, referred as *switches*. CPL offers over the values present in the call originator, locations and over time. Here, we restrict to the AS(address-switch) and to the TS(time-switch). The *location modifiers* change the locations or search for locations, in one location set.

If the condition is satisfied, the script refines the condition with an extra sequence of conditions, referred as *outputs*. There is one single output for the *address-switch*, the *address*. CPL provides a very reach set of outputs for the *time-switch*. There are no outputs for the location insertion and removal. If all refinements fail, the *otherwise* may be used as the last refinement condition.

For every output, the script proceeds with another switch, or a script leaf. CPL defines script leaves by event launch, designated as *signalling operations*. In this article, script leaves are references to features.

Example 3.1 The POTS is automatically made available to all users. Also, the incoming calls must be forwarded to Bob, between 10:00 and 12:00 AM. Other times, the incoming calls activate the ICS(Alice) feature.



The abstract syntax of the CPL is depicted in the figure 5. δ, π, τ, ν are syntactical elements of, respectively, a time-stamp, a location, a tag name and a value name.

$$Script : N ::= N\Sigma, \quad Node : C\Pi N ::= N\Sigma \mid N\Lambda \mid \Phi$$

$$SNode : N\Sigma ::= \Sigma_\nu\ O\lambda \mid \Sigma_\nu\ T\lambda\ O\lambda, \quad LNode : N\Lambda ::= \Omega_\Lambda\ O\lambda \mid \Omega_\Lambda\ T\lambda\ O\lambda$$

$$OutList : O\lambda ::= O_\nu\ C\Pi N \mid O_\nu\ C\Pi N\ O\lambda, \quad TagList : T\lambda ::= T \mid T\ T\lambda$$

$$Tag : T ::= \tau = \pi \mid \tau = \delta \mid \tau = \nu$$

$$LocOper : \Omega_\Lambda ::= location \mid removal \mid lookup$$

$$SwName : \Sigma_\nu ::= AS \mid TS, \quad OutName : O_\nu ::= address \mid time \mid otherwise$$

Fig. 5. Abstract syntax of CPL language

3.2 CPL semantics

The CPL semantics is presented informally [8] and in temporal logics [13]. Here, we provide an overview of the CPL denotational semantics.

The figure 6 depicts the CPL semantic domains. $\Lambda = \mathbb{P}Term$, representing a location set and $\Delta_T = \{2003/11/26@11:00, \ldots\}$, representing the set of all possible time stamps, are syntactic categories. The semantics of a script is a pair, made of the new system status and a new set location.

$$B_\pi ::= OutList \to Term \to Bool_\perp,\ B_\delta ::= OutList \to \Delta_T^2 \to (Bool_\perp \times \Delta_T)$$

$$S ::= Node \to (Status \times \Lambda) \to (Term^3 \times \Delta_T) \to (Status \times \Lambda)$$

Fig. 6. CPL Semantic domains

The figure 7 depicts some of the valuations of the B functions. δ is the current time and ι is the beginning of one time span. The time mark may contain several tags, such as the delimited time range (*time dtstart>"9" dtend<"17"*): in this case, the result is the conjunction of the logical result for each tag. The tag may embrace several values, such as the non-continuous periods (*time by-hour="10" "12"*): in this case, the result is the disjunction of the logical result for each value in the tag.

$$B_\pi[\![address\ is = A]\!](\pi) = (if\ \pi = A\ then\ 1, else\ 0)$$

$$B_\delta[\![time\ T_h\ T_t]\!](\delta, \iota) = let\ \iota_{middle} = B_\delta[\![time\ T_t]\!](\delta, B_\delta[\![time\ T_h]\!](\delta, \iota)\downarrow 2)$$
$$in < B_\delta[\![time\ T_h]\!](\delta, \iota)\downarrow 1 * \iota_{middle}\downarrow 1, B_\delta[\![time\ T_t]\!](\delta, \iota_{middle})\downarrow 2 >$$

$$B_\delta[\![time\ dtstart = T]\!](\delta, \iota) = (if\ \delta > T\ then\ <1, T>, else\ <0, \iota>)$$

Fig. 7. Valuation of B functions

The figure 8 depicts some valuation functions of S. For the *address-switch*, we focus on the originator address. For the *time-switch*, we simply adapt the

current time to the designated time-zone of the call (using the auxiliary $shift$ function). When the server performs a switch in the script, the branch is selected according to the evaluation of the B_π and B_δ functions: if selected, it evaluates the corresponding node, if not S proceeds in the output list.

$$S[\![AS\ field{=}origin\ O\ N]\!](\sigma,\Lambda)(<\pi_1,\pi_2,\pi_3>,\delta) =$$
$$(if\ B_\pi[\![O]\!](\pi_1) = 1\ then\ S[\![N]\!](\sigma,\Lambda)(<\pi_1,\pi_2,\pi_3>,\delta), else <\sigma,\Lambda>)$$
$$S[\![TS\ T_z\ O\ N]\!](\sigma,\Lambda)(Term^3,\delta) =$$
$$(if\ B_\delta[\![O]\!](shift(\delta),0)\downarrow 1 = 1\ then\ S[\![N]\!](\sigma,\Lambda)(Term^3, shift(\delta)),$$
$$else <\sigma,\Lambda>)$$
$$S[\![location\ tel = \pi_t\ N]\!](\sigma,\Lambda)(Term^3,\delta) = S[\![N]\!](\sigma,\Lambda\cup\{\pi_t\})(Term^3,\delta)$$
$$S[\![\Phi]\!](\sigma,\Lambda)(Term^3,\delta) =< F[\![\Phi]\!]self\sigma\downarrow 2,\Lambda>$$

Fig. 8. Valuation of S function

4 Logic constrained state driven scheme

The final selection of the feature to be executed is implemented IRS-Interaction Resolution and feature Selection, which incorporates deontic logic constraints.

The logic approach makes easier the abstract definition of the selection policies. The deontic operators are used at the top abstract SE layer, not on the CP layer [1]. We adopt a reduction of the deontic logics to the dynamic logics [9], where *Interdiction* is the failure of an action execution. Hence, the paradoxes of the deontic logics do not occur.

4.1 IRS Language

SE is divided into two parts, the interaction resolution and the policy selection.

4.1.1 Interaction resolution

The undesired interaction between pairs of features is resulted by a set of interdition constraints, in the form shown in (2).

(2) $(Request \wedge Condition) \rightarrow Interdictions$

The *Request* subformula is a conjunction of propositions, each one representing a feature selected in the CU layer. *Condition* is a ground formula, that identifies the values the terminal must hold and the events that must

have been launched. The *Interdictions* identify the features that cannot be executed, if the *Request* and the *Condition* subformulas are evaluated to *true*.

Example 4.1 Consider the ICS feature holds the highest priority, and CFB holds higher priority than the CW feature.

$$(RequestICS \wedge RequestCFB) \rightarrow I\ CFB, (RequestCFB \wedge RequestCW) \rightarrow I\ CW$$

Figure 9 gives the abstract syntax of resolution part of the IRS language. The symbols ρ and ϕ are, respectively, a proposition representing a feature request for execution and a set of features.

$$Form_{IR} : \gamma ::= R, C_\iota \Rightarrow \Delta_{IR} \mid \gamma;\gamma$$
$$Request : R ::= \rho \mid R \wedge R$$
$$Cond_{IR} : C_\iota ::= R \mid C_\iota\ \Omega_\iota\ C_\iota \mid \neg C_\iota \mid \pi\ T\lambda \mid \exists \xi : C_\iota \mid \forall \xi : C_\iota$$
$$Deontic : \Delta_{IR} ::= I\ \phi \mid \Delta_{IR} \wedge I\ \phi, \quad OperL : \Omega_\iota ::= \wedge \mid \vee \mid \rightarrow$$

Fig. 9. Abstract syntax of the resolution part

4.1.2 Selection policies

The IRS selection part is a declarative language, with a script made of guarded commands. The instructions are simple: clear an event, execute the selected feature, print one message and update the hop's value (a terminal counter, to avoid feature looping [3]). The abstract syntax of the IRS selection part is depicted in the figure 10. τ, Φ and $self$ are defined in the FET language. s and η are, respectively, a string and a natural number.

$$Script : \Sigma R ::= C\lambda$$
$$Cond_{IS} : C_\sigma ::= size = \Delta_{IS} \mid head = \rho \mid C_\sigma \Omega_\iota C_\sigma \mid \neg C_\sigma \mid \epsilon T\lambda$$
$$Stmt : \Sigma ::= C\lambda \mid CLEAR\ \epsilon \mid EXEC\ \Phi \mid MSG\ M \mid UPDATE\ \tau, I$$
$$CondList : C\lambda ::= C\lambda\ C_\sigma \rightarrow \Sigma\lambda \mid C_\sigma \rightarrow \Sigma\lambda, \quad StmtList : \Sigma\lambda ::= \Sigma\lambda\ \Sigma \mid \Sigma$$
$$Dimension : \Delta_{IS} ::= many \mid one \mid null$$
$$Value : I ::= \eta \mid I + I \mid HOP\ \tau, \quad Msg : M ::= s \mid M + M \mid self$$

Fig. 10. Abstract syntax of IRS language-selection part

Example 4.2 One simple selection policy is to choose the first feature in the sequence outcome of the resolution part of the SE layer.

[size=many ∨ size=one] → (
[(head=CFA ∨ head=CFB) ∧ event=ev_call(A,B,self)] →
(update(A,hop(A)+1); exec(head))
[head≠CFA ∧ head≠CFB] → (update(A,0); exec(head)))
[size=null] → (Msg("Internal error in terminal " + self); Clear(event))

4.2 IRS Semantics

The semantics domains of the constraint part are depicted in the figure 11. The semantics of a formula is a subset of features, candidate for execution. The denotational semantics of the IRS selection part and of the programming languages [12] are similar.

$$FList:\ \Phi R ::= \mathbb{P}\Phi$$

$$Constr:\Gamma ::= Form_I \to \Phi R \times Status_\pi \times Term \to \Phi R$$

$$Expr:\quad E ::= Cond_{IR} \to \Phi R \times Status_\pi \times Term \to \mathbb{N}$$

Fig. 11. Semantic domains of the constraint part

The valuation functions for the *Expr* logical operators are equal to the valuation functions for similar C_ϕ constructs. The figure 12 depicts the valuation functions for the *Expr* syntactic constructs and for the feature request. Because the number of terminals is fixed, E is decidable.

$$E[\![\rho]\!](\phi\rho,\sigma,self) = (if\ \rho \in \phi\rho\ then\ 1, else\ 0)$$

$$E[\![\forall\xi : C_\iota]\!](\phi\rho,\sigma,self) = E[\![[C_\iota]^\xi_{term_1}]\!](\phi\rho,\sigma,self) * \ldots * E[\![[C_\iota]^\xi_{term_N}]\!](\phi\rho,\sigma,self)$$

Fig. 12. Evaluation functions for Expr

The figure 13 depicts some valuation functions for *Constr*. The evaluation of a sequence of two formulas, with equal Requests and equal Conditions, is the same as one single formula with the same Request and Condition and conjuntion of the interdictions of the two formulas.

$$\Gamma[\![R, C_\iota \Rightarrow I\ \phi_1]\!](\phi\rho,\sigma,self) =$$
$$(if\ E[\![R]\!](\phi\rho,\sigma,self) = 1\ and\ E[\![C_\iota]\!](\phi\rho,\sigma,self) = 1\ then\ \phi\rho \setminus \phi_1, else\ \phi\rho)$$

$$\Gamma[\![R, C_\iota \Rightarrow \Delta_{IR} \wedge I\ \phi_1]\!](\phi\rho,\sigma,self) = \Gamma[\![R, C_\iota \Rightarrow I\ \phi_1]\!]$$
$$(\phi\rho \setminus \Gamma[\![R, C_\iota \Rightarrow \Delta_{IR}]\!](\phi\rho,\sigma,self),\sigma,self)$$

Fig. 13. Evaluation function for Constraint

References

[1] Barbuceanu, M., T. Gray and S. Mankovski, *Coordinating with obligations*, in: 2^{nd} *Int'l Conference on Autonomous Agents* (1998), pp. 62–69.

[2] Crespo, R., *The Semantics of Feature Execution Trees*, in: 3^{rd} *IEEE Workshop on Formal Specifications of Computer-based Systems*, 2002, pp. 39–46.

[3] Crespo, R., L. Logrippo and T. Gray, *Feature Execution Trees and Interactions*, in: *Int'l Conference on Parallel and Distributed Processing Techniques and Applications*, 2002, pp. 1230–1236.

[4] Eschbach, R., U. Glässer, R. Cotzhein and A. Prinz, *On the Formal Semantics of SDL-2000: A Compilation Approach Based on an Abstract SDL Machine*, in: *Abstract State Machines 2000*, LNCS 1912 (2000), pp. 242–265.

[5] Hennessy, M., "Algebraic Theory of Processes," MIT Press, 1988.

[6] Hoare, C., *An axiomatic basis for computer programming*, Communications of the ACM **12** (1969), pp. 576–580,583.

[7] Hoare, C., "Communicating Sequential Processes," Prentice-Hall, 1985.

[8] Lennox, J. and H. Schulzrinne, *CPL: A Language for User Control of Internet Telephony Services*, Technical Report draft-ietf-iptel-cpl-06 (2002).

[9] Meyer, J., *A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic*, Notre Dame Journal of Formal Logic **29** (1988), pp. 109–136.

[10] Schulzrinne, H. and J. Lennox, *The IETF Internet Telephony Architecture and Protocols*, IEEE Network Magazine **13** (1999), pp. 18–23.

[11] Wegner, P., *The Vienna Definition Language*, ACM Computing Surveys **4** (1972), pp. 5–63.

[12] Winskel, G., "Formal Semantics of Programming Languages," MIT Press, 1993.

[13] Xu, Y., "Detecting Feature interactions in CPL," Master's thesis, School of Information Technology and Engineering at University of Ottawa (2003).