A Model-Checking Tool for Families of Services

M.H. ter Beek
ISTI–CNR, Pisa, Italy

joint work with

P. Asirelli
ISTI–CNR

A. Fantechi
University of Florence

S. Gnesi
ISTI–CNR

FMOODS/FORTE 2011

Reykjavík, Ísland
8 June 2011

Outline

Aim of our research activity at large

Emerging topic in engineering distributed systems

Synergy between (Software) Product Line Engineering (SPLE) and Service-Oriented Computing (SOC)

Aim & foresight

Rigorous modelling techniques and analysis tools for the systematic, large-scale provision and market segmentation of *software services*

Design techniques for *software service line organizations* to develop novel classes of applications well adaptable to customer requirements as well as to changes in the context in which, and while, they execute

Outcome

Techniques and support tools to assist organizations to plan, optimize, and control the quality of software service provision at design-/run-time

Aim of our research activity at large

Emerging topic in engineering distributed systems

Synergy between (Software) Product Line Engineering (SPLE) and Service-Oriented Computing (SOC)

Aim & foresight

Rigorous modelling techniques and analysis tools for the systematic, large-scale provision and market segmentation of *software services*

Design techniques for *software service line organizations* to develop novel classes of applications well adaptable to customer requirements as well as to changes in the context in which, and while, they execute

Outcome

Techniques and support tools to assist organizations to plan, optimize, and control the quality of software service provision at design-/run-time

Aim of our research activity at large

Emerging topic in engineering distributed systems

Synergy between (Software) Product Line Engineering (SPLE) and Service-Oriented Computing (SOC)

Aim & foresight

Rigorous modelling techniques and analysis tools for the systematic, large-scale provision and market segmentation of *software services*

Design techniques for *software service line organizations* to develop novel classes of applications well adaptable to customer requirements as well as to changes in the context in which, and while, they execute

Outcome

Techniques and support tools to assist organizations to plan, optimize, and control the quality of software service provision at design-/run-time

Product Line Engineering (PLE)

Paradigm

To develop a family of products using a common platform and mass customization

Aim

To lower production costs of the individual products by

- letting them share an overall reference model of the product family
- allowing them to differ w.r.t. particular characteristics to serve, e.g., different markets

Product variants are derivable from a product family, thus allowing for reuse and differentiation

Production process

Organized so as to maximize commonalities of the products and at the same time minimize the cost of variations

Product Line Engineering (PLE)

Paradigm

To develop a family of products using a common platform and mass customization

Aim

To lower production costs of the individual products by

- letting them share an overall reference model of the product family
- allowing them to differ w.r.t. particular characteristics to serve, e.g., different markets

Product variants are derivable from a product family, thus allowing for reuse and differentiation

Production process

Organized so as to maximize commonalities of the products and at the same time minimize the cost of variations

Product Line Engineering (PLE)

Paradigm

To develop a family of products using a common platform and mass customization

Aim

To lower production costs of the individual products by

- letting them share an overall reference model of the product family
- allowing them to differ w.r.t. particular characteristics to serve, e.g., different markets

Product variants are derivable from a product family, thus allowing for reuse and differentiation

Production process

Organized so as to maximize commonalities of the products and at the same time minimize the cost of variations

Variability management

Feature modelling

Provide compact representations of all the products of a product family (product line) in terms of their *features*

Variability modelling

How to explicitly define the features or components of a product family that are **optional**, **alternative**, **mandatory**, **required**, or **excluded** as *variation points*

Managing variability with formal methods

Show that a certain product belongs to a product family or, instead, derive one from a family by properly selecting features/components

Variability management is the key aspect differentiating SPLE from 'conventional' software engineering

Variability management

Feature modelling

Provide compact representations of all the products of a product family (product line) in terms of their *features*

Variability modelling

How to explicitly define the features or components of a product family that are **optional**, **alternative**, **mandatory**, **required**, or **excluded** as *variation points*

Managing variability with formal methods

Show that a certain product belongs to a product family or, instead, derive one from a family by properly selecting features/components

Variability management is the key aspect differentiating SPLE from 'conventional' software engineering

Variability management

Feature modelling

Provide compact representations of all the products of a product family (product line) in terms of their *features*

Variability modelling

How to explicitly define the features or components of a product family that are **optional**, **alternative**, **mandatory**, **required**, or **excluded** as *variation points*

Managing variability with formal methods

Show that a certain product belongs to a product family or, instead, derive one from a family by properly selecting features/components

Variability management is the key aspect differentiating SPLE from 'conventional' software engineering

Service Product Lines

Service-Oriented Computing (SOC)

Distributed computing paradigm to develop loosely-coupled, evolvable, interoperable systems/applications, exploiting Internet's pervasiveness

Recent successful research effort

1. investigate existing modelling structures that allow (behavioural) variability to be described and product derivation to be defined
2. develop a temporal logic that is interpreted over such structures and that can express properties over families and products alike

Series of recent publications in ACOTA @ ASE 2010, IFM 2010, PLEASE @ ICSE 2011, SEW-34 @ FM 2011, SPLC 2011

Present a step towards extending our results to service families

Service Product Lines

Service-Oriented Computing (SOC)

Distributed computing paradigm to develop loosely-coupled, evolvable, interoperable systems/applications, exploiting Internet's pervasiveness

Recent successful research effort

1. investigate existing modelling structures that allow (behavioural) variability to be described and product derivation to be defined
2. develop a temporal logic that is interpreted over such structures and that can express properties over families and products alike

Series of recent publications in ACOTA @ ASE 2010, IFM 2010, PLEASE @ ICSE 2011, SEW-34 @ FM 2011, SPLC 2011

Present a step towards extending our results to service families

Service Product Lines

Service-Oriented Computing (SOC)

Distributed computing paradigm to develop loosely-coupled, evolvable, interoperable systems/applications, exploiting Internet's pervasiveness

Recent successful research effort

1. investigate existing modelling structures that allow (behavioural) variability to be described and product derivation to be defined
2. develop a temporal logic that is interpreted over such structures and that can express properties over families and products alike

Series of recent publications in ACOTA @ ASE 2010, IFM 2010, PLEASE @ ICSE 2011, SEW-34 @ FM 2011, SPLC 2011

Present a step towards extending our results to service families

Running example: Travel agency family

Software company sells a package to start a travel agency web service

Provides choice among products of family with different prices/features

All products provide features *hotel*/*flight*/*train* reservation: coordination component uses *predefined* external services to retrieve list of quotes

These products can be enhanced in two ways:

1. By adding as *alternative* feature the possibility to choose, only for flights and hotels, from *multiple* external services in order to retrieve the best quotes through more than one service
2. By adding as *optional* feature the possibility for a customer to book a *leisure tour* during his/her stay at a hotel; as the provided tour packages may include a hotel in a different location for a subset of nights, a tour reservation *requires* interaction with the hotel service to offer a feature that allows to *cancel part* of the room reservations at the main hotel location for such nights

Running example: Travel agency family

Software company sells a package to start a travel agency web service

Provides choice among products of family with different prices/features

All products provide features *hotel*/*flight*/*train* reservation: coordination component uses *predefined* external services to retrieve list of quotes

These products can be enhanced in two ways:

1. By adding as *alternative* feature the possibility to choose, only for flights and hotels, from *multiple* external services in order to retrieve the best quotes through more than one service
2. By adding as *optional* feature the possibility for a customer to book a *leisure tour* during his/her stay at a hotel; as the provided tour packages may include a hotel in a different location for a subset of nights, a tour reservation *requires* interaction with the hotel service to offer a feature that allows to *cancel part* of the room reservations at the main hotel location for such nights

Running example: Travel agency products

When combined, this choice of features leads to 8 different products

features		variability	products							
			1	2	3	4	5	6	7	8
train res.	predef. serv.	mandatory	√	√	√	√	√	√	√	√
hotel res.	predef. serv.	alternative	√	√			√	√		
	multiple serv.				√	√			√	√
flight res.	predef. serv.	alternative	√			√	√			√
	multiple serv.			√	√			√	√	
leisure tour reservation		optional					√	√	√	√
cancel part reservation		requires					√	√	√	√

Modal Transition Systems (MTSs)

Originally introduced by Larsen & Thomsen @ LICS 1988

MTSs are now an accepted model to formalize a product family's

- *underlying behaviour*, shared among all products, and
- *variation points*, differentiating between products

MTS is an LTS that distinguishes between may and must transitions (modelling *optional* or *mandatory* features, resp.)

MTS cannot model variability constraints regarding *alternative* features (only one may be present) nor those regarding inter-feature relations (a feature's presence *requires* or *excludes* that of another feature)

We will model such advanced variability constraints by means of an associated set of logical formulae expressed in the variability and action-based branching-time temporal logic VACTL

Modal Transition Systems (MTSs)

Originally introduced by Larsen & Thomsen @ LICS 1988

MTSs are now an accepted model to formalize a product family's

- *underlying behaviour*, shared among all products, and
- *variation points*, differentiating between products

MTS is an LTS that distinguishes between may and must transitions (modelling *optional* or *mandatory* features, resp.)

MTS cannot model variability constraints regarding *alternative* features (only one may be present) nor those regarding inter-feature relations (a feature's presence *requires* or *excludes* that of another feature)

We will model such advanced variability constraints by means of an associated set of logical formulae expressed in the variability and action-based branching-time temporal logic VACTL

Modal Transition Systems (MTSs)

Originally introduced by Larsen & Thomsen @ LICS 1988

MTSs are now an accepted model to formalize a product family's

- *underlying behaviour*, shared among all products, and
- *variation points*, differentiating between products

MTS is an LTS that distinguishes between may and must transitions (modelling *optional* or *mandatory* features, resp.)

MTS cannot model variability constraints regarding *alternative* features (only one may be present) nor those regarding inter-feature relations (a feature's presence *requires* or *excludes* that of another feature)

We will model such advanced variability constraints by means of an associated set of logical formulae expressed in the variability and action-based branching-time temporal logic VACTL

Definition of MTS

$(Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ is an MTS with

- *underlying* LTS $(Q, A, \overline{q}, \delta^{\Box} \cup \delta^{\Diamond})$
- *may* transition relation $\delta^{\Diamond} \subseteq Q \times A \times Q$ (*possible* transitions)
- *must* transition relation $\delta^{\Box} \subseteq Q \times A \times Q$ (*mandatory* transitions)

By definition, mandatory transitions must also be possible: $\delta^{\Box} \subseteq \delta^{\Diamond}$

Reasoning on 3-valued logic; truth values *true*, *false* and *unknown*

The set of all must paths from q_1 is denoted by $\Box$-$path(q_1)$

$\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$ is a *must path* (from q_1), denoted by $\sigma^{\Box}$, if $q_i \xrightarrow{a_i}_{\Box} q_{i+1}$, for all $i > 0$

Subfamilies/products obtained by preserving **all** must transitions, turning **some** may transitions into must transtions, and removing **some**/**all** remaining ones

Definition of MTS

$(Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ is an MTS with

- *underlying* LTS $(Q, A, \overline{q}, \delta^{\Box} \cup \delta^{\Diamond})$
- *may* transition relation $\delta^{\Diamond} \subseteq Q \times A \times Q$ (*possible* transitions)
- *must* transition relation $\delta^{\Box} \subseteq Q \times A \times Q$ (*mandatory* transitions)

By definition, mandatory transitions must also be possible: $\delta^{\Box} \subseteq \delta^{\Diamond}$

Reasoning on 3-valued logic; truth values *true*, *false* and *unknown*

The set of all must paths from q_1 is denoted by $\Box$-*path*(q_1)

$\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$ is a *must path* (from q_1), denoted by $\sigma^{\Box}$, if $q_i \xrightarrow{a_i}_{\Box} q_{i+1}$, for all $i > 0$

Subfamilies/products obtained by preserving **all** must transitions, turning **some** may transitions into must transtions, and removing **some**/**all** remaining ones

Definition of MTS

$(Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ is an MTS with

- *underlying* LTS $(Q, A, \overline{q}, \delta^{\Box} \cup \delta^{\Diamond})$
- *may* transition relation $\delta^{\Diamond} \subseteq Q \times A \times Q$ (*possible* transitions)
- *must* transition relation $\delta^{\Box} \subseteq Q \times A \times Q$ (*mandatory* transitions)

By definition, mandatory transitions must also be possible: $\delta^{\Box} \subseteq \delta^{\Diamond}$

Reasoning on 3-valued logic; truth values *true*, *false* and *unknown*

The set of all must paths from q_1 is denoted by $\Box$-*path*(q_1)

$\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$ is a *must path* (from q_1), denoted by $\sigma^{\Box}$, if $q_i \xrightarrow{a_i}_{\Box} q_{i+1}$, for all $i > 0$

Subfamilies/products obtained by preserving **all** must transitions, turning **some** may transitions into must transtions, and removing **some**/**all** remaining ones

Definition of vACTL

Variability and action-based branching-time temporal logic

A temporal logic based on the "Hennessy-Milner logic with until", but augmented with deontic O (*obligatory*) and P (*permitted*) operators, CTL's path operators E and A and ACTL's action-based Until operator, both with and without a deontic interpretation

Syntax of vACTL

$$\phi ::= \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi' \mid \langle a \rangle\,\phi \mid [a]\,\phi \mid \langle a \rangle^{\Box}\,\phi \mid [a]^{\Box}\,\phi \mid E\,\pi \mid A\,\pi$$

$$\pi ::= \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi' \mid \phi\ \{\varphi\}\ U^{\Box}\ \{\varphi'\}\ \phi'$$

Thus defines state formulae ϕ, path formulae π and action formulae φ (boolean compositions of actions) over set of atomic actions $\{a, b, \ldots\}$

$\langle a \rangle^{\Box}$ and $[a]^{\Box}$ represent the classic deontic modalities O and P, resp.

Definition of VACTL

Variability and action-based branching-time temporal logic

A temporal logic based on the "Hennessy-Milner logic with until", but augmented with deontic O (*obligatory*) and P (*permitted*) operators, CTL's path operators E and A and ACTL's action-based Until operator, both with and without a deontic interpretation

Syntax of VACTL

$$\phi ::= \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi' \mid \langle a \rangle\,\phi \mid [a]\,\phi \mid \langle a \rangle^{\Box}\,\phi \mid [a]^{\Box}\,\phi \mid E\,\pi \mid A\,\pi$$

$$\pi ::= \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi' \mid \phi\ \{\varphi\}\ U^{\Box}\ \{\varphi'\}\ \phi'$$

Thus defines state formulae ϕ, path formulae π and action formulae φ (boolean compositions of actions) over set of atomic actions $\{a, b, \ldots\}$

$\langle a \rangle^{\Box}$ and $[a]^{\Box}$ represent the classic deontic modalities O and P, resp.

Definition of VACTL

Variability and action-based branching-time temporal logic

A temporal logic based on the "Hennessy-Milner logic with until", but augmented with deontic O (*obligatory*) and P (*permitted*) operators, CTL's path operators E and A and ACTL's action-based Until operator, both with and without a deontic interpretation

Syntax of VACTL

$$\phi ::= \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi' \mid \langle a \rangle\,\phi \mid [a]\,\phi \mid \langle a \rangle^{\Box}\,\phi \mid [a]^{\Box}\,\phi \mid E\,\pi \mid A\,\pi$$

$$\pi ::= \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi' \mid \phi\ \{\varphi\}\ U^{\Box}\ \{\varphi'\}\ \phi'$$

Thus defines state formulae ϕ, path formulae π and action formulae φ (boolean compositions of actions) over set of atomic actions $\{a, b, \ldots\}$

$\langle a \rangle^{\Box}$ and $[a]^{\Box}$ represent the classic deontic modalities O and P, resp.

vACTL: semantics with MTS as interpretation structure

- $q \models \mathit{true}$ always holds
- $q \models \neg\phi$ iff not $q \models \phi$
- $q \models \phi \wedge \phi'$ iff $q \models \phi$ and $q \models \phi'$
- $q \models \langle a \rangle \phi$ iff $\exists\, q' \in Q$ such that $q \xrightarrow{a}_{\Diamond} q'$, and $q' \models \phi$
- $q \models [a] \phi$ iff $\forall\, q' \in Q$ such that $q \xrightarrow{a}_{\Diamond} q'$, we have $q' \models \phi$
- $q \models \langle a \rangle^{\Box} \phi$ iff $\exists\, q' \in Q$ such that $q \xrightarrow{a}_{\Box} q'$, and $q' \models \phi$
- $q \models [a]^{\Box} \phi$ iff $\forall\, q' \in Q$ such that $q \xrightarrow{a}_{\Box} q'$, we have $q' \models \phi$
- $q \models E\,\pi$ iff $\exists\, \sigma' \in \mathit{path}(q)$ such that $\sigma' \models \pi$
- $q \models A\,\pi$ iff $\forall\, \sigma' \in \mathit{path}(q)$ such that $\sigma' \models \pi$
- $\sigma \models \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi'$ iff $\exists j \geq 1 : \sigma(j) \models \phi'$, $\sigma\{j\} \models \varphi'$, and $\sigma(j+1) \models \phi'$, and $\forall 1 \leq i < j : \sigma(i) \models \phi$ and $\sigma\{i\} \models \varphi$
- $\sigma \models \phi\ \{\varphi\}\ U^{\Box}\ \{\varphi'\}\ \phi'$ iff σ is a must path $\sigma^{\Box}$ and $\sigma^{\Box} \models \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi'$

Abbreviations: $EF\,\phi = E(\mathit{true}\ \{\mathit{true}\}\ U\ \{\mathit{true}\}\ \phi)$; $EF^{\Box}\,\phi = E(\mathit{true}\ \{\mathit{true}\}\ U^{\Box}\ \{\mathit{true}\}\ \phi)$; $EF\ \{\varphi\}\ \mathit{true} = E(\mathit{true}\ \{\mathit{true}\}\ U\ \{\varphi\}\ \mathit{true})$; $EF^{\Box}\ \{\varphi\}\ \mathit{true} = E(\mathit{true}\ \{\mathit{true}\}\ U^{\Box}\ \{\varphi\}\ \mathit{true})$; $AG\,\phi = \neg EF\,\neg\phi$; $AG^{\Box}\,\phi = \neg EF^{\Box}\,\neg\phi$; etc.

vACTL: semantics with MTS as interpretation structure

- $q \models \mathit{true}$ always holds
- $q \models \neg\phi$ iff not $q \models \phi$
- $q \models \phi \wedge \phi'$ iff $q \models \phi$ and $q \models \phi'$
- $q \models \langle a \rangle \phi$ iff $\exists\, q' \in Q$ such that $q \xrightarrow{a}_{\Diamond} q'$, and $q' \models \phi$
- $q \models [a] \phi$ iff $\forall\, q' \in Q$ such that $q \xrightarrow{a}_{\Diamond} q'$, we have $q' \models \phi$
- $q \models \langle a \rangle^{\Box} \phi$ iff $\exists\, q' \in Q$ such that $q \xrightarrow{a}_{\Box} q'$, and $q' \models \phi$
- $q \models [a]^{\Box} \phi$ iff $\forall\, q' \in Q$ such that $q \xrightarrow{a}_{\Box} q'$, we have $q' \models \phi$
- $q \models E\,\pi$ iff $\exists\, \sigma' \in \mathit{path}(q)$ such that $\sigma' \models \pi$
- $q \models A\,\pi$ iff $\forall\, \sigma' \in \mathit{path}(q)$ such that $\sigma' \models \pi$
- $\sigma \models \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi'$ iff $\exists j \geq 1 : \sigma(j) \models \phi'$, $\sigma\{j\} \models \varphi'$, and $\sigma(j+1) \models \phi'$, and $\forall 1 \leq i < j : \sigma(i) \models \phi$ and $\sigma\{i\} \models \varphi$
- $\sigma \models \phi\ \{\varphi\}\ U^{\Box}\ \{\varphi'\}\ \phi'$ iff σ is a must path $\sigma^{\Box}$ and $\sigma^{\Box} \models \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi'$

Abbreviations: $EF\phi = E(\mathit{true}\ \{\mathit{true}\}\ U\ \{\mathit{true}\}\ \phi)$; $EF^{\Box}\phi = E(\mathit{true}\ \{\mathit{true}\}\ U^{\Box}\ \{\mathit{true}\}\ \phi)$; $EF\ \{\varphi\}\ \mathit{true} = E(\mathit{true}\ \{\mathit{true}\}\ U\ \{\varphi\}\ \mathit{true})$; $EF^{\Box}\ \{\varphi\}\ \mathit{true} = E(\mathit{true}\ \{\mathit{true}\}\ U^{\Box}\ \{\varphi\}\ \mathit{true})$; $AG\,\phi = \neg EF\,\neg\phi$; $AG^{\Box}\ \phi = \neg EF^{\Box}\ \neg\phi$; etc.

Advanced variability management

vACTL can complement behavioural description of MTS by expressing constraints over possible products of a family that MTSs cannot model

Template ALT: Features F1 and F2 are *alternative*

$$(EF^{\Box}\,\{F1\}\,true \vee EF^{\Box}\,\{F2\}\,true) \wedge \neg(EF\,\{F1\}\,true \wedge EF\,\{F2\}\,true)$$

Template EXC: Feature F1 *excludes* feature F2

$$((EF\,\{F1\}\,true) \Longrightarrow (AG\,\neg\langle F2\rangle\,true)) \wedge ((EF\,\{F2\}\,true) \Longrightarrow (AG\,\neg\langle F1\rangle\,true))$$

Template REQ: Feature F1 *requires* feature F2

$$(EF\,\{F1\}\,true) \Longrightarrow (EF^{\Box}\,\{F2\}\,true)$$

Note: no temporal ordering among the related features

This is duty of the behavioural LTS/MTS description of product/family, which can then be verified by appropriate vACTL formulae

Advanced variability management

vACTL can complement behavioural description of MTS by expressing constraints over possible products of a family that MTSs cannot model

Template ALT: Features F1 and F2 are *alternative*

$$(EF^{\Box}\,\{F1\}\,true \vee EF^{\Box}\,\{F2\}\,true) \wedge \neg(EF\,\{F1\}\,true \wedge EF\,\{F2\}\,true)$$

Template EXC: Feature F1 *excludes* feature F2

$$((EF\,\{F1\}\,true) \Longrightarrow (AG\,\neg\langle F2\rangle\,true)) \wedge ((EF\,\{F2\}\,true) \Longrightarrow (AG\,\neg\langle F1\rangle\,true))$$

Template REQ: Feature F1 *requires* feature F2

$$(EF\,\{F1\}\,true) \implies (EF^{\Box}\,\{F2\}\,true)$$

Note: no temporal ordering among the related features

This is duty of the behavioural LTS/MTS description of product/family, which can then be verified by appropriate vACTL formulae

Advanced variability management

VACTL can complement behavioural description of MTS by expressing constraints over possible products of a family that MTSs cannot model

Template ALT: Features F1 and F2 are *alternative*

$$(EF^{\Box}\,\{F1\}\,true \vee EF^{\Box}\,\{F2\}\,true) \wedge \neg(EF\,\{F1\}\,true \wedge EF\,\{F2\}\,true)$$

Template EXC: Feature F1 *excludes* feature F2

$$((EF\,\{F1\}\,true) \Longrightarrow (AG\,\neg\langle F2\rangle\,true)) \wedge ((EF\,\{F2\}\,true) \Longrightarrow (AG\,\neg\langle F1\rangle\,true))$$

Template REQ: Feature F1 *requires* feature F2

$$(EF\,\{F1\}\,true) \implies (EF^{\Box}\,\{F2\}\,true)$$

Note: no temporal ordering among the related features

This is duty of the behavioural LTS/MTS description of product/family, which can then be verified by appropriate VACTL formulae

Advanced variability management

vACTL can complement behavioural description of MTS by expressing constraints over possible products of a family that MTSs cannot model

Template ALT: Features F1 and F2 are *alternative*

$$(EF^{\Box}\,\{F1\}\,true \vee EF^{\Box}\,\{F2\}\,true) \wedge \neg(EF\,\{F1\}\,true \wedge EF\,\{F2\}\,true)$$

Template EXC: Feature F1 *excludes* feature F2

$$((EF\,\{F1\}\,true) \Longrightarrow (AG\,\neg\langle F2\rangle\,true)) \wedge ((EF\,\{F2\}\,true) \Longrightarrow (AG\,\neg\langle F1\rangle\,true))$$

Template REQ: Feature F1 *requires* feature F2

$$(EF\,\{F1\}\,true) \implies (EF^{\Box}\,\{F2\}\,true)$$

Note: no temporal ordering among the related features

This is duty of the behavioural LTS/MTS description of product/family, which can then be verified by appropriate vACTL formulae

Advanced variability management

vACTL can complement behavioural description of MTS by expressing constraints over possible products of a family that MTSs cannot model

Template ALT: Features F1 and F2 are *alternative*

$$(EF^{\Box}\,\{F1\}\,true \vee EF^{\Box}\,\{F2\}\,true) \wedge \neg(EF\,\{F1\}\,true \wedge EF\,\{F2\}\,true)$$

Template EXC: Feature F1 *excludes* feature F2

$$((EF\,\{F1\}\,true) \Longrightarrow (AG\,\neg\langle F2\rangle\,true)) \wedge ((EF\,\{F2\}\,true) \Longrightarrow (AG\,\neg\langle F1\rangle\,true))$$

Template REQ: Feature F1 *requires* feature F2

$$(EF\,\{F1\}\,true) \implies (EF^{\Box}\,\{F2\}\,true)$$

Note: no temporal ordering among the related features

This is duty of the behavioural LTS/MTS description of product/family, which can then be verified by appropriate vACTL formulae

Model checking families and products

VMC

An on-the-fly model-checker for VACTL defined as particularization of the FMC model checker for ACTL over a CCS-like input language

Recently implemented also an algorithm for product derivation

Assumptions constraints and behaviour of Travel agency family

1. Only service orchestration is modelled, ignoring data exchange
2. Services are invoked by request-response interaction interface: requests are modelled by actions suffixed with ‘serv’, responses by correlated actions ‘result’, ‘paymentOK’, and ‘denied’ actions
3. All remaining actions model interaction with the client
4. The alternative of contacting multiple reservation services is modelled by a sequential invocation of three services

Model checking families and products

VMC

An on-the-fly model-checker for VACTL defined as particularization of the FMC model checker for ACTL over a CCS-like input language

Recently implemented also an algorithm for product derivation

Assumptions constraints and behaviour of Travel agency family

1. Only service orchestration is modelled, ignoring data exchange
2. Services are invoked by request-response interaction interface: requests are modelled by actions suffixed with ‘serv’, responses by correlated actions ‘result’, ‘paymentOK’, and ‘denied’ actions
3. All remaining actions model interaction with the client
4. The alternative of contacting multiple reservation services is modelled by a sequential invocation of three services

MTS specification of Travel agency family

```
TravAgFam = must(login).Menu
Menu = must(trainreserve).TrainRes + must(flightreserve).FlightRes
     + must(hotelreserve).HotelRes + may(tourreserve).TourRes

TrainRes = must(datainput).must(timetableserv).must(result).
           must(showquotes).(must(choose).Pay + must(reject).Menu)

FlightRes = must(datainput).
          ( may(flightAserv).must(result).must(flightBserv).must(result).
            must(flightCserv).must(result).must(showbestquotes).
            (must(choose).Pay + must(reject).Menu)
          + may(defaultflightserv).must(result).must(showquotes).
            (must(choose).Pay + must(reject).Menu) )

HotelRes = must(datainput).
         ( may(hotelAserv).must(result).must(hotelBserv).must(result).
           must(hotelCserv).must(result).must(showbestquotes).
           (must(choose).Pay + must(reject).Menu)
         + may(defaulthotelserv).must(result).must(showquotes).
           (must(choose).Pay + must(reject).Menu) )

TourRes = must(possibletourserv).must(result).must(showquotes).
          (must(choose).HotelRes + must(reject).Menu)
Pay = must(payserv).(must(paymentOK).TravAgFam + must(denied).Menu)
```

Model checking Travel agency family

Property "A travel agency service always provides a selection of best quotes"

$$AF\ \langle must(showbestquotes)\rangle\ true$$

False, as contacting multiple services for best quotes is an alternative which is moreover only available for flight and hotel reservations

Since train reservations is a service that is mandatory, the formula is false for any product of the family

Property "A travel agency service must always provide the possibility to reject a proposal (of quotes)"

$$AG\ [must(result)]\ AF^{\Box}\ \{must(reject)\}\ true$$

True for the family and — since universal formulae of this form dictate the existence of a must path with certain characteristics, which by definition are found in all products — thus for all products

Model checking Travel agency family

Property "A travel agency service always provides a selection of best quotes"

$$AF\ \langle must(showbestquotes)\rangle\ true$$

False, as contacting multiple services for best quotes is an alternative which is moreover only available for flight and hotel reservations

Since train reservations is a service that is mandatory, the formula is false for any product of the family

Property "A travel agency service must always provide the possibility to reject a proposal (of quotes)"

$$AG\ [must(result)]\ AF^{\Box}\ \{must(reject)\}\ true$$

True for the family and — since universal formulae of this form dictate the existence of a must path with certain characteristics, which by definition are found in all products — thus for all products

Result of “A travel agency service always provides a selection of best quotes”

FMC

http://fmtlab.isti.cnr.it/fmc/V5.0/fmc.html

Google

FMC v5.0b

Commands Menu

- New Model ...
- Edit Current Model
- Explore the Model
- Show Abstract LTS
- Show Abstract Traces
- Welcome
- Quit

The formula:

AF <must(showbestquotes)>
true

is FOUND_FALSE in State C1

This happens because

C1 --> C2{must(login)} /* ... */
C2 --> C3{must(trainreserve)} /* ... */
C3 --> C7{must(datainput)} /* ... */
C7 --> C8{must(timetableserv)} /* ... */
C8 --> C9{must(result)} /* ... */
C9 --> C10{must(showquotes)} /* ... */
C10 --> C11{must(choose)} /* ... */
C11 --> C12{must(payserv)} /* ... */
C12 --> C1{must(paymentOK)} /* ... */

Logical Formula

```
AF <must(showbestquotes)> true
```

Check The Formula | Explain the Result

Experiment with products

FMC

http://fmtlab.isti.cnr.it/fmc/V5.0/fmc.html

Google

FMC v5.0b

Commands Menu

New Model ...

Load Current Model

Welcome

Quit

```
TravAgFam = must(login).Menu

Menu = must(trainreserve).TrainRes + must(flightreserve).FlightRes
     + must(hotelreserve).HotelRes + must(tourreserve).TourRes

TrainRes = must(datainput).must(timetableserv).must(result).
           must(showquotes).(must(choose).Pay + must(reject).Menu)

FlightRes = must(datainput).
            must(defaultflightserv).must(result).must(showquotes).
            (must(choose).Pay + must(reject).Menu)

HotelRes = must(datainput).
           must(defaulthotelserv).must(result).must(showquotes).
           (must(choose).Pay + must(reject).Menu)

TourRes = must(possibletourserv).must(result).must(showquotes).
          (must(choose).HotelRes + must(reject).Menu)

Pay = must(payserv).(must(paymentOK).TravAgFam + must(denied).Menu)

net SYS = TravAgFam
```

Logical Formula

```
EF <must(showbestquotes)> true
```

Check The Formula

Explain the Result

Variability constraint: contacting multiple reservation services and contacting a single reservation service are alternative features

Instantiation of Template ALT

$$(EF^{\Box}\ \{may(hotelAserv)\}\ true \vee EF^{\Box}\ \{may(defaulthotelserv)\}\ true) \wedge \neg(EF\ \{may(hotelAserv)\}\ true \wedge EF\ \{may(defaulthotelserv)\}\ true)$$

False for the family, but true for the product shown in the screenshot, which might hence be a valid product of the family

VMC can be used to experiment with products, using VACTL to guide the derivation of valid products satisfying all variability constraints

VMC can be used to automatically derive, from an MTS description of a product family **and** an associated set of VACTL formulae expressing further constraints for this family, **all** its valid products (a set of LTS descriptions of products, each one correct w.r.t. all VACTL constraints)

Variability constraint: contacting multiple reservation services and contacting a single reservation service are alternative features

Instantiation of Template ALT

$$(EF^{\Box}\ \{may(hotelAserv)\}\ true \vee EF^{\Box}\ \{may(defaulthotelserv)\}\ true) \wedge \neg(EF\ \{may(hotelAserv)\}\ true \wedge EF\ \{may(defaulthotelserv)\}\ true)$$

False for the family, but true for the product shown in the screenshot, which might hence be a valid product of the family

VMC can be used to experiment with products, using VACTL to guide the derivation of valid products satisfying all variability constraints

VMC can be used to automatically derive, from an MTS description of a product family **and** an associated set of VACTL formulae expressing further constraints for this family, **all** its valid products (a set of LTS descriptions of products, each one correct w.r.t. all VACTL constraints)

Variability constraint: contacting multiple reservation services and contacting a single reservation service are alternative features

Instantiation of Template ALT

$$(EF^{\Box}\ \{may(hotelAserv)\}\ true \vee EF^{\Box}\ \{may(defaulthotelserv)\}\ true) \wedge \neg(EF\ \{may(hotelAserv)\}\ true \wedge EF\ \{may(defaulthotelserv)\}\ true)$$

False for the family, but true for the product shown in the screenshot, which might hence be a valid product of the family

VMC can be used to experiment with products, using VACTL to guide the derivation of valid products satisfying all variability constraints

VMC can be used to automatically derive, from an MTS description of a product family **and** an associated set of VACTL formulae expressing further constraints for this family, **all** its valid products (a set of LTS descriptions of products, each one correct w.r.t. all VACTL constraints)

Conclusions and future work

We addressed model checking families of services, starting from the addition of variability to a simple action-based branching-time temporal logic interpreted over a basic form of variable transition systems

Services are traditionally modelled with richer transition systems, which need to be addressed in future work

In particular, FMC/VMC is closely related to the CMC model checker for SocL (a service-oriented logic) formulae over the process algebra COWS (Calculus for Orchestration of Web Services)

Adding variability management to this tool will allow handling families of services that use more complex mechanisms typical of SOC, like asynchronous communication, compensation and correlation

Conclusions and future work

We addressed model checking families of services, starting from the addition of variability to a simple action-based branching-time temporal logic interpreted over a basic form of variable transition systems

Services are traditionally modelled with richer transition systems, which need to be addressed in future work

In particular, FMC/VMC is closely related to the CMC model checker for SocL (a service-oriented logic) formulae over the process algebra COWS (Calculus for Orchestration of Web Services)

Adding variability management to this tool will allow handling families of services that use more complex mechanisms typical of SOC, like asynchronous communication, compensation and correlation

Conclusions and future work

We addressed model checking families of services, starting from the addition of variability to a simple action-based branching-time temporal logic interpreted over a basic form of variable transition systems

Services are traditionally modelled with richer transition systems, which need to be addressed in future work

In particular, FMC/VMC is closely related to the CMC model checker for SocL (a service-oriented logic) formulae over the process algebra COWS (Calculus for Orchestration of Web Services)

Adding variability management to this tool will allow handling families of services that use more complex mechanisms typical of SOC, like asynchronous communication, compensation and correlation

Publicity

SOAP track on Service-Oriented Architectures and Programming
@ 27th Annual ACM Symposium on Applied Computing (SAC'12)

⇒ `http://www.itu.dk/acmsac2012-soap/`

IMPORTANT DATES (strict)

August 31, 2011: Paper submission
October 12, 2011: Author notifications
November 2, 2011: Camera-ready copy
March 25-29, 2012: SOAP @ SAC 2012