Integración de Lógica Lineal Temporal a Modelos de Arquitecturas de Software Basadas en Componentes Especificadas a Través del Cálculo ρ_{arq}

Autor: Oscar Javier Puentes Director: Henry Alberto Diosa

Maestría en Ciencias de la Información y las Comunicaciones

Universidad Distrital Francisco José de Caldas, 2017

Contenido

Formulación del problema

Actualmente con el cálculo ρ_{arq} no es posible especificar propiedades temporales, relacionadas con el orden lógico de ejecución de una arquitectura; es decir, determinar en un momento dado si una arquitectura llega a cierto estado deseado que depende de la ejecución en cierto orden de sus componentes.

Hipótesis
Es posible formular un mecanismo que permita especificar y verificar propiedades temporales para modelos de arquitecturas de software basadas en componentes modelados a través del cálculo ρ_{arq}

Chequeo de modelos y Arquitecturas de software

- El chequeo de modelos es una técnica de verificación formal para la evaluación de propiedades de sistemas de información y comunicación.
- Requiere un modelo del sistema que se va a analizar, una propiedad deseada y un método para verificar sistemáticamente si el modelo satisface o no la propiedad.
- Sus fundamentos formales se encuentran en la lógica proposicional, teoría de autómatas, lenguajes formales, estructuras de datos y algoritmos de grafos. [1, 2]

Cálculo ρ_{arq} - Sintaxis

Se compone de un conjunto de símbolos y expresiones basadas en las entidades del cálculo ρ original y adaptando algunas al contexto de las arquitecturas de software [3].

SÍMBOLOS	
x, y, z	variables
a, b, c	nombres
$u, v, w ::= x \mid a$	referencias

EXPRESIONES		
$E, F, G ::=$	$\top$	Null
$\mid$	$E \wedge F$	Composición
$\mid$	$E^{(int)}$	Parte interna o encapsulada del componente E
$\mid$	$if(C_1 \cdots C_n)\ else\ G$	Combinador de selección condicionada
$\mid$	$x :: \overline{y}/E$	Abstracción
$\mid$	$x\overline{y}/E$	Aplicación
$\mid$	τ/E	Reacción interna
$\mid$	$\exists wE$	Declaración
$\mid$	$x : \overline{y}/E$	Replicación de abstracción
$\mid$	$E^{\top}$	Ejecución exitosa del componente E
$\mid$	$E^{\perp}$	Ejecución no exitosa del componente E
$\mid$	$OSO(E)\ do\ F\ else\ G$	On Success Of
$\mid$	$!OSO(E)\ do\ F\ else\ G$	Observación repetida
$\phi, \psi ::=$	$\dot{\top}$	Verdad Lógica
$\mid$	$\perp$	Falsedad Lógica
$\mid$	$x = y$	Restricción ecuacional
$\mid$	$\phi \dot{\wedge} \psi$	Conjunción de restricciones
$\mid$	$\dot{\exists} x\phi$	Cuantificador existencial

Figura: *Sintaxis del cálculo* ρ_{arq}

Fuente: [4]

Cálculo ρ_{arq} - Semántica Operacional y Ejecución de una arquitectura

Reglas de reducción

Permiten especificar formalmente el avance de la ejecución de una arquitectura.

$(A_{\rho_{arq}})$ $\phi \wedge x : \overline{y}/E \wedge x'\overline{z}/F \longrightarrow \phi \wedge x : \overline{y}/E \wedge [\overline{z}/\overline{y}]E^{(int)} \wedge F$ $si\ \phi \models_\Delta x = x', \mathcal{V}(\overline{z}) \cap \mathcal{BV}(E^{(int)}) = \emptyset$
$(C_{\rho_{arq}})$ $\phi_1 \wedge \phi_2 \longrightarrow \psi$ $si\ \phi_1 \hat{\wedge} \phi_2 \models\!\!\models_\Delta \psi$
$(Comb_{\rho_{arq}})$ $\phi \wedge if\ (C_1) \ldots (C_n)\ else\ F\ fi \longrightarrow \begin{cases} E_k, & si\ \phi \models_\Delta \psi_k \\ F, & si\ \phi \models_\Delta \neg\psi_k\ ; \forall k = 1, 2, \ldots, n \end{cases}$ $Con\ \ C_k ::= \exists\overline{x}(\psi_k\ Then\ E_k)\ ;\ k = 1, 2, \ldots, n$
$(Ejec_\tau)$ (a) $[OSO(E)\ do\ F\ else\ G] \wedge E^\top \longrightarrow F$, *debido a que hay ejecución exitosa del componente*   (b) $[OSO(E)\ do\ F\ else\ G] \wedge E^\perp \longrightarrow G$, *debido a que no hay ejecución exitosa del componente*

Figura: *Reglas de reducción del* ρ_{Arq}

Fuente: [4]

Especificación de un componente

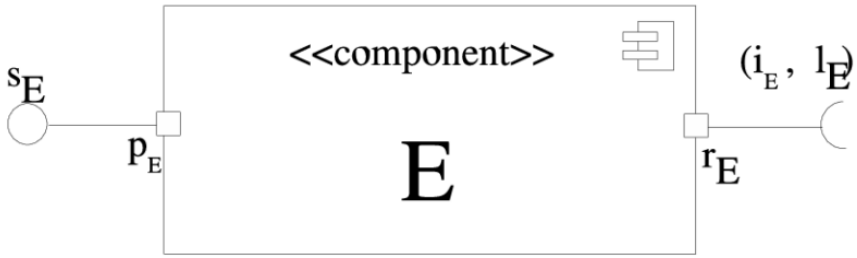


Figura: *Representación gráfica de un componente*

Fuente: [4]

La especificación formal de cada componente se realiza a través de la descripción de cada una de sus interfaces.

Provisión de servicio: $PROV_E(p,s) \stackrel{def}{=} p_E : x/xs_E \equiv p_E :: x/xs_E \wedge p_E : x/xs_E$

Requisitor de servicio: $REQ_E(r,l,i) \stackrel{def}{=} \exists l_E[(r_E :: y/yl_E \wedge l_E :: i_E/E^{(int)})]$

Especificación formal: $E \stackrel{def}{=} PROV_E(p,s) \wedge REQ_E(r,l,i)$

Lógica Lineal Temporal (LLT)

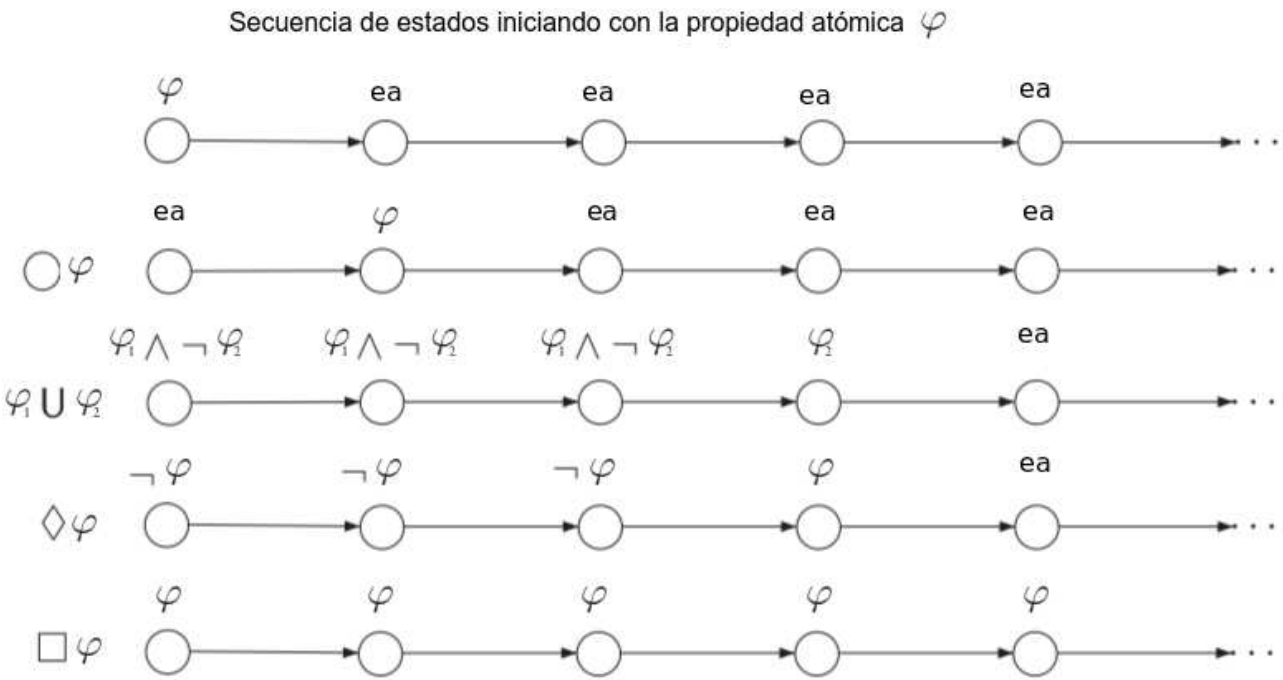


Figura: Representación gráfica de los operadores LLT

Adaptada de [5] y [2]

Sintaxis y Semántica

- Para especificar las propiedades en Lógica Lineal Temporal (LLT) se debe construir una fórmula.

- Estas fórmulas se componen de:
  - Proposiciones Atómicas (PA) representadas como $a_i \in AP$ en donde se establece a_i como una etiqueta de estado (o una **letra** del alfabeto) en el sistema,
  - Conectores booleanos básicos $\wedge, \vee, \neg$ (*and*/y, *or*/o, *not*/negación)
  - Modalidades temporales básicas $\bigcirc, \bigcup$ (*next*/siguiente, *until*/hasta)

- De esta forma, una fórmula LLT se puede expresar a través de la notación Backus-Naur así:

$$\varphi ::= true \,|\, a_i \,|\, \varphi_1 \wedge \varphi_2 \,|\, \neg\varphi \,|\, \bigcirc \varphi \,|\, \varphi_1 \bigcup \varphi_2$$

Reglas de equivalencia

$$\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2) \tag{1}$$

$$\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2 \tag{2}$$

$$\varphi_1 \leftrightarrow \varphi_2 = (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) \tag{3}$$

$$\Diamond\varphi = true \bigcup \varphi \tag{4}$$

$$\Box\varphi = \neg\Diamond\neg\varphi \tag{5}$$

$$\Box\Diamond\varphi = \Box(true \bigcup \varphi) \tag{6}$$

$$\Diamond\Box\varphi = true \bigcup (\Box\varphi) \tag{7}$$

Definiciones

- AP es el conjunto de proposiciones atómicas del sistema, es decir: $AP = \{a_0, a_1, a_2, ..., a_n\}$
- $\mathscr{P}(AP)$: Es el conjunto potencia sobre AP (conjunto formado por todos los subconjuntos de AP), es decir:

$$\mathscr{P}(AP) = \{\{\}, \{a_0\}, \{a_1\}, \{a_2\}, ..., \{a_n\}, ..., \{a_0, a_1\}, \{a_0, a_2\}, ..., \{a_0, .., a_n\}...\{a_1, a_2\}...\}$$

- Una **palabra** es una secuencia de elementos sobre un conjunto. Por ejemplo: una palabra sobre el conjunto AP sería: $a_0 a_2$ o sobre el conjunto $\mathscr{P}(AP)$ sería: $\{a_0\}\{a_1\}\{a_0, a_1\}\{a_2\}$
- AP_{INF}: es el conjunto infinito de **palabras** sobre el conjunto potencia $\mathscr{P}(AP)$. Por ejemplo:

$$AP_{INF} = \{\{a_0\}, \{a_0\}\{a_1\}, \{a_0\}\{a_1\}\{a_0\}, \{a_0\}\{a_1\}\{a_0, a_1\}, ...\}$$

- $Traces(a_i)$: Es el conjunto de caminos cuyo estado inicial es a_i. $Traces(a_i) \subseteq AP_{INF}$
- $Traces(ST)$: Es el conjunto de caminos desde los estados iniciales del sistema de transición STPA. $Traces(ST) \subseteq AP_{INF}$

Ejemplo (I)

Un ejemplo especificación de una propiedad sobre un modelo puede ser: a_1 es verdadera siempre

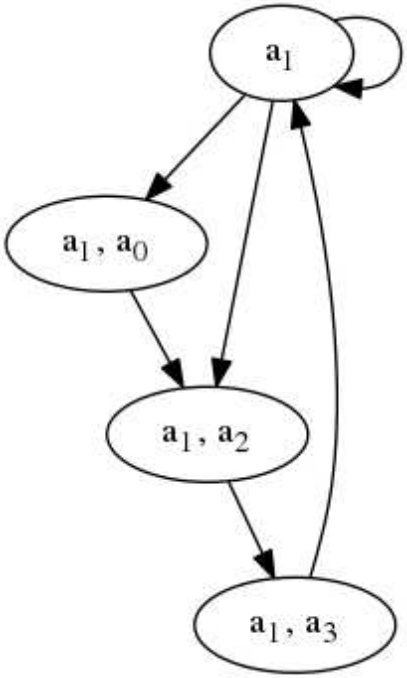


Figura: Ejemplo sistema

Para este caso, un conjunto de palabras de ejemplo que satisface la propiedad, consiste en: $\{\{a_1\},\{a_1\}\{a_1,a_0\},\{a_1,a_2\},\{a_1,a_3\},\{a_1\},...\}$.

Ejemplo (II)

- En este sentido, una **fórmula** especificada en lógica lineal temporal (LLT) describe subconjuntos de AP_{INF}.

- AP_{INF}: es el conjunto infinito de **palabras** sobre el conjunto potencia $\mathscr{P}(AP)$. Por ejemplo:

$$AP_{INF} = \{\{a_0\},\{a_0\}\{a_1\},\{a_0\}\{a_1\}\{a_0\},\{a_0\}\{a_1\}\{a_0,a_1\},...\}$$

- Dada una fórmula LLT φ,
  - se asocia a ella un conjunto de palabras que pueden identificarse con la expresión $Words(\varphi)$
  - sus elementos corresponden a la secuencia de estados alcanzados en cada transición
  - Si φ es una fórmula LLT: $\varphi \longrightarrow Words(\varphi) \subseteq AP_{INF}$

- $Words(\varphi)$ es el conjunto de palabras que satisfacen la fórmula

$$\varphi\text{: } Words(\varphi) = \{\sigma \in AP_{INF} \,|\, \sigma \text{ satisface } \varphi\}$$

Reglas de verificación I

Se tiene la palabra σ : $Word\,\sigma : A_0 A_1 A_2 ... A_n \in AP_{INF}$

- Toda palabra en AP_{INF} satisface *true*.

$$Words(true) = AP_{INF} \tag{1}$$

- σ satisface a_i, si $a_i \in A_0$.

$$Words(a_i) = \{A_0 A_1 A_2 ... \mid a_i \in A_0\} \tag{2}$$

- σ satisface $\varphi_1 \vee \varphi_2$, si σ satisface φ_1 o σ satisface φ_2.

$$Words(\varphi_1 \vee \varphi_2) = Words(\varphi_1) \cup Words(\varphi_2) \tag{3}$$

- σ satisface $\neg\varphi$, si σ no satisface φ.

$$Words(\neg\varphi) = Words(\varphi)' \tag{4}$$

Reglas de verificación II

- σ satisface $\bigcirc\varphi$, si $A_1 A_2 ...$ satisface φ.

$$Words(\bigcirc\varphi) = \{A_0 A_1 A_2 ... | A_1, A_2 \in Words(\varphi)\} \quad (5)$$

- σ satisface $\varphi_1 \bigcup \varphi_2$, si existe un j tal que $A_j A_{j+1} ...$ satisfacen φ_2 y para todos los $0 \leq i < j$ $A_i A_{i+1} ...$ satisfacen φ_1.

$$Words(\varphi_1 \bigcup \varphi_2) = \{A_0 A_1 A_2 ... | \exists j . A_j A_{j+1} ... \in Words(\varphi_2) \quad (6)$$

$$\text{y } \forall 0 \leq i < j, A_i A_{i+1} ... \in Words(\varphi_1)\}$$

Configuración arquitectónica objeto de estudio

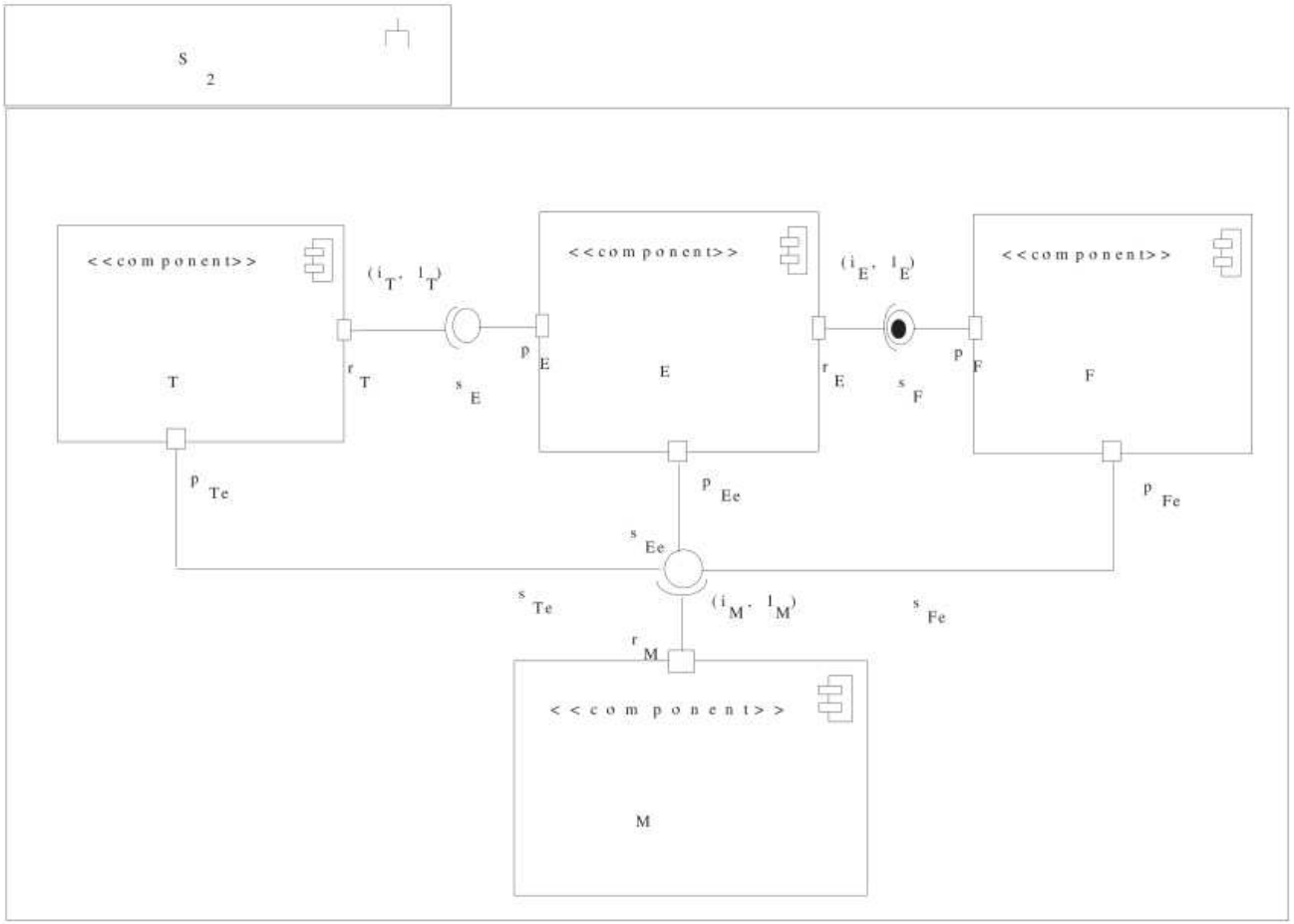


Figura: Ensamble complejo de componentes

Fuente: Tomado de [6]

Traducción al cálculo formal

El conjunto de fórmulas que especifican la arquitectura se define de la siguiente forma:

$$E \stackrel{def}{=} [(p_E : x/xs_E)] \wedge \exists l_E[(r_E :: y/yl_E) \wedge (l_E :: i_E/E^{(int)})]$$
$$F \stackrel{def}{=} (p_F : z/zs_F) \wedge (p_{Fe} : w/ws_{Fe})$$
$$M \stackrel{def}{=} \exists l_M[(r_M :: y/yl_M) \wedge (l_M :: i_M/M^{(int)})]$$
$$T \stackrel{def}{=} [(p_{Te} : n/ns_{Te})] \wedge \exists l_T[(r_T :: q/ql_T) \wedge (l_T :: i_T/T^{(int)})]$$

$$C_F E = r_E \overline{p_F}$$
$$C_F M = r_M \overline{p_F e}$$
$$C_E T = r_T \overline{p_E}$$
$$C_E M = r_M \overline{p_E e}$$
$$C_T M = r_M \overline{p_T e}$$

$$S = [F \wedge OSO(F) do C_F E \wedge E else \wedge C_F M \wedge M] \wedge [OSO(E) do C_E T \wedge T else C_E M \wedge M] \wedge [OSO(T) do T else C_T M \wedge M]$$

Pasos para la especificación

Construir un Sistema de Transición de Proposiciones Atómicas (STPA): Representa los estados del sistema susceptibles de análisis.

- Se realiza a través de la definición de la arquitectura (la especificación de componentes y conexiones) y las reglas de ejecución de la fórmula provista por el cálculo ρ_{arq}.
- **Primer paso**: identificar el componente fuente (único que puede iniciar la ejecución).
- **Segundo paso**: Por cada componente dentro del modelo representar un estado, su ejecución (por ejemplo F) y sus posibles siguientes estados:
  - Representan ejecución exitosa o ejecución de fracaso (por ejemplo, $F^{\top}$ y $F^{\perp}$ respectivamente), (Figura 7)

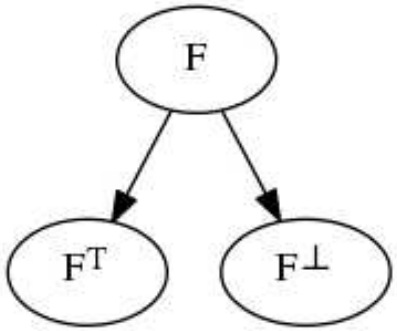


Figura: Ejemplo de representación de la ejecución de un componente en el STPA

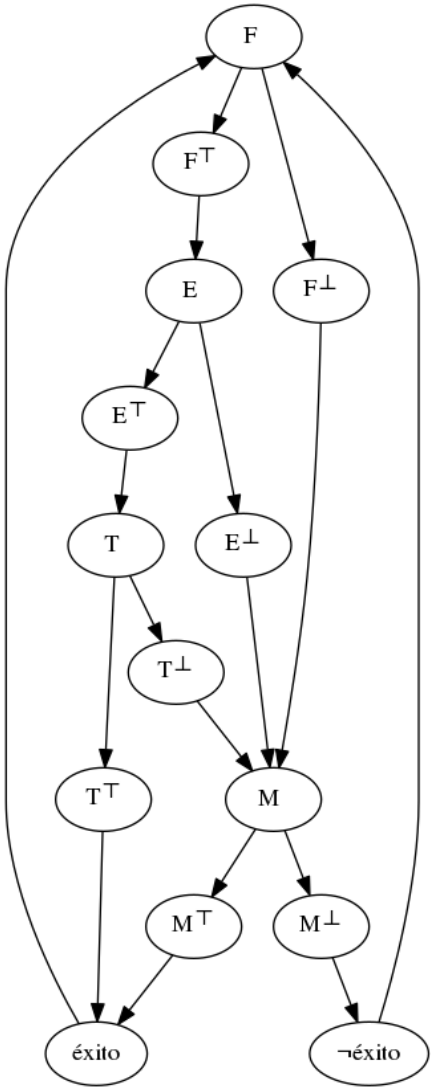
Especificación de una propiedad temporal V

- **Tercer paso** consiste en obtener las transiciones entre estados
  - Capturar comportamiento a través de las reglas de observación
  - Sintaxis: $OSO(F)\,do\,[C_F\,E \wedge E]\,else\,[C_F\,M \wedge M]$
  - En este caso si F se ejecuta correctamente se comunica con E de lo contrario se conecta con M.

- Con la aparición de nuevos componentes se realiza con ellos el primer paso y se va desarrollando el STPA hasta que ya no hayan más componentes por analizar.

- **Cuarto paso**, cuando se llega a los estados finales (componentes sumidero) se indica un estado global del sistema:
  - Este estado solo se puede obtener de los nodos terminales que representa uno de los dos estados finales de éxito o fracaso global (*éxito* o *fracaso*)
  - Se conecta nuevamente con los nodos que representan a los componentes fuente (nueva ejecución)

STPA Resultado



Otros escenarios - Configuración arquitectónica con selección alternativa I

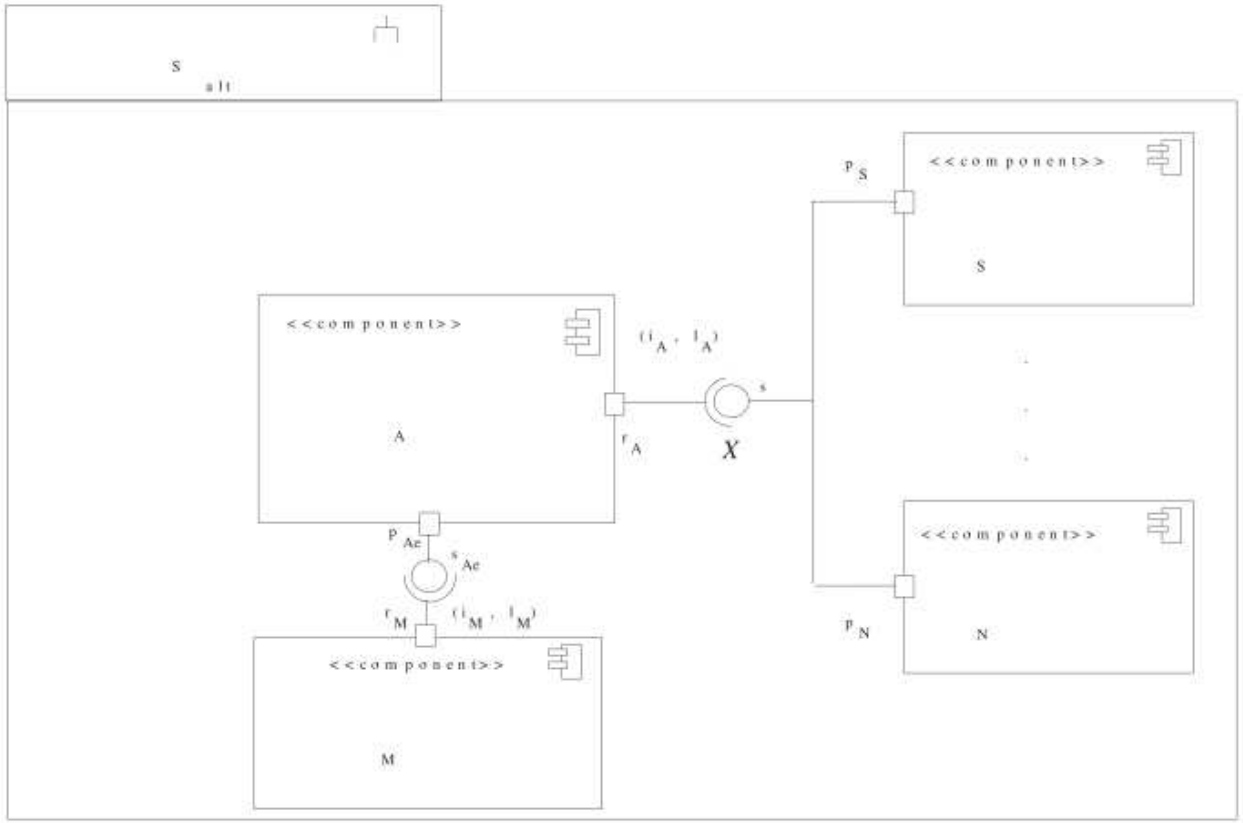


Figura: Configuración componentes alternativos (Combinador)

Fuente: Tomado de [4]

Otros escenarios - Configuración arquitectónica con selección alternativa II

El STPA para este escenario es:

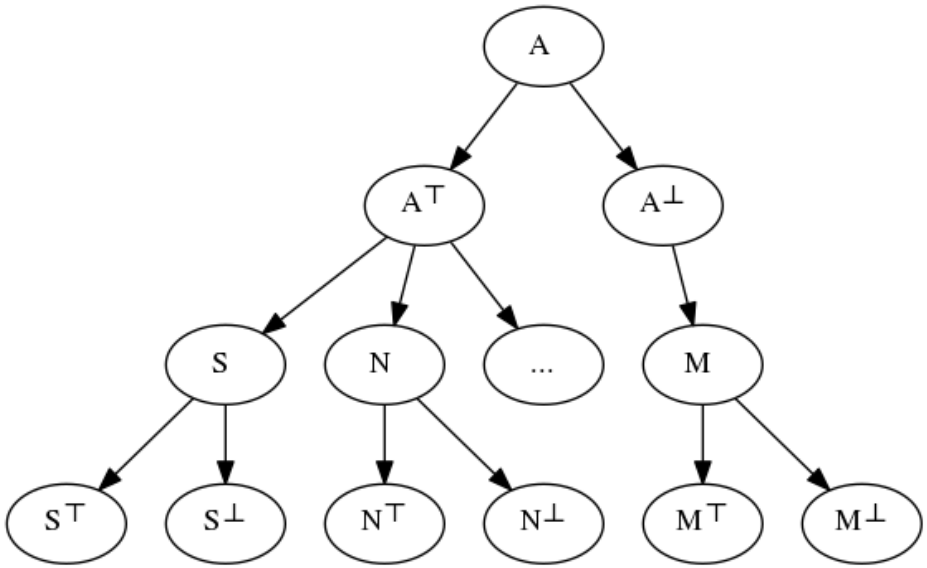


Figura: STPA para una Configuración con componentes alternativos

A partir del STPA definido se pueden especificar propiedades temporales.

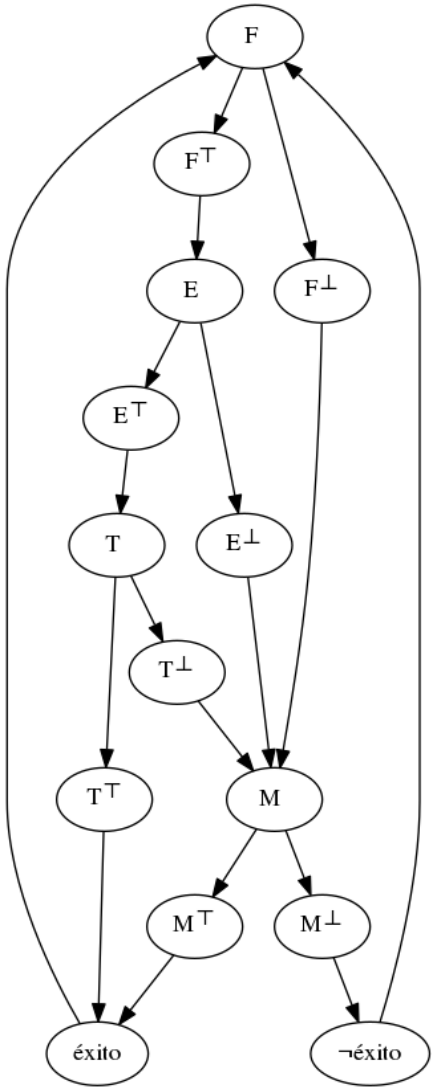
Propiedad temporal objetivo I

- Vivacidad (*liveness*): afirma que "algo bueno" eventualmente ocurrirá o también que un programa eventualmente llegará a un estado deseado.
  - Libre de inanición (*Starvation freedom*)
  - Terminación (*Termination*)
  - **Servicio garantizado / Capacidad de respuesta (*Guaranteed service* / *Responsiveness*)**

$$\varphi = F \rightarrow \Diamond M^{\top}$$

Objeto de estudio

Para el sistema:



Especificación Propiedad temporal objetivo (I)

Se tienen por definición los siguientes conjuntos :
$AP = \{F, F^{\top}, F^{\bot}, E, E^{\top}, E^{\bot}, T, T^{\top}, T^{\bot}, M, M^{\top}, M^{\bot}, éxito, fracaso\}$

$AP_{INF} = \{\{\}, \{F\}, \{F^{\top}\}, \{F^{\bot}\}, \{E\}, \{E^{\top}\}, \{E^{\bot}\}, \{T\}, \{T^{\top}\}, \{T^{\bot}\},$
$\{M\}, \{M^{\top}\}, \{M^{\bot}\}, \{F\}\{F^{\top}\}, \{F\}\{F^{\bot}\}, \{F\}\{F^{\top}\}\{E\}, ...\}$

Aplicando las reglas de composición, la fórmula se puede expresar en subfórmulas que pueden ser evaluadas en la siguiente secuencia:

$Words(\varphi) = Words(F \rightarrow \Diamond M^{\top})$ se aplica $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$
$Words(\varphi) = Words(\neg F \vee \Diamond M^{\top})$ se aplica $Words(\varphi_1 \vee \varphi_2) = Words(\varphi_1) \cup Words(\varphi_2)$
$Words(\varphi) = Words(\neg F) \cup Words(\Diamond M^{\top})$ se aplica $Words(\neg\varphi) = Words(\varphi)'$
$Words(\varphi) = Words(F)' \cup Words(\Diamond M^{\top})$ se aplica $\Diamond\varphi = true \bigcup \varphi$
$Words(\varphi) = Words(F)' \cup Words(true \bigcup M^{\top})$ se describe por extensión el conjunto del primer término:
$Words(\varphi) = \{\{F\}, \{F\}\{F^{\bot}\}, \{F\}\{F^{\bot}\}\{M\}, \{F\}\{F^{\bot}\}\{M\}\{M^{\top}\},$
$\{F\}\{F^{\bot}\}\{M\}\{M^{\top}\}\{éxito\}, \{F\}\{F^{\bot}\}\{M\}\{M^{\bot}\},$
$\{F\}\{F^{\bot}\}\{M\}\{M^{\bot}\}\{\neg éxito\}\} \cup Words(true \bigcup M^{\top})$
...

- El proceso continúa recursivamente hasta encontrar los conjuntos de secuencias de estados (palabras) que satisfacen la propiedad.

Propiedad temporal objetivo (II)

- Finalmente se debe satisfacer que $Traces(TS) \cap Words(\varphi)$,

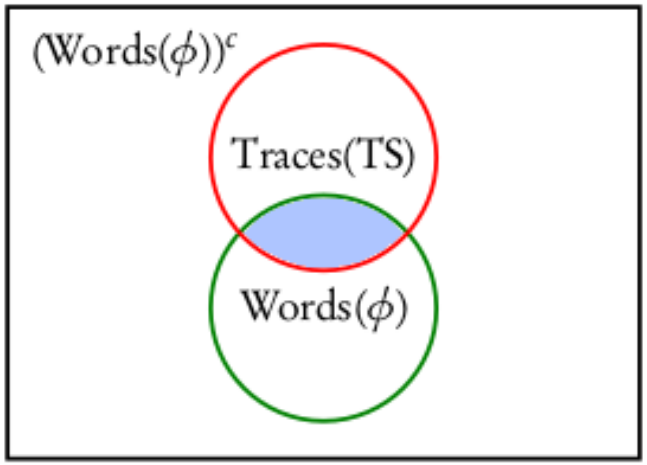


Figura: Relación entre $Traces(TS)$ y $Words(\varphi)$

Fuente [7]

$$Traces(TS) \cap Words(\varphi) = \{\{F\}, \{F\}\{F^{\perp}\}, \{F\}\{F^{\perp}\}\{M\}, \{F\}\{F^{\perp}\}\{M\}\{M^{\top}\}\}$$

Modelo Funcional

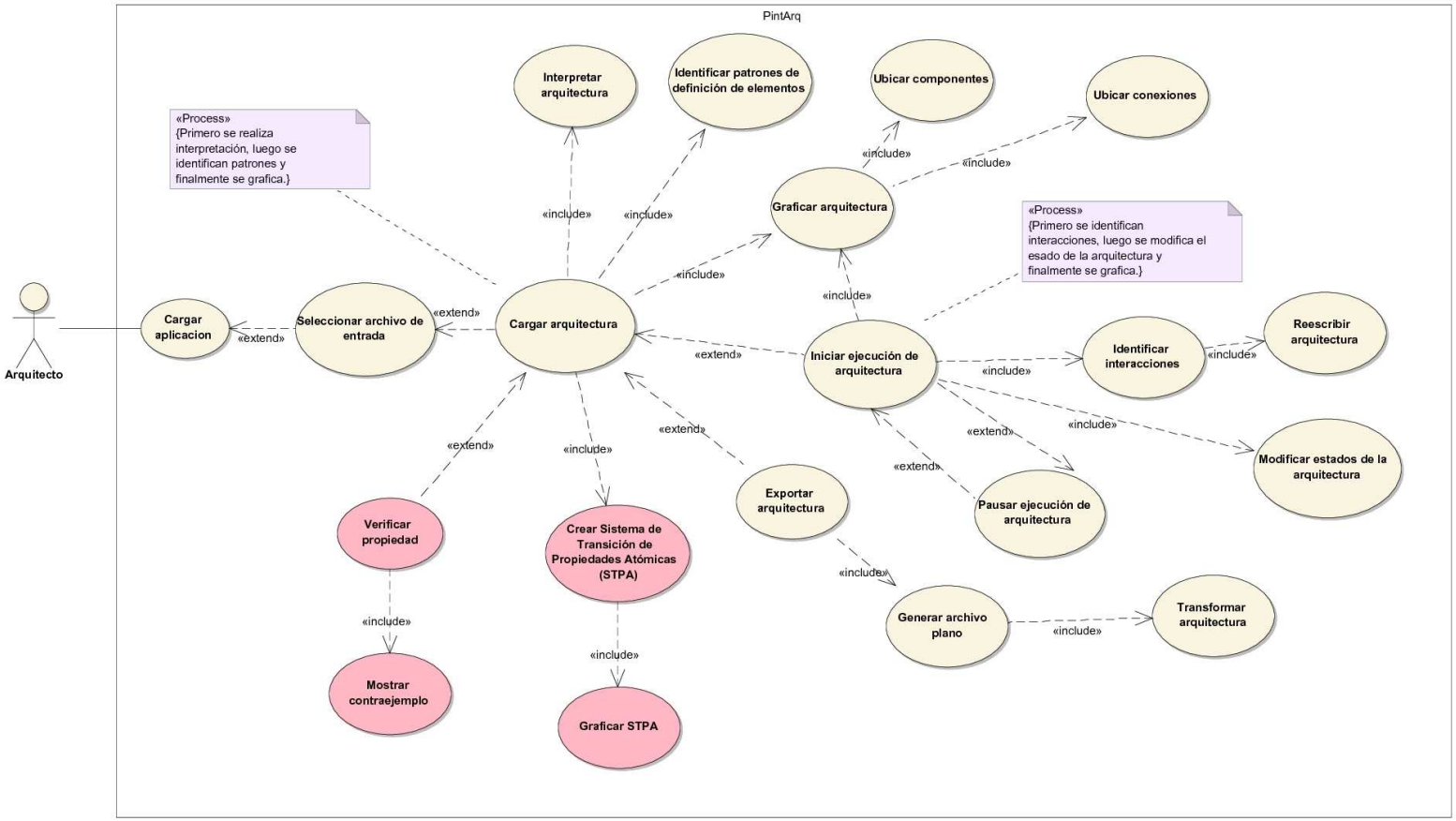


Figura: Modelo funcional de PintArq extendido.

Arquitectura prescriptiva extendida

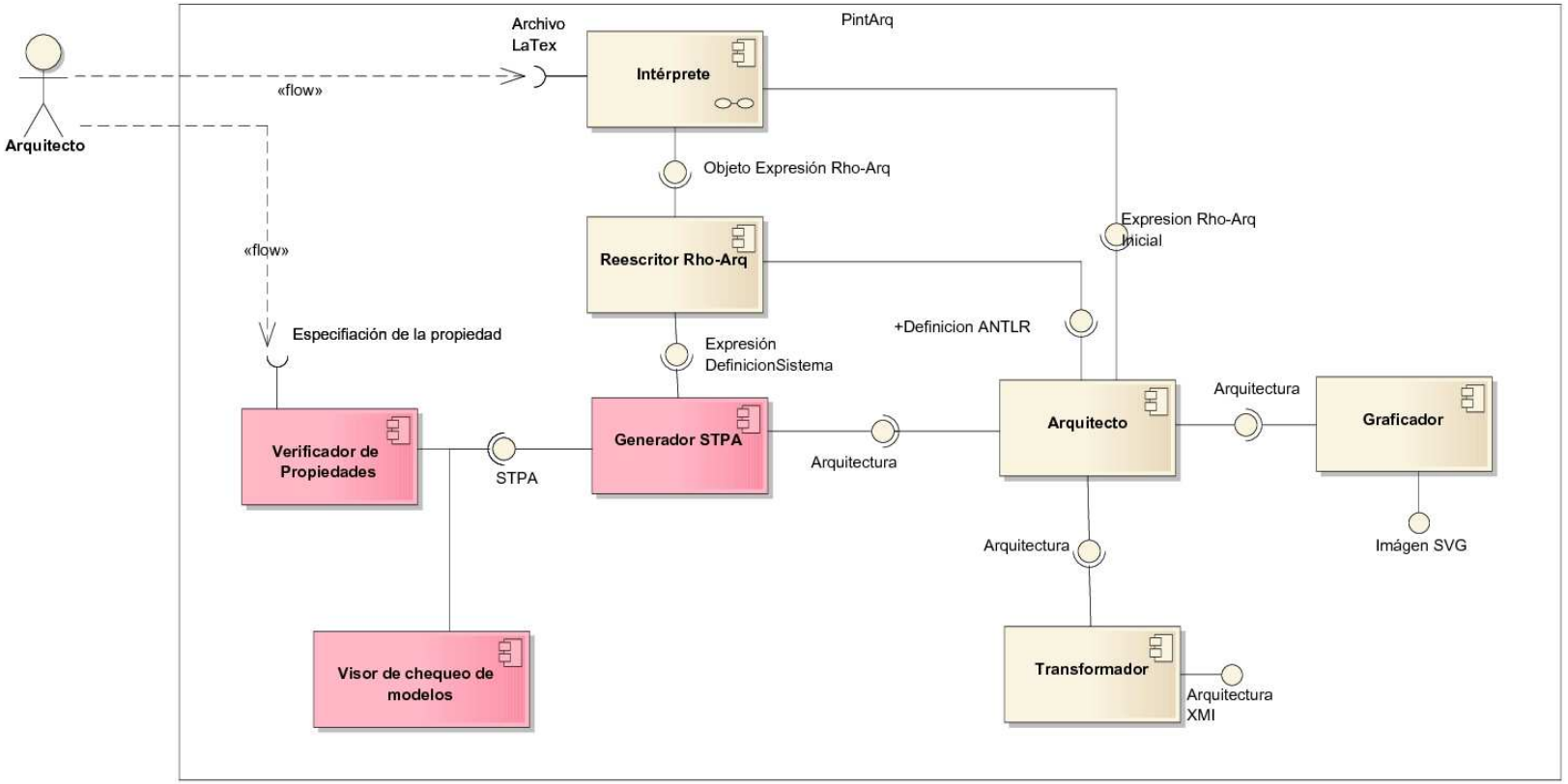


Figura: Arquitectura prescriptiva con los componentes adicionales.

Breve descripción de interfaz gráfica de usuario

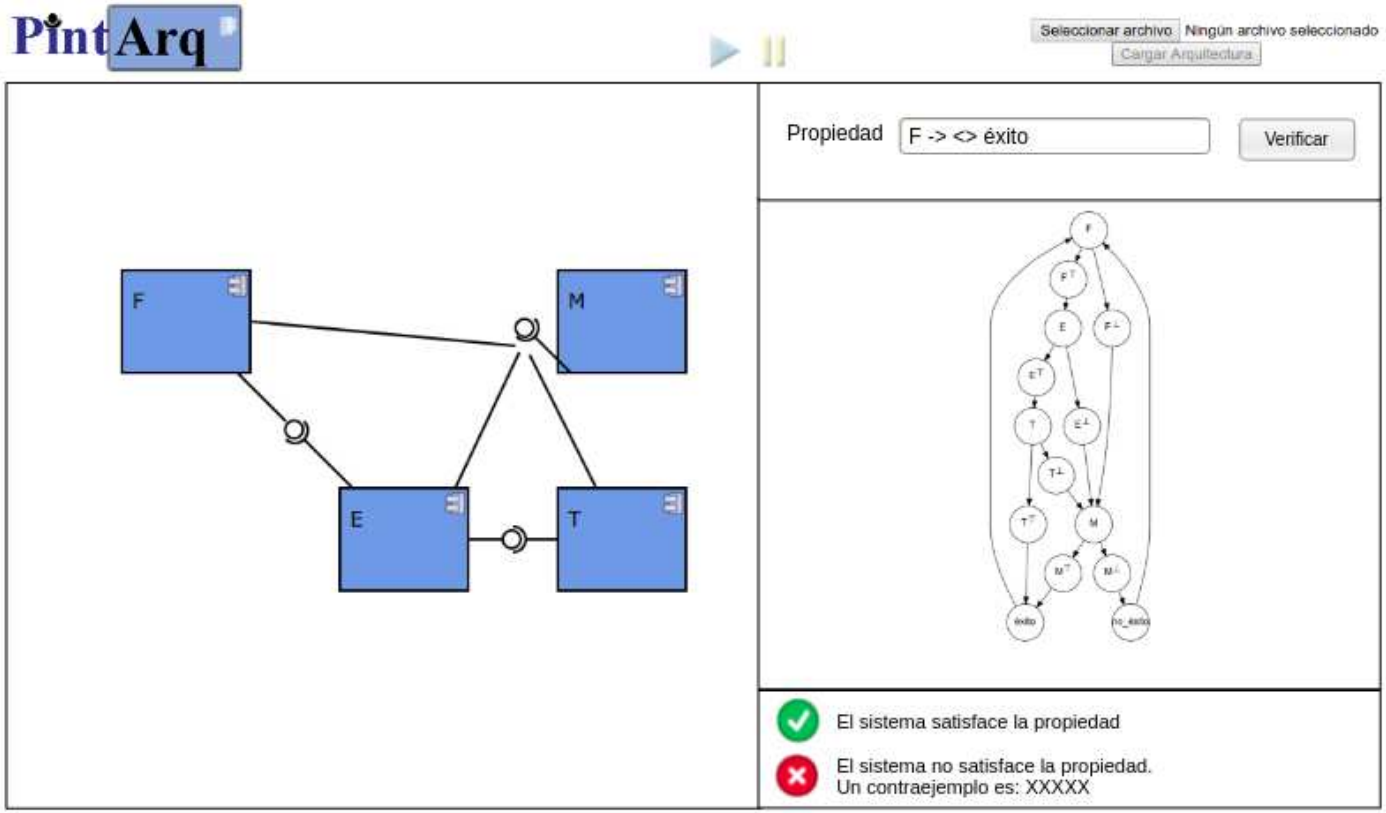


Figura: Prototipo de Interfaz Gráfica de Usuario.

Creación y visualización del STPA I

```
1  \documentclass[10pt,letterpaper,fleqn]{article}
2  \usepackage[utf8]{inputenc}
3  \usepackage{amsmath}
4  \usepackage{amsfonts}
5  \usepackage{amssymb}
6  \author{Alejandro Rico}
7  \title{Prueba Rho-arq}
8  \begin{document}
9  \begin{equation*}
10     S= F^{\top} \wedge E^{\top} \wedge T^{\bot} \wedge F \wedge [OSO (F) do C_FE \wedge E else C_FM \wedge M
       ] \wedge [OSO (E) do C_ET \wedge T else C_EM \wedge M] \wedge [OSO (T) do T else C_TM \wedge M]
11 \end{equation*}
12
```

Figura: Fragmento Documento LaTeX con reglas de observación OSO

Creación y visualización del STPA II

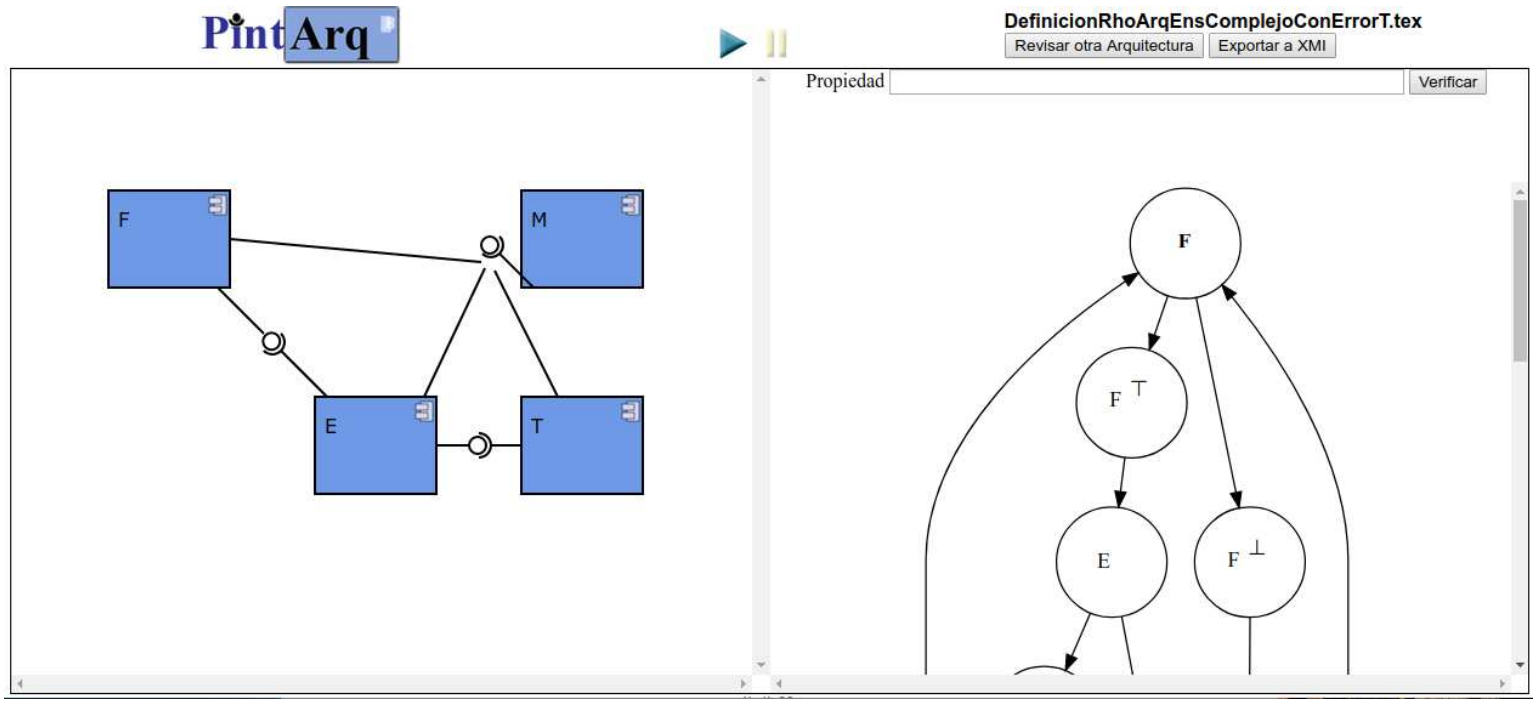


Figura: Visualización de la configuración arquitectural y el STPA en la extensión

Creación y visualización del STPA III

```
digraph GraficadorSTPA848447096 {

    graph [size="16,12", ranksep=.25];
    node [fontname=Arial, fontsize=14, shape=circle, fixedsize=true];

    E_EstadoEjecucionComponente [label=<E> ];
    F_EstadoEjecucionComponente [label=<<B>F</B>> ];{ rank=source; F_EstadoEjecucionComponente }
    T_EstadoEjecucionExitosa [label=<T<SUP>&#8868;</SUP>> ];
    M_EstadoEjecucionComponente [label=<M> ];
    M_EstadoEjecucionExitosa [label=<M<SUP>&#8868;</SUP>> ];
    T_EstadoEjecucionNoExitosa [label=<T<SUP>&#8869;</SUP>> ];
    éxito_Estado [label=<éxito> ];
    T_EstadoEjecucionComponente [label=<T> ];
    no_éxito_Estado [label=<no_éxito> ];
    F_EstadoEjecucionExitosa [label=<F<SUP>&#8868;</SUP>> ];
    M_EstadoEjecucionNoExitosa [label=<M<SUP>&#8869;</SUP>> ];
    E_EstadoEjecucionExitosa [label=<E<SUP>&#8868;</SUP>> ];
    F_EstadoEjecucionNoExitosa [label=<F<SUP>&#8869;</SUP>> ];
    E_EstadoEjecucionNoExitosa [label=<E<SUP>&#8869;</SUP>> ];

    E_EstadoEjecucionComponente -> E_EstadoEjecucionExitosa [   ]
    E_EstadoEjecucionComponente -> E_EstadoEjecucionNoExitosa [   ]
    T_EstadoEjecucionComponente -> T_EstadoEjecucionExitosa [   ]
    T_EstadoEjecucionComponente -> T_EstadoEjecucionNoExitosa [   ]
    F_EstadoEjecucionComponente -> F_EstadoEjecucionExitosa [   ]
    F_EstadoEjecucionComponente -> F_EstadoEjecucionNoExitosa [   ]
    éxito_Estado -> F_EstadoEjecucionComponente [   ]
    no_éxito_Estado -> F_EstadoEjecucionComponente [   ]
    M_EstadoEjecucionComponente -> M_EstadoEjecucionExitosa [   ]
    M_EstadoEjecucionComponente -> M_EstadoEjecucionNoExitosa [   ]
    M_EstadoEjecucionExitosa -> éxito_Estado [   ]
    M_EstadoEjecucionNoExitosa -> no_éxito_Estado [   ]
    F_EstadoEjecucionExitosa -> E_EstadoEjecucionComponente [   ]
    F_EstadoEjecucionNoExitosa -> M_EstadoEjecucionComponente [   ]
    E_EstadoEjecucionExitosa -> T_EstadoEjecucionComponente [   ]
    E_EstadoEjecucionNoExitosa -> M_EstadoEjecucionComponente [   ]
    T_EstadoEjecucionExitosa -> éxito_Estado [   ]
    T_EstadoEjecucionNoExitosa -> M_EstadoEjecucionComponente [   ]

}
```

Figura: Representación de un STPA en lenguaje DOT

Verificación de propiedades (SPIN) - satisface

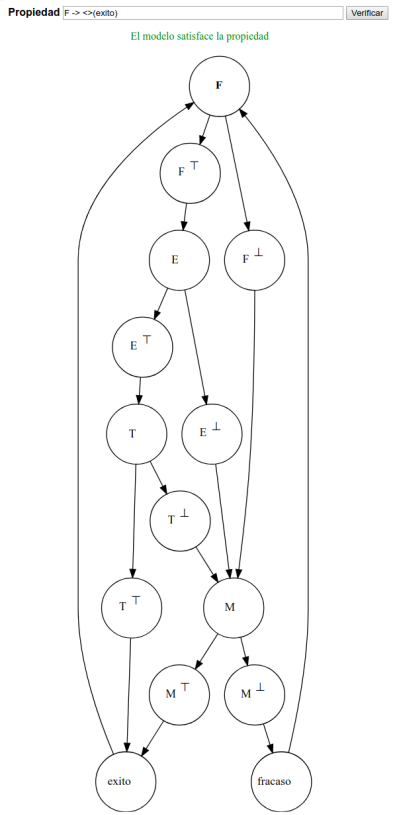


Figura: Verificación de una propiedad temporal. El modelo satisface la propiedad

Verificación de propiedades (SPIN) - no satisface

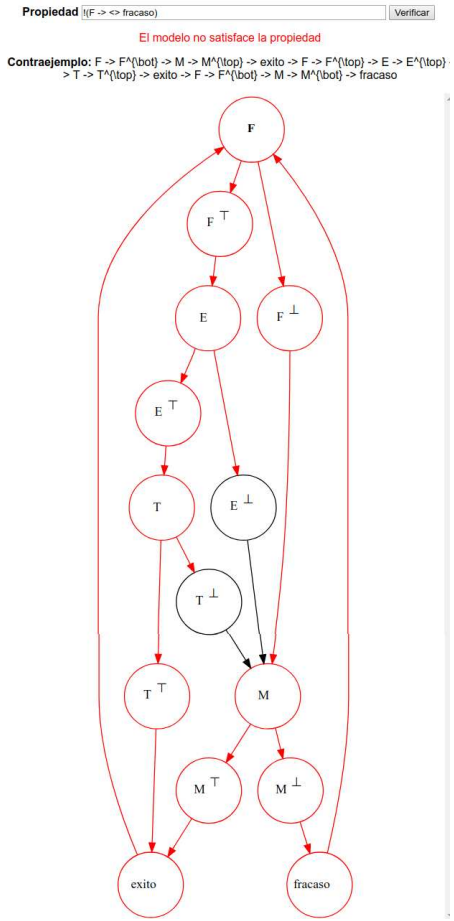


Figura: Verificación de una propiedad temporal. El modelo no satisface la propiedad

Verificación de propiedades (SPIN) - Promela

```
mtype = {
    T_EstadoEjecucionExitosa,
    E_EstadoEjecucionComponente,
    F_EstadoEjecucionComponente,
    fracaso,
    F_EstadoEjecucionExitosa,
    E_EstadoEjecucionExitosa,
    M_EstadoEjecucionComponente,
    E_EstadoEjecucionNoExitosa,
    T_EstadoEjecucionNoExitosa,
    M_EstadoEjecucionNoExitosa,
    F_EstadoEjecucionNoExitosa,
    T_EstadoEjecucionComponente,
    M_EstadoEjecucionExitosa,
    exito
};
mtype estado = F_EstadoEjecucionComponente
int turno = 0
#define maxE_EstadoEjecucionComponente 2
mtype estado_t_E_EstadoEjecucionComponente[maxE_EstadoEjecucionComponente];
int index_E_EstadoEjecucionComponente = 0
#define maxF_EstadoEjecucionComponente 2
mtype estado_t_F_EstadoEjecucionComponente[maxF_EstadoEjecucionComponente];
int index_F_EstadoEjecucionComponente = 0
#define maxM_EstadoEjecucionComponente 2
mtype estado_t_M_EstadoEjecucionComponente[maxM_EstadoEjecucionComponente];
int index_M_EstadoEjecucionComponente = 0
#define maxT_EstadoEjecucionComponente 2
mtype estado_t_T_EstadoEjecucionComponente[maxT_EstadoEjecucionComponente];
int index_T_EstadoEjecucionComponente = 0

/** Declaración de la propiedad que debe ser satisfecha por el sistema **/
#define p0 (estado == F_EstadoEjecucionComponente)
#define p1 (estado == M_EstadoEjecucionExitosa)
#define p2 (estado == T_EstadoEjecucionExitosa)
#define p3 (estado == E_EstadoEjecucionExitosa)
ltl propiedad { p0-><>(p1 || p2 || p3) }
```

Figura: Representación en Promela de un STPA. (Fragmento)

Conclusiones

1. El mayor esfuerzo del proyecto estuvo en la abstracción de la ejecución de la arquitectura a través del Sistema de Transición de Proposiciones Atómicas (STPA) que es sobre el cual se puede razonar sobre el comportamiento de la arquitectura.
2. Se logró implementar una extensión de la aplicación PintArq para la visualización del modelo que representa el comportamiento de la arquitectura y sobre el que se puede hacer su análisis a través de lógicas temporales.
3. Contar con la herramienta PintArq y su documentación y modelos fue de vital importancia para implementar de forma mas rápida y precisa el mecanismo formal de especificación de propiedades temporales.
4. Las herramientas de software libre y abierto bien documentadas y modulares permiten incorporar y expandir proyectos sin necesidad de construir todos sus elementos desde cero.
5. La carga conceptual que conlleva investigar temas relacionados con métodos formales en ingeniería de software es insuficiente en la Maestría, especialmente para comprender con mayor rapidez temas como las lógicas temporales pues no son temas de estudio regular en las asignaturas del proyecto curricular.

Trabajo futuro

A través del proceso de investigación se identificaron varios escenarios sobre los que se pueden iniciar o continuar trabajos relacionados:

1. Implementar una herramienta de software propia de verificación de propiedades temporales para el cálculo ρ_{arq}.
2. Ampliar el alcance del mecanismo para verificar propiedades temporales como Fiabilidad (*Safety*), Detección de abrazos mortales o aseguramiento de libertad de inanición.
3. Optimizar la implementación para encontrar el camino mas corto que ilustra el contraejemplo cuando el modelo no satisface la propiedad especificada.
4. Mejorar la implementación de la extensión en PintArq para visualizar el contraejemplo en la sintaxis del cálculo formal y no en formato LaTeX.

Gracias por su atención.

Preguntas, comentarios

Referencias


Dimitra Giannakopoulou,
Model Checking for Concurrent Software Architectures,
PhD thesis, 1999.


Christel Baier and Joost-Pieter Katoen,
Principles of Model Checking,
2008.


Joachim Niehren and Gert Smolka,
"A Confluent Relational Calculus for Higher-Order Programming with Constraints",
2010.


Henry Diosa, Juan Francisco Díaz Frías, and Carlos Mauricio Gaona,
"Especificación formal de arquitecturas de software basadas en componentes: chequeo de corrección con cálculo rho-arq",
, no. 12, 2010.


Edmund Clarke,
"Model Checking", 2000.


Henry Diosa, Juan Francisco Díaz Frías, and Carlos Mauricio Gaona,
"Cálculo para el modelado formal de arquitecturas de software basadas en componentes: cálculo r-Arq", 2009.