Reasoning with Behavioural Knowledge in Application Domain Models

Ernesto Compatangelo[1], Francesco M. Donini[2], and Giovanni Rumolo[3]

[1] Istituto di Informatica della Facoltà di Ingegneria, Università di Ancona
[2] Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"
[3] Dipartimento di Informatica ed Automazione, Università di Roma TRE

Abstract. This paper describes an analyst-oriented approach to conceptual knowledge representation and reasoning based on description logics. The approach is introduced to model and analyse the static part of behavioural concepts used in application domains. Behaviours are captured in a parametric way with respect to the description logic which corresponds to the structural modelling language. Structural concepts are classified according to the usual ISA hierarchy, while behaviours are organised into a hierarchy based on countervariance. Reasoning about the domain model is performed in terms of subsumption and consistency in the adopted description logic, and complexity results carry over. The proposed approach is used to formalise and reason on process schemes introduced in the engineering of computer-based and information systems, as well as in enterprise modelling.

1 Introduction

The adoption of knowledge representation and reasoning techniques in application domain modelling is not a new idea. At the beginning of the eighties, research work on requirements specification [2] first pointed out the importance of an explicit representation of knowledge captured before the beginning of systems development. Since then, several languages and systems have been proposed to capture and manage this kind of conceptual (domain) knowledge [12, 11, 19]. However, most of them focus on the representation of structural aspects only. Moreover, those modelling approaches which deal with the representation of behavioural aspects are quite always implicitly unbalanced towards procedural details. This is the case of both Structured Data Flow (SDF) [8] and Object Modelling Technique (OMT) [13]. The observed unbalancement towards procedurality means that behavioural models often fail to support the "what vs. how" distinction, which is ubiquitous in conceptual modelling and analysis. The actual problem with most existing behavioural models is that domain knowledge should not contain any reference to the internal structure and behaviour of domain elements [11]. A domain description should give a coherent intensional image of what composes a domain with no reference to extensional individuals composing it. Data should be thus described as intensional classes, while processes should be described as black boxes with no transfer function.

A new kind of approach should be thus adopted to support domain modelling and analysis in different application areas. This approach should be explicitly conceived to formalise domain models, providing at the same time automated reasoning support which depends on the considered model as well as on the purposes of domain analysis. Formalisation is needed to avoid an ambiguous interpretation of domain concepts. Automated support is needed to deal with the high number of concepts in real-world application domain models, where manual analysis is unpractical and ineffective.

This paper describes a modelling approach based on Description Logics (DLs), which are a family of representation languages designed to model rich hierarchies of classes (concepts) [21, 15]. A DL is a subset of first-order logic with equality that contains only unary and binary predicates. It is endowed with a set of *constructors* that determines the language in which class properties can be specified. Much of the research in DLs concentrated on algorithms for reasoning about concepts. The computational and expressive properties of DLs have been extensively studied [9, 15, 1, 5].

While other approaches based on DLs [18, 5] deal with the structural component of domain knowledge only, we use DLs also to formalise some relevant aspects of the static part of behavioural domain modelling (i.e. process descriptions). Although the DL used in this paper is $\mathcal{ALN}$ [9], behaviours are captured in a parametric way with respect to the adopted DL. In fact, the only requirement we impose on a DL to be used in structural modelling is that it contains the conjunction of concepts. This means that the structural part could have been modelled using a very simple DL, such as $\mathcal{FL}^-$, as well as a much more expressive DL such as $\mathcal{CATS}$ [10]. In our approach, structural concepts are classified according to the usual ISA hierarchy, while behaviours are organised into a hierarchy based on countervariance. In fact, we propose a distinct hierarchy-forming rule for behavioural concepts instead of the well-known subsumption rule used for structural concepts. We use a DL to formalise the flow assertion between two behavioural concepts, which is a relevant domain-dependent constraint. Reasoning about the domain model is performed in terms of subsumption and consistency in the adopted DL. Moreover, we show that we only need a linear number of consistency concept checking with respect to the size of the domain model. Therefore, results about the complexity of reasoning in the adopted DL imply the same results about the complexity of reasoning in the domain model.

The presentation is organised as follows. Section two introduces a real-world domain fragment in order to point out some main features of our approach. Section three shows the set-theoretic semantics of the modelling language as well as the properties of the related deduction problems. Differences between our consistency checking and countervariant functional typing are also shown. Finally, section four summarises our approach and outlines how it captures some relevant properties of conceptual models in different application domains.

2 Representing Domain Behaviours: an Example

In order to point out the main features of our approach, we present a modelling example derived from the healthcare enterprise domain. The example is first introduced using a textual description and then modelled using the analyst-oriented language $\mathcal{EDDL}_{DP}$ (Epistemological Domain Description Language for Data and Processes) presented in [7]. Each $\mathcal{EDDL}_{DP}$ statement can be translated into a finite set of expressions belonging to the $\mathcal{ALN}$ description logic.

A set of Physiological Data about a patient (Blood Pressure, Temperature, Pulse Frequency) is periodically read from different devices, formatted and successively registered in a Chronologically Ordered List. Every time a data set is read, the value of each element of the set is compared with a fixed range of corresponding Normal Physiological Values. If at least one element is outside its normal range, an alarm condition is signalled. A Patient Monitoring Report containing a chronologically ordered subset of the last N data taken from the previously cited list is issued whenever requested by the Medical Staff.

The corresponding $\mathcal{EDDL}_{DP}$ description is given in Tables 1, 2, 3 and 4. In this section, an intuitive meaning is outlined for the $\mathcal{EDDL}_{DP}$ description, while a formal semantics for $\mathcal{EDDL}_{DP}$ statements is deferred to the next section. Here and in the following, terms in bold denote reserved words, terms in small capital with first letter in capital denote concept names while terms all in small capital denote attributes. If present, an L **:** U declaration, where L and U are both natural numbers, denotes the cardinality of attributes and channels. In both cases, U $\geq$ L and we use the special symbol "M" to denote an unbounded value for U (infinite). When considering channels, L $\geq$ 1 must hold. For instance, let us consider the $\mathcal{EDDL}_{DP}$ description of PHYS-DATA shown in Table 1. This data concept has the meaning of a set (class) of domain elements with an attribute PRES of type BLOOD-PRES, an attribute FREQ of type PULSE-FREQ and an attribute TEMP of type MEAN-TEMP. All these attributes are functional mandatory ones, i.e. upper and lower cardinality constraints have unit value.

Following a well-known semantical distinction introduced in description logics [21], each new data concept can be defined as being either **equals** to or included **in** its description structure composed of already-existing data concepts. For example, T-PHYS-DATA is defined as a structure with the same properties as FMT-PHYS-DATA plus another attribute SAMPLING-TIME. The definition entails that all the individuals belonging to T-PHYS-DATA also belong to FMT-PHYS-DATA. Moreover the **equals** operator asserts that for each new data it is sufficient to define it with the four attributes TEMP, PRES, FREQ and SAMPLING-TIME to infer that it is subset of T-PHYS-DATA. Instead, when we choose the **in** operator for each new data defined in terms of the same four properties, we cannot infer that it is a T-PHYS-DATA. In this case, the definition of attributes only states necessary but not sufficient condition. The T-DATABASE concept, which represents the set of all individuals that have one or more elements of class T-PHYS-DATA, is neither a subset nor a superset of T-PHYS-DATA.

In this DL framework the ISA relationship has the meaning of set containment, i.e. a concept *subsumes* another when the former is included in the lat-

Table 1. Data descriptions in a patient monitoring domain

```
data

% Atomic concepts

Control ;

Time ;

Phys-Value ;

Person-Name ;

Family-Name ;

Disease ;

Fmt-Value ;

% Primitive concepts

Report-Request in Control ;

Alarm-Condition in Control ;

Blood-Pres in Phys-Value ;

Pulse-Freq in Phys-Value ;

Mean-Temp in Phys-Value ;

Fmt-Blood-Pres in
(Fmt-Value and Blood-Pres) ;

Fmt-Pulse-Freq in
(Fmt-Value and Pulse-Freq) ;

Fmt-Mean-Temp in
(Fmt-Value and Mean-temp) ;

% Defined concepts

Phys-Data equals
structure where
pres is Phys-Value
temp is Phys-Value
freq is Phys-Value ;

Fmt-Data equals
structure where
pres is Fmt-Value
temp is Fmt-Value
freq is Fmt-Value ;

Fmt-Phys-Data equals
structure where
pres is Fmt-Blood-Pres
temp is Fmt-Mean-Temp
freq is Fmt-Pulse-Freq ;

T-Phys-Data equals
Fmt-Phys-Data
and structure where
sampling-date is Time ;

Person equals
structure where
name is Person-Name
surname is Family-Name
birthday is Time ;

Patient equals
Person and structure where
pathology is 1 : M Disease
treated-by is Physician-Data ;

Physician equals
Person
and structure where
specialist-in is Disease
patients is Patient-Data ;

T-Database equals
structure where
element is 1 : M T-Phys-Data ;

Patient-Report equals
structure where
patient is Patient-Data
history is T-Database ;

Norm-Phys-Ranges equals
structure where
n-blood-pres is Fmt-Blood-Pres
n-mean-temp is Fmt-Mean-Temp
n-pulse-freq is Fmt-Pulse-Freq ;
```

Table 2. Data constraints in a patient monitoring domain

data constraints

disjoint Disease, Phys-Value, Time, Control, String ;

disjoint Alarm-condition, Report-Request ;

disjoint Pulse-Freq, Mean-Temp, Blood-Pres ;

ter. For example, Fmt-Phys-Data can be automatically classified under both Phys-Data and Fmt-Data since it is *subsumed* by these two parent concepts.

The formalisation of processes using DLs is the innovative feature of our approach which was introduced to capture some static aspects of behaviours. Each process definition in the $\mathcal{EDDL}_{DP}$ language is given by the corresponding I/O context, i.e. I/O channels together with their domains (see Table 3). Although this resembles the declaration usually given in a structured data flow schema, the formal semantics of DLs makes automatic deduction possible.

Table 3. Process descriptions in a patient monitoring domain

processes

Collect **has**
input structure where
pres **is** Blood-Pres
freq **is** Pulse-freq
temp **is** Mean-Temp
output structure where
result **is** Phys-Data ;

Sampling **has**
input structure where
data **is** Fmt-Data
output structure where
sampled-data **is** T-Phys-Data ;

Comparison **has**
input structure where
current-values **is** Fmt-Phys-Data
ref-values **is** Norm-Phys-Ranges
output structure where
status **is** Alarm-Condition ;

Report-Generation **has**
input structure where
request **is** Report-Request
rep-data **is** $1:$ M T-Database
rep-patient **is** $1:$ M Patient
output structure where
report **is** $1:$ M Patient-Report ;

Update **has**
input structure where
most-recent-data **is** T-Phys-Data
old-data-store **is** T-Database
output structure where
up-data-store **is** T-Database ;

Table 4. Flow descriptions in a patient monitoring domain

process constraints

flow sample
from Collect . result
to Sampling . data ;

flow monitoring
from Sampling . sampled-data
to Comparison . current-values ;

flow insertion
from Sampling . fmt-data
to Update . most-recent-data ;

flow registration
from Update . up-data-store
to Update . old-data-store ;

In the case of the Sampling and Comparison processes, the flow assertion monitoring (see Table 4) states that the output resulting from the Sampling process has to be transferred to the Comparison process. The intuitive meaning of flow assertions is that the output of the source process is the same as the input of the target process, i.e. the sets denoted by the two concepts are equal. This interpretation gives rise to three possible situations involving the output of the source process, i.e. $Out(\mathrm{P}_1)$ and the input of the target process, i.e. $In(\mathrm{P}_2)$.

- If $Out(\mathrm{P}_1) \cap In(\mathrm{P}_2) = \emptyset$ an inconsistency can be automatically detected.
- If $Out(\mathrm{P}_1) \subseteq In(\mathrm{P}_2)$ then the flow assertion is consistent and we can infer new constraints in the domain model. For example, the flow monitoring is consistent because Fmt-Phys-Data subsumes T-Phys-Data. Moreover, we can infer that the process Comparison actually has the same restricted concept T-Phys-Data as its input instead of Fmt-Phys-Data.
- If $Out(\mathrm{P}_1) \not\subseteq In(\mathrm{P}_2)$ but $Out(\mathrm{P}_1) \cap In(\mathrm{P}_2) \neq \emptyset$ we can infer new constraints for both source and target processes. For example the flow sample imposes that both Out(Collect) = Phys-Data and In(Sampling) = Fmt-Data are constrained to (Phys-Data **and** Fmt-Data). This concept can be automatically classified, discovering that it coincides with Fmt-Phys-Data. When reported to the analyst, this derived property can change his/her understanding of the domain.

It is very important to note that our interpretation of the flow assertion differs from the composition of functions. More specifically, if a process P_1 were to be considered as a function $\mathrm{P}_1 : Range_1 \rightarrow Domain_1$ and a process P_2 were to be considered as a function $\mathrm{P}_2 : Range_2 \rightarrow Domain_2$, then functional composition would have required $Domain_1 \subseteq Range_2$. Conversely, in our approach we only require that $Domain_1 \cap Range_2 \neq \emptyset$, warning the analyst that the output of process P_1 has to be constrained to $Domain_1 \cap Range_2$ in order to guarantee flow coherence. We reconsider this aspect in the next section, after the definition of a formal semantics for $\mathcal{EDDL}_{DP}$.

3 Reasoning with Domain Behaviours

All expressions in our $\mathcal{EDDL}_{DP}$ language can be given a set-theoretic semantics as follows. An interpretation $\mathcal{I}$ is defined as a triple $(\varepsilon[\,\cdot\,], \Delta_C, \Delta_P)$, where Δ_C and Δ_P are two disjoint sets of elements and $\varepsilon[\,\cdot\,]$ is a mapping. $\varepsilon[\,\cdot\,]$ assigns to each atomic data concept C a subset $\varepsilon[\,\mathrm{C}\,]$ of Δ_C (i.e., classes are interpreted as sets), to each attribute ATT a subset $\varepsilon[\,\mathrm{ATT}\,]$ of $\Delta_C \times \Delta_C$ (i.e. an attribute is interpreted as a binary relation), to each process P an element of Δ_P and to each channel CH a subset of $\Delta_P \times \Delta_C$. From the interpretation of the above atomic names, the intepretation of more complex constructs can be defined as follows. Let C and D be two concepts, ATT be an attribute. Conjunction of structures is interpreted as set intersection, i.e. $\varepsilon[\,(\mathrm{C}\ \textbf{and}\ \mathrm{D})\,] = \varepsilon[\,\mathrm{C}\,] \cap \varepsilon[\,\mathrm{D}\,]$. We denote the cardinality of a set S as $\sharp S$. Moreover, given an element $x \in \Delta_C$ and an attribute ATT, we denote with $\varepsilon[\,\mathrm{ATT}\,](x)$ the set $\{y \in \Delta_C \mid (x, y) \in \varepsilon[\,\mathrm{ATT}\,]\}$.

The semantics of a structure is as follows.

$$\begin{aligned}&\varepsilon[\,\textbf{structure where}\ \mathrm{ATT}\ \textbf{is}\ \mathrm{L}:\mathrm{U}\ \mathrm{C}\,] = \\ &\quad = \{x \in \Delta_C \mid \mathrm{L} \leq \sharp\varepsilon[\,\mathrm{ATT}\,](x) \leq \mathrm{U} \text{ and } \forall y \in \varepsilon[\,\mathrm{ATT}\,](x) : y \in \varepsilon[\,\mathrm{C}\,]\}\end{aligned} \tag{1}$$

A concept C is *satisfiable* if there exists an interpretation $(\varepsilon[\,\cdot\,], \Delta_C, \Delta_P)$ such that $\varepsilon[\,\mathrm{C}\,] \neq \emptyset$. Moreover, a concept C *subsumes* a concept D if for every interpretation $(\varepsilon[\,\cdot\,], \Delta_C, \Delta_P)$ it holds $\varepsilon[\,\mathrm{C}\,] \subseteq \varepsilon[\,\mathrm{D}\,]$. Given a concept name C and a complex concept D, we say that an interpretation $\mathcal{I} = (\varepsilon[\,\cdot\,], \Delta_C, \Delta_P)$ satisfies the definition of a concept (C **equals** D **;**) if $\varepsilon[\,\mathrm{C}\,] = \varepsilon[\,\mathrm{D}\,]$. Similarly, $\mathcal{I}$ satisfies the introduction of a primitive concept (C **in** D **;**) if $\varepsilon[\,\mathrm{C}\,] \subseteq \varepsilon[\,\mathrm{D}\,]$. $\mathcal{I}$ satisfies a data constraint **disjoint** C_1**,** . . . **,** C_n**, ;** if for all $i, j \in \{1, \ldots, n\}$ and $i \neq j$, it holds $\varepsilon[\,\mathrm{C}_i\,] \cap \varepsilon[\,\mathrm{C}_j\,] = \emptyset$. The interpretation $\mathcal{I}$ satisfies the description of a process (P **has input** C **output** D **;**) if

$$\varepsilon[\,\mathrm{P}\,] \in \varepsilon[\,\mathrm{C}\,] \cap \varepsilon[\,\mathrm{D}\,] \tag{2}$$

where C and D are two structures interpreted in the same way as in (1), except that to each channel is assigned a subset of $\Delta_P \times \Delta_C$.

We turn now to the interpretation of flow assertions, which is a key feature in our approach. An interpretation $\mathcal{I}$ satisfies a flow assertion from a source process P_s through its output channel OUT_s to a target process P_t through its input channel IN_t, i.e. **flow** F **from** P_s **.** OUT_s **to** P_t **.** IN_t **;** if all the elements which are output of P_s through OUT_s are input of P_t through IN_t. In formulae, $\{x \in \Delta_C \mid (\varepsilon[\,\mathrm{P}_s\,], x) \in \varepsilon[\,\mathrm{OUT}_s\,]\} = \{y \in \Delta_C \mid (\varepsilon[\,\mathrm{P}_t\,], y) \in \varepsilon[\,\mathrm{IN}_t\,]\}$.

Finally, an interpretation satisfies a domain description if it satisfies all parts of the description, i.e., data descriptions and constraints, process and flow descriptions. We call such an interpretation a *model* of the description. We say that a description is *satisfiable* if it has a model.

Checking the satisfiability of a description, i.e. checking that it admits at least one model is a considerable support tool in domain analysis. We can give necessary and sufficient conditions for the satisfiability of a description as follows.

Theorem 1. *A domain description is satisfiable if and only if:*

1. *for each process description* P **has input** C **output** D **;** *concepts* C *and* D *are satisfiable;*
2. *for each flow assertion* **flow** F **from** P_s **.** OUT_s **to** P_t **.** IN_t **;**
   (a) *source process* P_s *contains in its output* **structure where** OUT_s **is** L_s **:** $\mathrm{U}_s\mathrm{C}_s$*;*
   (b) *target process* P_t *contains in its input* **structure where** IN_t **is** L_t **:** $\mathrm{U}_t\mathrm{D}_t$*;*
   (c) *cardinalities of source and target channels are compatible, i.e., the two integer intervals* $[\mathrm{L}_s, \mathrm{U}_s]$ *and* $[\mathrm{L}_t, \mathrm{U}_t]$ *have the non-empty intersection* $[\,max(\mathrm{L}_s, \mathrm{L}_t), min(\mathrm{U}_s, \mathrm{U}_t)\,]$*;*
   (d) *The concept* $(\mathrm{C}_s$ **and** $\mathrm{D}_t)$ *is satisfiable.*

□

Proof. (sketch)
Only-if part: if condition 1 is not met, then there exists a process description for which every interpretation assigns to either C or D the empty set. Hence, no interpretation can satisfy that process description according to Formula (2). A similar conclusion holds if one of the conditions in 2 is not met.

If part: The proof makes use of the following property (see [4]): given an interpretation $\mathcal{I}$ of the domain description, one can always build another interpretation $\mathcal{I}'$, defined as the disjoint union of many copies of $\mathcal{I}$ (i.e., the union of interpretations which are the same as $\mathcal{I}$ but renaming with new names all elements in Δ_C, Δ_P). In this way, if $\mathcal{I}$ assigns to a concept C a non-empty set, containing at least one element $x \in \Delta_C$, $\mathcal{I}'$ assigns to C a set containing many copies of x, i.e. new elements having exactly the same properties as x. Suppose that for a given flow condition 2d holds. Then there exists an interpretation $\mathcal{I}= (\varepsilon\,[\,\cdot\,],\, \Delta_C,\, \Delta_P)$ such that $\varepsilon\,[\,(\mathrm{C}_s$ **and** $\mathrm{D}_t)\,]$ is non-empty. If $\sharp\,\varepsilon\,[\,(\mathrm{C}_s$ **and** $\mathrm{D}_t)\,] < max(\mathrm{L}_s,\, \mathrm{L}_t)$, then a new interpretation $\mathcal{I}' = (\varepsilon'\,[\,\cdot\,],\, {\Delta_C}',\, {\Delta_P}')$ can be defined as disjoint unions of $\mathcal{I}$ such that now $\sharp\,\varepsilon'\,[\,(\mathrm{C}_s$ **and** $\mathrm{D}_t)\,] \geq max(\mathrm{L}_s, \mathrm{L}_t)$. From $\mathcal{I}'$, one can build another interpretation $\mathcal{I}''$, which interprets concepts in the same way as $\mathcal{I}'$, and assigns to channels OUT_s and IN_t a number of elements which is inside the interval $[\,max(\mathrm{L}_s, \mathrm{L}_t), min(\mathrm{U}_s, \mathrm{U}_t)\,]$. Now $\mathcal{I}''$ satisfies the flow assertion, and also the part of the descriptions of P_s and P_t regarding OUT_s and IN_t, respectively, appearing in conditions (2a) and (2b). This operation can be iterated for each flow assertion, yielding a model of the domain description. □

The above theorem shows that to check the satisfiability of a domain description, one mainly needs to perform a number of satisfiability checkings which is linear in the number of process and flow descriptions. Hence, the complexity of satisfiability checking of a domain description is linearly related to the complexity of concept satisfiability in the underlying DL chosen for the data descriptions and constraints. For the DL used in this paper, which is a syntactic variant of the description logic $\mathcal{ALN}$, satisfiability checking of a concept of size n is a problem solvable in $O(n \log n)$ [3, 20], hence the satisfiability of a domain description of size n can be checked in $O(n \log n)$. We remark that the above theorem does not hold for DLs admitting the ONE-OF constructor—which allows classes to be described as explicit enumeration of their elements—since for such DLs the property of building new interpretations by disjoint unions does not hold.

Automated reasoning about processes is performed computing a process hierarchy based on functional type replacement. Following a well-known approach, we say that a process P is a *subtype* of a process P′ when the input context of P subsumes the input context of P′, and the output context of P is subsumed by the output context of P′. The subtype relation is a well established notion for programming languages. It was developed to guarantee the safe replacement of a function inside a composition of functions, where by "safe" we mean that the function resulting from the composition is always defined. In our framework, this notion is slightly extended. In fact, we not only check a safe process composition but we also individuate possibly more restrictive constraints that have to be imposed to processes in order to obtain a safe composition.

Let us consider the case represented in Table 5. In our framework the flow P_1-INTO-P_2 is satisfiable if and only if the intersection of input domain of P_2 has a non-empty intersection with the output domain of P_1. Let us recall that $In(\mathrm{Q})$ and $Out(\mathrm{Q})$ are the input and the output of process Q respectively. Then, the flow is satisfied in the following situations:

1. $Out(\mathrm{P}_1) \subseteq In(\mathrm{P}_2)$, i.e. there exists an inclusion between the output domain of P_1 and the input domain of P_2
2. $Out(\mathrm{P}_1) \nsubseteq In(\mathrm{P}_2)$, and $Out(\mathrm{P}_1) \cap In(\mathrm{P}_2) \neq \emptyset$, i.e. there only exists an intersection between the output domain of P_1 and the input domain of P_2

Note that the former represents the usual condition for functions composition while the latter represents our extension and in both the situations we consider both $Out(\mathrm{P}_1)$ and $In(\mathrm{P}_2)$ satisfiable. Moreover, while the former situation implies the latter, it isn't true the contrary. The same considerations are applied in the case of the flow P_2-INTO-P_3. Therefore, the process P_2' can replace the process P_2 iff: (i) P_2 is a subtype of P_2', (ii) the input domain of P_2' still has a non-empty intersection with the output domain of P_1 and (iii) the input domain of P_3 still has a non-empty intersection with the output domain of P_2'.

When (a) $Out(\mathrm{P}_1) \subseteq In(\mathrm{P}_2)$ and (b) $Out(\mathrm{P}_2) \subseteq In(\mathrm{P}_3)$ are true before the process replacement, from the countervariance subtype relation we infer: (i) $In(\mathrm{P}_2) \subseteq In(\mathrm{P}_2')$ and (ii) $Out(\mathrm{P}_2') \subseteq Out(\mathrm{P}_2)$ and therefore we conclude that: (c) $Out(\mathrm{P}_1) \subseteq In(\mathrm{P}_2')$ and (d) $Out(\mathrm{P}_2') \subseteq In(\mathrm{P}_3)$. This situation represents the usual safeness for function composition.

More interesting considerations apply when just (a) $Out(\mathrm{P}_1) \cap In(\mathrm{P}_2) \neq \emptyset$ or (b) $Out(\mathrm{P}_2) \cap In(\mathrm{P}_3) \neq \emptyset$ are true before process replacement. Let us consider the case (b): the countervariant subtype relation is not sufficient to infer that (d) $Out(\mathrm{P}_2') \subseteq In(\mathrm{P}_3)$. Hence, the composition of P_2' and P_3 is not safe. However, if $Out(\mathrm{P}_2') \cap In(\mathrm{P}_3) \neq \emptyset$ the composition is possible, with the further restriction that the *actual* output of P_2' is in $Out(\mathrm{P}_2') \cap In(\mathrm{P}_3)$. This additional restriction on P_2' is reported to the analyst when he/she tries to substitute P_2 with P_2'. In this way we extend the correct type checking for process composition in those cases where it is possible to add new constraints to obtain a safe process

Table 5. Abstract flow descriptions for the composition of processes

P_1 **has**	P'_2 **has**
input structure where IN **is** I_1	**input structure where** IN **is** I'_2
output structure where OUT **is** O_1 ;	**output structure where** OUT **is** O'_2 ;
P_2 **has**	**flow** P_1-INTO-P_2
input structure where IN **is** I_2	**from** P_1 **.** OUT **to** P_2 **.** IN ;
output structure where OUT **is** O_2 ;	**flow** P_2-INTO-P_3
P_3 **has**	**from** P_2 **.** OUT **to** P_3 **.** IN ;
input structure where IN **is** I_3	
output structure where OUT **is** O_3 ;	

composition. The rationale for this less restrictive process composition follows. In our framework channels define the requirements perceived by the analyst about the process parameters. In a true functional interpretation channels do not exist and the only relevant requirements are function domain and co-domain definitions. the domain analyst does not perceive However, as in the case of an SDF schema, the analyst does not perceive a process as a true function; he/she only defines the class of the input/output parameters. The flow assertion not only states the process composition but should also constrain the communication between processes. This means that the analyst is interested in discovering which constraints — if any — allow information transfer through the flow.

4 Summary and Discussion

We presented a treatment of behavioural aspects which encompasses a wide number of behavioural concepts such as data transformations, physical or conceptual input/output relations and generic processes. The $\mathcal{EDDL}_{DP}$ (i.e. $\mathcal{ALN}$) concept language adopted in this paper is a subset of the CLASSIC language [21], for which there is a corresponding implemented system [16]. In this way, an implementation of our approach is straightforward. We summarise below the main features of behavioural descriptions in the $\mathcal{EDDL}_{DP}$ language.

- *input and output contexts*, i.e. the signature of a black-box transformation function;
- *multiple inheritance of input and output contexts* using the **and** operator;
- *automated classification* of both processes and input/output contexts. These are separately classified according to the subsumption relation;
- *channels*, i.e. process properties which define the input and output contexts of the process itself in terms of their cardinality and data constraints;
- *flow assertions*, i.e. constraints imposing that an output of a source process equals an input of a target process.

Behaviours as functions are ubiquitous in Information Systems Engineering (ISE), where they represent data processing elements. Behaviours as processes are the essence of SDF modelling [8]. In SDF schemes, black-box transformers are connected into a network by way of conceptual or material flows. Our DL-based approach is able to capture SDF schemes. In particular, it models functions in a way which is similar to (but more expressive than) SDF processes, introducing an explicit distinction between function, input, output and flow concepts. More details are shown in [6]. Behaviours are also used in object-oriented modelling approaches such as Object Modelling Technique (OMT, [13]) to represent the functional part of the object model. In our approach, concepts correspond to OMT classes, instances to OMT objects and processes to functions in the functional model Moreover, the structural part and the behavioural one are integrated in a unique scheme. However, our approach does not capture the so-called OMT "dynamic model".

Behaviours are widely used in enterprise modelling (EM) [14], to provide a representation of enterprise functions. The enterprise domain is defined in terms of a set of "domain processes" which fulfil these objectives. Domain processes can be decomposed into a hierarchy of business processes or enterprise activities. The corresponding knowledge contents are easily described using our approach. We allow the construction of business process hierarchies based on functional typing. Moreover, the classification of process contexts let analysts find out the relation between processes that share common input or output elements.

Behaviours are used in the Parameter-Based (PBR, [17]) model for representing domain knowledge in the Engineering of Computer-Based Systems (ECBS). They can be captured as "functions with mechanisms" in our approach. In fact, a "mechanism context component" built using the same input and output context constructors can be easily added to our processes, giving rise to a new kind of behavioural entity. Moreover, given two functions with the same inputs and outputs but with different mechanisms, the first one is a subtype of the second one if and only if the context mechanism of the former subsumes the context mechanism of the latter. Similar relations hold in the other cases.

Acknowledgements

This research has been partially supported by ASI, CNR (Progetto SARI) and MURST 60% (Linguaggi per la modellazione concettuale dei requisiti).

References

1. A. Borgida and P. F. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994.
2. A. Borgida and S. Greenspan and J. Mylopoulos. Knowledge Representation as the basis for Requirements Specification. *IEEE Computer*, pages 82–91, Apr. 1985.
3. B. Nebel. Computational Complexity of Terminological Reasoning in BACK. *Artificial Intelligence Journal*, 34(3):371–383, 1988.

4. B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Number 422 in Lecture Notes In Artificial Intelligence. Springer-Verlag, 1990.
5. D. Calvanese and M. Lenzerini and D. Nardi. A Unified Framework for Class-Based Representation Formalisms. In J. Doyle and E. Sandewall and P. Torasso, editor, *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 109–120. Morgan Kaufmann, 1994.
6. E. Compatangelo and G. Rumolo. Modelling Domain Knowledge with $\mathcal{EDDL}_{DP}$. In A. Sutcliffe and D. Benyon and F. van Assche, editor, *Proc. of the IFIP Joint Working Conference on Domain Knowledge for Interactive System Design (DKISD'96)*, pages 134–148. Chapman & Hall, 1996.
7. E. Compatangelo and G. Rumolo. $\mathcal{EDDL}_{DP} + \mathcal{TDDL}_{DP} =$ a double-level approach to Domain Knowledge Modelling. In H. Kangassalo and J. F. Nilsson, editor, *Information Modelling and Knowledge Bases VIII*. IOS Press, 1997.
8. E. Yourdon. *Modern Structured Analysis*. Prentice-Hall, 1989.
9. F. M. Donini and others. The Complexity of Concept Languages. In J. Allen and R. Fikes and E. Sandewall, editor, *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*, pages 151–162. Morgan Kaufmann, 1991.
10. G. De Giacomo and M. Lenzerini. What's in an Aggregate: Foundations for Description Logics with Tuples and Sets. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 801–807, 1995.
11. H. Kangassalo. COMIC: A system and methodology for conceptual modelling and information construction. *Data & Knowledge Engineering*, 9:287–319, 1992/93.
12. J. Mylopoulos and others. Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, 8(4):325–362, Oct. 1990.
13. J. Rumbaugh and others. *Object-Oriented Modelling and Design*. Prentice-Hall, 1991.
14. K. D. Tham. CIM - OSA: Enterprise Modelling. Technical report, Enterprise Integration Laboratory, University of Toronto, 1996.
15. M. Buchheit and F. M. Donini and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.
16. P. F. Patel-Schneider and others. The CLASSIC Knowledge Representation System: Guiding Principles and Implementation Rationale. *SIGART Bulletin*, 2(3):108–113, 1991.
17. S. A. Friedenthal and H. Lykins. Parameter-Based Representation for Modelling Complex Systems 2. In *IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, pages 65–71. IEEE Computer Society Press, 1996.
18. S. Bergamaschi and C. Sartori. On Taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, 1992.
19. S. Bergamaschi and S. Lodi and C. Sartori. The E/S Knowledge Representation System. *Data & Knowledge Engineering*, 14:81–115, 1994.
20. S. Salomone. An $O(n \log n)$ algorithm for Subsumption in $\mathcal{FL}^-$. In A. Marchetti Spaccamela and P. Mentrasti and M. Venturini Zilli, editor, *Proc. of the 4th Italian Conference on Theoretical Computer Science*, pages 125–139. World Scientific Publishing Co., Oct. 1992.
21. W. A. Woods and J. G. Schmolze. The KL-ONE Family. *Computers and Mathematics with Applications*, 23(2-9):1–50, 1992.