An Integrated Model for Capturing the Process Design Process
Silvio Gonnet‡, Horacio Leone‡, Gabriela Henning†*
† INTEC, Güemes 3450, 3000 - Santa Fe, Argentina
‡ GIPSI- UTN / INGAR, Avellaneda 3657, 3000 - Santa Fe, Argentina

Abstract
This paper presents an integrated deductive object-oriented model that, in relation to a design process, is able to capture (i) the activities, operations and actors that have generated each design product, (ii) the imposed requirements as well as (iii) the rationale behind each adopted decision. Furthermore, it also offers an explicit mechanism to manage the different model versions that have participated during the design process. Thus, the proposed model allows the tracing of such design process and its resulting products.

Keywords: Design Process Support, Version Management, Situational Calculus, Object-Oriented Technology, Deductive Object Base.

1. Introduction
Design is one of the most intricate activities undertaken by chemical engineers, involving the consideration and resolution of many problem-specific and ill-defined tasks. Increasingly powerful computer-aided tools are available to face these complex activities. However, most design knowledge still rests in the minds of experienced designers, being lost as time goes by. Therefore, it is desirable to make it part of a computer support environment. Precisely this issue has been the aim of some other contributions (Roda et al., 2000; Westerberg et al., 1997) who recognize that the design rationale should be a key component of the knowledge of any organization and the history of a design process (DP), captured in a useful form, can form the basis for learning and reuse. To develop such environment it is first necessary to have a comprehensive model of the DP. This contribution addresses this objective by introducing an integrated deductive object-oriented model that not only captures and manages the products being designed, but also the activities that occurred, their associated contexts and the adopted decisions at two different granularity levels, which are discussed in the next section. The proposed model is based on a hybrid approach that combines object-oriented technology (OT), first order logic (FOL) and situational calculus (Reiter, 2001). The model has been specified in the O-Telos language (Jarke et al., 2002), which successfully combines object-oriented and FOL properties, and has been implemented in ConceptBase (Jarke et al., 2002), a deductive object base.

* Author to whom correspondence should be addressed: ghenning@intec.unl.edu.ar
2. Activity and Operation Contexts

In this proposal, the DP is envisioned as a series of design activities that are performed in a given context, called activity context. The activity context is modelled across the different representation spaces that compose it, i.e. (i) the Actor Representation Space, which keeps track of whom performed a given activity, (ii) the Requirement Representation Space, modelling the imposed requirements, (iii) the Artefact Representation Space, modelling the artefact being designed, and (iv) the Decision Representation Space, which captures the underlying rationale behind each design-related decision. On the other hand, activities operate on the results or products of the DP, called design objects by creating, deleting, modifying, and/or using them. In consequence, the design process model has to handle different levels of granularity of contexts. There is an activity context, that requires exploring decision making alternatives about how the DP advances, and an operation context which implements a given decision through the execution of operations which transform the product under development. This originates new contexts, which are themselves subjects of decisions. The upper level, which is depicted in Fig. 1 in a concise view (not showing specific attributes), represents the activities being performed during a DP (Process Representation Space) and its associated contexts (Activity Contexts). The design process is carried out by a set of activities, which may be described at various abstraction levels. An activity may be decomposed into a set of sub-activities.

As it is presented in the class diagram illustrated in Fig. 1, the realization of one activity is guided by one or more requirements, fact that is represented by the guidedBy relationship. Therefore, the DP is interpreted as a series of activities guided by requirements, specifying the functional and non-functional characteristics that a product must satisfy. Because different kinds of activities (e.g. synthesis, decision, etc.) are executed during the DP, the guide that a requirement provides to an activity has diverse roles and depends on the activity type. Then, requirements are used in every activity, but with a different purpose or aim. Regarding activity types, though it is not within the scope of this paper, it is assumed that activities are identified and classified according to the different types that were presented in Eggersmann et al. (2003), and they are guided by the various requirements’ roles defined by Gonnet (2003). Often requirements may not be stated explicitly or in sufficient detail at the beginning of the DP. They are refined and specified more precisely as greater comprehension of the design problem is reached. Then, it is very important to represent how requirements evolve during a project execution. This is analyzed below, where a requirement is represented as a design object.

Activities are performed by actors. An actor may be either an individual (a human being or a computational program) or a team. Teams allow to represent compound skills that are needed for performing activities. The Actor Representation Space presented in this paper extends the actor model introduced by Eggersmann et al. (2001). This extension is made with the aim of answering the questions “Who has performed a given activity?” “Which requirements did the actor try to reach?” Indeed, each activity is related to the actor who executes it (execution relationship in Fig. 1). Each actor may have goals (actor’s goals), which express the actor’s intentions and desires. The actor’s decision of executing a given activity for reaching one or more goals with the final aim of satisfying...
a set of requirements is represented by the promote links among activity, actor’s goal and requirement (Fig. 1). These links reflect the actor’s intention. Moreover, the model shown in Fig. 1 allows to represent the fact that activities are executed by those actors having the necessary skills to carry them out. As seen, the actor, activity and skill classes are connected to each other.

With the aim of representing the rationale associated with the execution of a given activity, the IBIS model (Kunz and Rittel, 1970) is refined in this paper. The IBIS model focuses on articulating key design issues. An issue is a question to be answered and a position is an alternative which exists for solving such issue. We extend this idea by introducing requirements, which specify issues, decomposing positions into artefacts, attributes (PosAttribute), and values (PosValue), and also by adding resolutions, which represent the selection of an alternative with the aim to resolve a requirement (see Fig. 1). An artefact represents the product that it is being designed, whereas attributes and values characterize the position. Then, the different alternative products that arise in the DP are represented by the position concept. A position is qualified by one or more arguments and addresses at least one requirement. An argument either supports or objects a position. It allows to test whether the position is capable of fulfilling the prescribed requirements by the answer relationship.

Positions, artefacts, attributes, values, resolutions and arguments evolve during the execution of a design project and their various states are fundamental for representing the different contexts where an activity is performed. Then, they are represented as design objects (see Fig. 1), and below it is described how their evolution is represented during the DP.

Figure 1. Design Process Representation
The proposed model was completely axiomatized using FOL, but this is not shown due to lack of space. For example, the relationship between an activity and its sub-activities, which is depicted in Fig. 1 by an aggregation link can be represented using FOL by the predicate subActivityOf(aᵢ, aⱼ), that means that aᵢ is a sub-activity of aⱼ. The subActivityOf relationship is transitive (1), irreflexive (2) and asymmetric (3):

\[
\forall a₁, a₂, a₃ \text{ subActivityOf}(a₁, a₂) \land \text{subActivityOf}(a₂, a₃) \Rightarrow \text{subActivityOf}(a₁, a₃) \quad (1)
\]

\[
\forall a \neg \text{subActivityOf}(a, a) \quad (2)
\]

\[
\forall a₁, a₂ \text{ subActivityOf}(a₁, a₂) \Rightarrow \neg \text{subActivityOf}(a₂, a₁) \quad (3)
\]

Then, the model defined using the OT and the set of axioms introduced by means of FOL are both unified in a single model specified in the O-Telos language. In order to exemplify it, Fig. 2 presents the specification of the activity concept. The axioms that allow to deduce new facts, such as (1), were defined as rules, and the axioms that put constraints on the model, such as expressions 2 and 3, were specified as constraints.

The multiple representation spaces mentioned above evolve when each new activity is executed, generating new activity contexts called Model Versions. Then, at a bottom level of granularity, each activity performed during a DP is represented in the Operation Context through the execution of a sequence of operations, which transform the product under development. This model evolution is posed as a history made up of discrete situations. The situational calculus is adopted for modelling such version generation process.

As it was previously introduced, activities operate on the outcomes or products of the DP, called design objects. A design object (Fig. 3) represents any entity that can evolve during a design project. It is represented in two levels, the repository and the versions’ level. The repository level keeps a unique entity for each design object that has been created and/or modified due to model evolution during a design project. This object is called versionable object (o). Furthermore, relationships among the different versionable objects are maintained in the repository. These relationships correspond, according to the notation being used, to the rules that allow associating objects in order to develop syntactically valid models.

On the other hand, the versions’ level keeps the different versions of each design object. These are called object versions (v). The relationship between a versionable object and one of its object versions is represented by the version association. Therefore, a given design object keeps a unique instance in the repository and all versions it assumes in different model versions belong to the versions’ layer.

<table>
<thead>
<tr>
<th>Individual Activity in Class end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual BasicActivity in Class isA Activity end</td>
</tr>
<tr>
<td>Individual CompoundActivity in Class isA Activity with attribute</td>
</tr>
<tr>
<td>subActivity : Activity; infSubActivity : Activity constraint</td>
</tr>
<tr>
<td>irreflexive: $\forall a/CompoundActivity \neg (a \text{ subActivity } a)$; asymmetric: $\forall a₁,a₂/\text{CompoundActivity} (a₁ \text{ infSubActivity } a₂) \Rightarrow \neg (a₂ \text{ infSubActivity } a₁)$</td>
</tr>
<tr>
<td>rule</td>
</tr>
<tr>
<td>transitive: $\forall a₃/a₃/\text{CompoundActivity} a₁/\text{Activity} ((a₃ \text{ subActivity } a₁) \lor (\exists a₂/\text{CompoundActivity} (a₃ \text{ subActivity } a₂) \land (a₂ \text{ infSubActivity } a₁))) \Rightarrow (a₃ \text{ infSubActivity } a₁)$</td>
</tr>
</tbody>
</table>

Figure 2. Activity Specification Using O-Telos
At a given stage during the execution of a design project, the states assumed by the set of relevant design objects, from now on called model version, supply a snapshot description of the state of the DP. According to the proposed model, a new model version $m_n$ is generated when one activity $a$ (a basic activity) is executed. Activity $a$ is materialized by a sequence of operations $\phi$ and the new model version $m_n$ is the result of applying such sequence $\phi$ to the components of a previous model version $m_p$. This is achieved by performing the following evaluation: $apply(\phi, m_p) = m_n$.

The primitive operations that were proposed to represent the transformation of model versions are add, delete, and modify. By using the $add(v)$ operation an object version that did not exist in a previous model version can be incorporated into a successor one. Conversely, the $delete(v)$ operation eliminates an object version that exists in the previous model version. Also, if a design object has a version $v_p$, the $modify(v_p, v_s)$ operation creates a new version $v_s$ of the existing design object, where $v_s$ is a successor version of $v_p$. Thus, an object version $v$ belongs to the model version that arises after applying the sequence of operations $\phi$ to model version $m$, if and only if: (i) $v$ is added when the new model version is created ($add(v) \in \phi \lor modify(v_p, v) \in \phi$); or (ii) $v$ belonged to the previous model version $m$ and it is not deleted when $\phi$ is applied ($delete(v) \notin \phi \lor modify(v, v_* \notin \phi$). From these definitions and by using the format of successor state axioms proposed by Reiter (2001), it is presented a formal specification of the cases in which an object version belongs to a model version. In expression (4), the predicate $belong(v, m)$ is true when the object version $v$ belongs to the model version $m$. From this expression, the object versions that belong to a model version can be determined. Then, it is possible to reconstruct a model version $m_{i+1}$ by applying all the sequences of operations from the initial model version $m_0$.

The proposed scheme is strengthened by the OT, which models the relationships existing among object versions of different model versions, allowing the navigation along the history of the object versions that comprise a given model version. The relationships among object versions are represented by means of explicit links at the versions’ level, named add history, delete history and version history (Fig. 3). Each transformation operation that is applied to a model version incorporates the necessary information to trace the model evolution.

$$(\forall \phi, v, m) \; belong(v, apply(\phi, m)) \iff (add(v) \in \phi \lor (\exists v_p) \; modify(v_p, v) \in \phi \lor$$

$belong(v, m)) \land (delete(v) \notin \phi \land (\forall v_s) \; modify(v, v_s) \notin \phi)$$

(4)

Figure 3. Version Administration Model
The introduced model can be specialized according to the particular domain being tackled. It can be done in terms of the different operations that are applied to the distinct design objects, and in terms of the different design objects that participate in the design process. This is not shown due to lack of space, but it is discussed in Gonnet (2003).

3. Conclusions

The integrated model described in this paper captures the activities, operations and actors that generated each design product, the requirements that were imposed as well as the rationale behind each adopted decision. Furthermore, it also offers an explicit mechanism to manage the different model versions that participated during the DP. Thus, the proposed model allows the tracing of the design process and its resulting products, as well as the analysis of the reasoning line employed during the DP, setting the grounds for learning and future reuse. This is a fundamental step towards the development of computational tools to support the DP and to guide designers in the different activities of a design project. These ideas were proved by applying the proposed model to a case-study that tackles the design of a separation system (Gonnet, 2003). Fig. 4 shows a partial view of the graphical representation of one of the steps of such DP, generated by ConceptBase. It corresponds to the synthesis of the Pre-treatment section of the plant, containing a series of flash units.

Figure 4. Partial View of the of the Pre-treatment Section Synthesis Process that was captured in ConceptBase

Acknowledgments

This work was sponsored by CONICET, FONCYT, Universidad Tecnológica Nacional and Universidad Nacional del Litoral. Authors gratefully acknowledge help received from these institutions.

References


Roda, I., M. Poch and R. Bañares-Alcántara, 2000, Application of a support system to the design of wastewater treatment plants. Artificial Intelligence in Engineering, 14, 45-61.