Refinement trees: Calculi, Tools and Applications

Mihai Codescu and Till Mossakowski

DFKI GmbH Bremen

31.08.2011, CALCO 2011

Sichere Kognitive Systeme

Stepwise Refinement

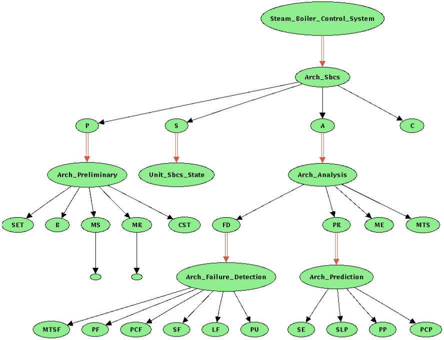
Start with a requirement specification SP_0 which only fixes the expected properties of the software system.

At each refinement step, add more details of the design, until the specification reached can be easily implemented by a program.

$$
SP_0 \rightsquigarrow \begin{cases} SP_1 \rightsquigarrow P_1 \\ \vdots \\ SP_n \rightsquigarrow \begin{cases} SP_{n1} \rightsquigarrow \{\; SP_{n11} \rightsquigarrow P_{n11} \\ \cdots \\ SP_{nm} \rightsquigarrow P_{nm} \end{cases} \end{cases}
$$

Objectives

- explicit representation of CASL refinement language as refinement trees
- prove that refinements are correct
- prove that refinements are consistent
- applications: consistency of large theories - DOLCE
- implement all of the above in Hets



Related Work

This work extends:

- T. Mossakowski, D. Sannella, A. Tarlecki - "A Simple Refinement Language for CASL", WADT 2004 [CASL-Ref]
- P. Hoffman - "Architectural Specification Calculus", Chapter IV.5 of CASL Reference Manual [CASL-RM]

CASL, the Common Algebraic Specification Language

specification libraries

architectural refinements

structured specifications

subsorted first-order logic
+ partiality + induction

CASL, the Common Algebraic Specification Language

specification libraries

architectural refinements

structured specifications

put your favorite
logical system here!

Institutions

Sichere Kognitive Systeme

Institutions formalize logical systems (Goguen/Burstall 1984)

An institution consists of:

- a category **Sign** of signatures;
- a functor **Sen**: **Sign** $\to$ **Set**, giving a set **Sen**(Σ) of Σ-sentences for each signature $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Sen}(\sigma)(\varphi)$ is written $\sigma(\varphi)$;
- a functor **Mod**: $\mathbf{Sign}^{op} \to \mathbf{Cat}$, giving a category **Mod**(Σ) of Σ-models for each $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Mod}(\sigma)(M')$ is written $M'|_\sigma$;
- for each $\Sigma \in |\mathbf{Sign}|$, a satisfaction relation $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ such that for any $\sigma\colon \Sigma \to \Sigma'$, $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_{\Sigma'} \sigma(\varphi) \iff M'|_\sigma \models_\Sigma \varphi \qquad [\textit{Satisfaction condition}]$$

Specification frames

Specification frames formalize the notion of logical theory (Ehrig/Pepper/Orejas 1989).
A specification frame is an indexed category $\mathbf{Mod}\colon \mathbf{Th}^{op} \to \mathbf{Cat}$.

$$\begin{array}{ccc} T_1 & \qquad & \mathbf{Mod}(T_1) \\ \big\downarrow \sigma & & \big\uparrow \mathbf{Mod}(\sigma) = \_|_\sigma \\ T_2 & & \mathbf{Mod}(T_2) \end{array}$$

We assume that **Th** is (finitely) cocomplete.
Moreover, we assume that **Th** comes with an inclusion system ($\Rightarrow$ unions).
Sometimes we also need that **Mod** takes colimits to limits (amalgamation property).

In this work, we work over an arbitrary specification frame. Examples use first-order logic, with CASL notation.

Structured specifications

$$SP ::= Th \,|\, SP_1 \textbf{ and } SP_2 \,|\, SP \textbf{ with } \sigma \,|\, SP \textbf{ hide } \sigma$$

$\mathbf{Mod}(Th)$ is given above
$\mathbf{Mod}(SP_1 \textbf{ and } SP_2) = \mathbf{Mod}(\iota_1)^{-1}(\mathbf{Mod}(SP_1)) \cap \mathbf{Mod}(\iota_2)^{-1}(\mathbf{Mod}(SP_2))$
$\mathbf{Mod}(SP \textbf{ with } \sigma) = \mathbf{Mod}(\sigma)^{-1}(\mathbf{Mod}(SP))$
$\mathbf{Mod}(SP \textbf{ hide } \sigma) = \mathbf{Mod}(\sigma)(\mathbf{Mod}(SP))$

Architectural specifications

Branching points are represented in CASL as architectural specifications.

arch spec ADDITION_FIRST =
units
N : NAT;
F : NAT $\rightarrow$ {**op** $suc(n : Nat) : Nat = n + 1$};
result F [N]

An architectural specification is correct if the models of its units can be combined as prescribed by the result unit expression.

CASL Architectural Specifications

$$
\begin{aligned}
&ASP ::= S \mid \textbf{units}\ UDD_1 \ldots UDD_n\ \textbf{result}\ UE\\
&UDD ::= UDEFN \mid UDECL\\
&UDECL ::= UN : USP\ \ < \textbf{given}\ UT_1, \ldots, UT_n >\\
&USP ::= SP \mid SP_1 \times \cdots \times SP_n \to SP \mid ASP\\
&UDEFN ::= UN = UE\\
&UE ::= UT \mid \lambda\ A_1 : SP_1, \ldots,\ A_n : SP_n \bullet UT\\
&UT ::= UN \mid F\ [FIT_1] \ldots [FIT_n] \mid UT\ \textbf{and}\ UT \mid UT\ \textbf{with}\ \sigma : \Sigma \to \Sigma' \mid\\
&\qquad\quad UT\ \textbf{hide}\ \sigma : \Sigma \to \Sigma' \mid \textbf{local}\ UDEFN_1 \ldots UDEFN_n\ \textbf{within}\ UT\\
&FIT ::= UT \mid UT\ \textbf{fit}\ \sigma : \Sigma \to \Sigma'
\end{aligned}
$$

Deductive Calculus for Architectural Specs

Checks whether an architectural specification ASP has a denotation and that the units produced by ASP satisfy a given unit specification USP - denoted $\vdash ASP :: USP$.

- based on a diagram D_{UT} for unit terms UT (of dependencies between units), where nodes are labeled with sets of specifications.
- verification conditions are discharged in a quite complicated manner.
- in [CASL-RM], the architectural language is restricted.
- unit imports left out due to increased complexity.

Constructive Calculus for Architectural Specs

- extract the specification of each unit expression and uses it to compute the specification of the result unit - denoted $\vdash ASP ::_c USP$.

Specification of a unit term (first try)

Let ASP be an architectural specification and UT a unit term. Then the specification of UT, denoted $\mathscr{S}_{ASP}(UT)$ is defined as follows:

- if UT is a unit name, then $\mathscr{S}_{ASP}(UT) = SP$ where $UT : SP$ is the declaration of UT in ASP;
- if $\mathscr{S}_{ASP}(A_i) = SP_i$ then $\mathscr{S}_{ASP}(A_1 \textbf{ and } \dots \textbf{ and } A_n) = SP_1 \textbf{ and } \dots \textbf{ and } SP_n$;
- if $\mathscr{S}_{ASP}(A) = SP$, then $\mathscr{S}_{ASP}(A \textbf{ with } \sigma) = SP \textbf{ with } \sigma$;
- if $\mathscr{S}_{ASP}(A) = SP$, then $\mathscr{S}_{ASP}(A \textbf{ hide } \sigma) = SP \textbf{ hide } \sigma$;
- if $UT = F[UT_1 \textbf{ fit } \sigma_1] \dots [UT_n \textbf{ fit } \sigma_n]$, where $\mathscr{S}_{ASP}(F) = SP_1 \times \dots \times SP_n \to SP$ and for any $i = 1, \dots, n$, $\mathscr{S}_{ASP}(UT_i) \rightsquigarrow SP_i \textbf{ with } \sigma_i$, then $\mathscr{S}_{ASP}(UT) = \{SP \textbf{ with } \sigma\} \textbf{ and } \mathscr{S}_{ASP}(UT_1) \textbf{ with } \iota_1;\iota' \textbf{ and} \dots \textbf{and } \mathscr{S}_{ASP}(UT_n) \textbf{ with } \iota_n;\iota'$;

A problem

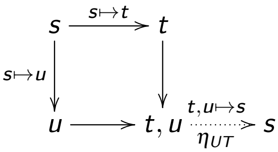
The specification of unit terms is sound, but too weak (i.e. incomplete).

arch spec ASP =

units U : **sort** s;

$\mathrm{UT} = (\mathrm{U}$ **with** $s \mapsto t)$ **and** $(\ \mathrm{U}$ **with** $s \mapsto u)$

result UT



Specification of a unit term (enhanced)

Sichere Kognitive Systeme

Let ASP be an architectural specification and UT a unit term. Then the specification of UT, denoted $\mathscr{S}_{ASP}(UT)$ is defined as follows:

- if UT is a unit name, then $\mathscr{S}_{ASP}(UT) = SP$ where $UT : SP$ is the declaration of UT in ASP;
- if $UT = F[UT_1 \textbf{ fit } \sigma_1]\dots[UT_n \textbf{ fit } \sigma_n]$, where $\mathscr{S}_{ASP}(F) = SP_1 \times \cdots \times SP_n \to SP$ and for any $i = 1, \dots, n$, $\mathscr{S}_{ASP}(UT_i) \models SP_i \textbf{ with } \sigma_i$, then $\mathscr{S}_{ASP}(UT) = \{SP \textbf{ with } \sigma\}$ **and** $\mathscr{S}_{ASP}(UT_1) \textbf{ with } \iota_1;\iota'$ **and** ... **and** $\mathscr{S}_{ASP}(UT_n) \textbf{ with } \iota_n;\iota'$ **and** $\mathbf{S}_{colim}(UT)$;
- if $UT = A_1$ **and** ... **and** A_n and $\mathscr{S}_{ASP}(A_i) = SP_i$ then $\mathscr{S}_{ASP}(UT) = SP_1$ **and** ... **and** SP_n **and** $\mathbf{S}_{colim}(UT)$;
- (rest remains)

where $\mathbf{S}_{colim}(UT) = Colim(D_{UT}) \textbf{ hide } \eta_{UT}$,
$\eta_{UT} : Sig(UT) \longrightarrow Colim(D_{UT})$ is the colimit injection of UT and D_{UT} is the diagram of UT.

Specification of a unit expression – Results

Theorem

If there are no imports and no generic unit is applied more than once, $\mathbf{Mod}(\mathscr{S}_{ASP}(UE)) = ProjRes(\mathbf{Mod}(ASP))$*, where UE is the result unit expression of ASP.*

Conjecture

With a generative semantics for architectural specifications, $\mathbf{Mod}(\mathscr{S}_{ASP}(UE)) = ProjRes(\mathbf{Mod}(ASP))$.

Constructive Calculus for Architectural Specs

$$\frac{\begin{array}{c}\Gamma_\emptyset \vdash UDD_1 ::_c \Gamma_1 \\ \vdots \\ \Gamma_{n-1} \vdash UDD_n ::_c \Gamma_n\end{array}}{\vdash \textbf{units } UDD_1 \ldots UDD_n \textbf{ result } UE ::_c \mathscr{S}_{\Gamma_n}(UE)}$$

$$\frac{\vdash UDECL ::_c \Gamma'}{\Gamma \vdash UDECL \textbf{ qua } UDD ::_c \Gamma \cup \Gamma'} \qquad \frac{\Gamma \vdash UDEFN ::_c \Gamma'}{\Gamma \vdash UDEFN \textbf{ qua } UDD ::_c \Gamma'}$$

$$\frac{\vdash SPR ::_c (USP, BSP)}{\vdash UN : SPR ::_c \{UN \mapsto USP\}} \qquad \frac{}{\Gamma \vdash UN = UE ::_c \Gamma \cup \{UN \mapsto \mathscr{S}_{\Gamma}(UE)\}}$$

Constructive Calculus for Arch Specs – Results

Assume: ASP has no unit imports, is syntactically correct, and each parametric unit is consistent and applied only once.

Theorem

$\vdash ASP ::_c USP$ *implies* $\vdash ASP :: USP$.

Theorem

If $\vdash ASP :: USP$ *for some* USP, *then* $\vdash ASP ::_c USP'$ *where* USP' *is the specification of the result unit of* ASP *and moreover* $USP' \rightsquigarrow USP$.

Corollary

$\vdash ASP ::_c USP$ *implies* $ProjRes(\mathbf{Mod}(ASP)) \subseteq \mathbf{Mod}(USP)$.

Corollary

If $ProjRes(\mathbf{Mod}(ASP)) \subseteq \mathbf{Mod}(USP)$ *then* $\vdash ASP ::_c USP'$ *and* $USP' \rightsquigarrow USP$.

Simple Refinement

The simplest form: model class inclusion. [CASL-Ref] introduces the following syntax:

refinement $\mathrm{R1} = \textsc{Monoid}$ **refined via** *Elem* $\mapsto$ *Nat* **to** $\textsc{Nat}$

Correctness of this refinement means that

$$M|_\sigma \in [\![\textsc{Monoid}]\!] \text{ for each } M \in [\![\textsc{Nat}]\!]$$

where σ maps *Elem* to *Nat*.

Composing Refinements

Refinements can be composed in chains of refinements:

refinement $\mathrm{R1} = \textsc{Monoid}$ **refined via** *Elem* $\mapsto$ *Nat* **to** $\textsc{Nat}$

refinement $\mathrm{R2} = \textsc{Nat}$ **refined via** *Nat* $\mapsto$ *Bin* **to** $\textsc{NatBin}$

refinement $\mathrm{R3} = \mathrm{R1}$ **then** $\mathrm{R2}$

Composition is defined only if the corresponding signatures match in the sense of [CASL-Ref].

Branching Refinement

Architectural specifications express branching points in refinements.

arch spec ADDITION_FIRST =
  units
    N : NAT;
    F : NAT → {**op** *suc*(*n* : *Nat*) : *Nat* = *n* + 1};
  result F [N]

refinement R4 =
    NATWITHSUC **refined to arch spec** ADDITION_FIRST

An architectural specification is correct if the models of its units can be combined as prescribed by the result unit expression.

Unit imports

Unit imports (written **given** N) are shorthand for parametric units that are applied once.

arch spec ADDITION_FIRST =
units
N : NAT;
M : NATWITHSUC **given** N;
result M

arch spec ADDITION_FIRST =
units
N : NAT;
means M : **arch spec** {**units**
F : NAT → NATWITHSUC ;
result F [N]}
result M

Component Refinement

arch spec ADDITION_FIRST =
  units
    N : NAT;
    M : NATWITHSUC **given** N;
  result M

Components of architectural specifications can be further refined:

refinement R =
    arch spec ADDITION_FIRST **then** {N **to** R2}

It is possible to refine more than one component at once (for example M could also be refined).

Refinement Trees

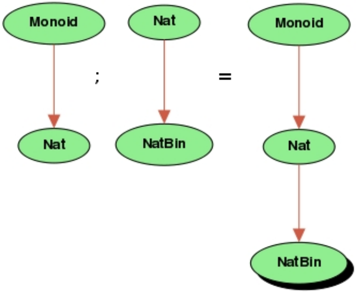
- nodes are labeled with unit specifications
- two types of links: refinement links and component links
- ”grow” both at the root and at the leaves
- come with an auxiliary structure for managing compositions

Composition of Refinement Trees

refinement R1 =
MONOID **refined via** *Elem* $\mapsto$ *Nat* **to** NAT

refinement R2 =
NAT **refined via** *Nat* $\mapsto$ *Bin* **to** NATBIN

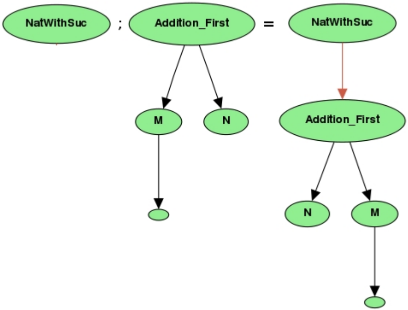
refinement R3 =
R1 **then** R2



Composition of Refinement Trees

arch spec ADDITION_FIRST =
units
N : NAT;
M : NATWITHSUC **given** N;
result M

refinement R4 =
NATWITHSUC **then**
arch spec ADDITION_FIRST

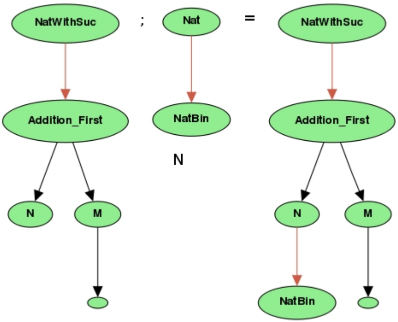


Composition of Refinement Trees

arch spec $\text{ADDITION\_FIRST} =$
units
$\text{N} : \text{NAT};$
$\text{M} : \text{NATWITHSUC}$ **given** $\text{N};$
result M

refinement $\text{R2} =$
NAT **refined via** *Nat* $\mapsto$ *Bin* **to** NATBIN

refinement $\text{R} =$
arch spec ADDITION_FIRST **then**
$\{\text{N}$ **to** $\text{R2}\}$



Proof Calculus for Refinement Language

Sichere Kognitive Systeme

Checks whether a refinement specification has a denotation and also constructs its refinement tree.

- based on $\vdash ASP ::_c USP$ for architectural specifications.
- extends to the refinement language in a natural way - specifications of units are now arbitrary refinements.
- unit imports can be replaced by an equivalent construction using the specification of the imported unit and raise no increase in complexity.

Proof Calculus for Refinements

$$\frac{(n, \mathscr{RT}) = \mathscr{RT}_{\emptyset}[USP]}{\vdash USP ::_c (USP, USP), \mathscr{RT}, (n, n)}$$

$$\frac{\begin{array}{c}\vdash USP ::_c (USP, USP), \mathscr{RT}_1, p_1 \\ \vdash SPR ::_c (USP', BSP), \mathscr{RT}_2, p_2 \\ (\mathscr{RT}, p) = \mathscr{RT}_1 \circ_{p_1, p_2} \mathscr{RT}_2 \\ USP \rightsquigarrow_{\sigma} USP'\end{array}}{\vdash USP \textbf{ refined via } \sigma \textbf{ to } SPR ::_c (USP' \textbf{ hide } \sigma, BSP), \mathscr{RT}, p}$$

Proof Calculus for Refinements

Sichere Kognitive Systeme

$$\frac{\begin{array}{c}\vdash ASP ::_c USP \\ \vdash SPR_i ::_c (USP_i, BSP_i), \mathscr{RT}_i, p_i \\ \text{for any } UN_i : SPR_i \text{ in } ASP \\ SPM(UN_i) = BSP_i \\ (n, \mathscr{RT}') = \mathscr{RT}_\emptyset[USP] \\ \mathscr{RT} = \mathscr{RT}'[n \to \mathscr{RT}_1, \ldots, \mathscr{RT}_k] \\ p = (n, \{UN_i \mapsto p_i\}_{i=1,\ldots,k})\end{array}}{\vdash ASP ::_c (USP, SPM), \mathscr{RT}, p}$$

Proof Calculus for Refinements

$$\frac{\begin{array}{c}\vdash SPR_i ::_c S_i, \mathscr{RT}_i, p_i \\ \mathscr{RT} = \cup \mathscr{RT}_i \\ p = \{UN_i \to p_i\}\end{array}}{\vdash \{UN_i \textbf{ to } SPR_i\}_{i \in \mathscr{I}} ::_c \{UN_i \ \to \ S_i\}_{i \in \mathscr{I}}, \mathscr{RT}, p}$$

$$\frac{\begin{array}{c}\vdash SPR_1 ::_c S_1, \mathscr{RT}_1, p_1 \\ \vdash SPR_2 ::_c S_2, \mathscr{RT}_2, p_2 \\ S = S_1; S_2 \\ (p, \mathscr{RT}) = \mathscr{RT}_1 \circ_{p_1, p_2} \mathscr{RT}_2\end{array}}{\vdash SPR_1 \textbf{ then } SPR_2 ::_c S, \mathscr{RT}, p}$$

Sichere Kognitive Systeme

Theorem (Soundness)

Let SPR be a refinement specification such that $\vdash SPR \rhd \square$ *and all generic units in the architectural specifications appearing in SPR are consistent. If* $\vdash SPR ::_c S$*, then there is* $\mathscr{R}$ *such that* $\vdash SPR \Rightarrow \mathscr{R}$ *and* $\mathscr{R} \models S$*.*

Consistency Calculus

$$\frac{\vdash cons(USP)}{\vdash cons(USP \;\; qua \;\; \mathtt{SPEC\text{-}REF})}$$

$$\frac{\vdash cons(SPR)}{\vdash cons(USP \;\mathbf{refined\ via}\; \sigma \;\mathbf{to}\; SPR)}$$

$$\frac{\vdash cons(SPR) \;\; for \;\; all \;\; UN : SPR \;\; in \;\; ASP}{\vdash cons(ASP)}$$

$$\frac{\vdash cons(SPR_i)}{\vdash cons(\{U_i \;\mathbf{to}\; SPR_i\}_{i \in \mathscr{J}})}$$

$$\frac{\begin{array}{c}\vdash cons(SPR_1) \\ \vdash cons(SPR_2)\end{array}}{\vdash cons(SPR_1 \;\mathbf{then}\; SPR_2)}$$

Consistency Calculus
Results

Sichere Kognitive Systeme

Theorem (Soundness)

If $\vdash SPR ::_c \square$, *the calculi for checking consistency of structured specifications and conservativity of extensions are sound and* $\vdash cons(SPR)$, *then* SPR *has a model.*

Theorem (Completeness)

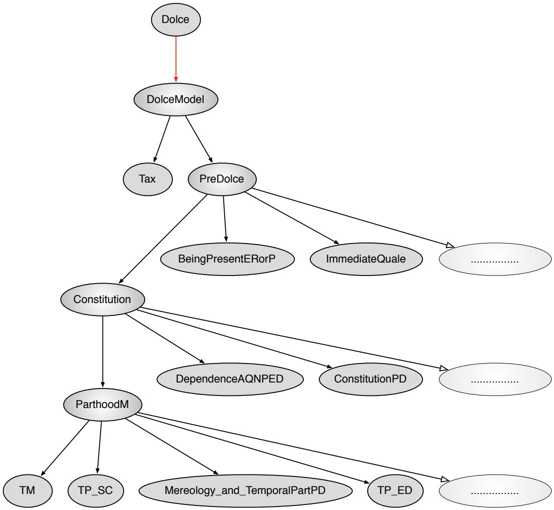
If unit imports are omitted, the calculi for checking consistency of structured specifications and conservativity of extensions are complete, $\vdash SPR ::_c \square$ *and* SPR *has a model, then* $\vdash cons(SPR)$.

Application: DOLCE consistency

DOLCE: Descriptive Ontology for Linguistic and Cognitive Engineering contains several hundreds of axioms $\Rightarrow$ model finders fail

- first attempt: architectural spec structure follows that of structured spec $\Rightarrow$ failed (due to DEPENDENCE)
- second attempt followed structure of taxonomy $\Rightarrow$ successful
- by using a strengthening of DEPENDENCE, we could rely on stronger assumptions for the interpretation of DEPENDENCE for various subconcepts when extending it to a superconcept.
- architectural spec has 38 units
  - well-formedness check using HETS not feasible
  - after split into four architectural specs, well-formedness check using HETS took 35h on i7
  - the split leads to a refinement tree with 4 branchings
- DOLCE models can now be built in a modular way

Dolce: Refinement tree for model construction



Conclusions

- logic-independent framework for refinements
- based on institutions resp. specification frames
- tool support through Heterogeneous Tool Set `www.dfki.de/sks/hets`
- specialized notion of refinements via institution comorphisms
- open question: completeness of refinement calculus