

Towards a Graphical Language for Process Modelling in Construction

Elisa Marengo¹, Patrick Dallasega², Marco Montali¹, Werner Nutt¹

¹ Faculty of Computer Science, ² Faculty of Science and Technology,
Free University of Bozen-Bolzano, Italy
`firstname.lastname@unibz.it`

Abstract. To guarantee the success of a construction project, a detailed process model specification is essential. The peculiarities of the domain, like the high number of details, the participation of multiple parties with different strategic goals, and the need of flexibility, make generic process modelling languages unsuitable. For this reason, the PRECISE methodology has been recently introduced by a group of civil engineers. PRECISE introduces a domain-specific graphical language, successfully employed in real construction projects. However, currently the language suffers of some limitations and ambiguities that prevents the development of tools for supporting the project management and possibly implementing automatic functionalities. In this paper, we highlight the problems related to the language and propose an extension to overcome them. The resulting language can then be formalized in Linear Temporal Logic formula over finite traces, paving the way for the development of (automatic) supporting tools.

Keywords: Process Modelling in Construction; Collaborative Design of a Process; Automatic Verification

1 Introduction

Process management in construction aims at defining and executing a construction process, guaranteeing a high quality of the final product and limiting time and cost overruns. Some peculiarities of the domain make this task challenging in reality. One aspect concerns the high number of details to be considered: besides defining the tasks to be performed, it is also necessary to specify the different locations where each of them is foreseen and the resources needed, potentially shared among different tasks. An efficient management of resources is important and aims at maximizing their usage while ensuring their availability when needed. These aspects require coordination among the different crafts that participate in the construction process. In fact, companies prefer to coordinate among themselves rather than working for general contractors, because the profit margins are lower in this case. As a consequence, before the process can start,

Copyright © by the paper's authors. Copying permitted only for private and academic purposes.

In: S. España, M. Ivanović, M. Savić (eds.): Proceedings of the CAiSE'16 Forum at the 28th International Conference on Advanced Information Systems Engineering, Ljubljana, Slovenia, 13-17.6.2016, published at <http://ceur-ws.org>

the companies need to agree on the way the activities will be carried out. Accordingly, modelling the process is important for an efficient management of the resources and to ensure good quality of the final result. The process model needs to be flexible so that it can be quickly adapted to face unpredictable events (e.g. bad weather conditions) or changes of the requirements (which often occur in construction projects).

In the literature, two main approaches for process modelling exist: *i)* the *data-centric* one, which identifies a number of relevant entities and describes the process in terms of their possible evolution; and *ii)* the *activity-centric* one, which focuses on the activity control flow, representing the data in a very limited way (e.g., BPMN [10]). Both approaches aim at process modelling in general domains. As a consequence, as discussed also in [9,11], they need to be general enough to accommodate the needs of many application domains, inevitably failing in capturing all the specificities of a particular application domain.

To meet the peculiarities of the construction domain, a group of civil engineers introduced a new domain-specific methodology, named PRECISE (Process REliability in ConstructIon for SmEs) [5]. It has been developed by re-engineering two completed construction projects and has been tested in several real construction projects, like in the construction of the facade of the extension of the hospital of Bolzano [6] and in the project “Softbridge” in Oxford [6]. The methodology foresees three phases: *i)* the *modelling* of the process; *ii)* the *scheduling* of the activities to be performed on-site; and *iii)* the *monitoring* of the progress of the work on-site. Considering the state of the art in construction, none of the existing approaches consider all the three phases in an integrated way. In particular, the most predominant approach used in construction is BIM (Building Information Modelling), which is supported by a variety of commercial tools. However, these tools are mainly oriented to the the building design and to the scheduling, considering the process modelling only marginally.

In this paper we focus on the process modelling phase of the PRECISE methodology for which a domain-specific graphical language has been introduced. The modelling is conceived as a collaborative activity to be performed by all the key actors taking part in the construction project. The resulting process model encompasses all the details that are needed for the different companies to synchronize and coordinate their activities on-site. A model constitutes the basis for the subsequent phase of the methodology, which is the scheduling of the activities to be performed on-site. According to PRECISE, during the modelling the following elements are defined: *i)* a representation of the building in terms of locations, suitable to locate the tasks to be executed; *ii)* the resources that are needed; *iii)* the tasks to be executed and on which the different crafts have to synchronize; and *iv)* the dependencies on the execution of the tasks.

In this setting, two main modelling requirements emerge: the modelling language should be easy to use and understand, and a model must be non-ambiguous in defining the activities of the different companies and the dependencies among them. This is important both to achieve the desired coordination among the

companies and for the implementation of (automatic) supporting tools. Currently, the language is used with limited support of IT and automatic tools (e.g., process models are defined on magnetic whiteboards) and it presents some ambiguities that are usually solved by annotating a model with additional information (e.g., with notes and comments) that is not part of the language. In this paper, after presenting the limitations of the PRECISE modelling language (Section 2), we propose an extension which allows one to overcome the described limitations (Section 3). We conclude by describing the future work of formalizing the language in terms of Linear Temporal Logic (LTL) (Section 4).

2 The PRECISE Process Modelling Language

According to the PRECISE methodology a process model is graphically represented as a graph where nodes are tasks and edges are dependencies among them. A task is represented as in Fig. 1 and contains the following information: ① a unique task *id*; ② the *number of workers* and ③ the *number of days* needed to complete the task; ④ the responsible *craft*; ⑤ the *activity* to be executed; ⑥ a space for annotations; and the information on the *locations* where the task is foreseen. The PRECISE language does not define a standard for the representation of the location. The figure refers to an example where locations were identified as pairs of ⑦ *section*, which defines the technological content of an area (e.g. room, swimming pool) and ⑧ *level*, i.e. a floor of the building. Tasks are connected one with another by means of *precedence dependencies*, represented as arrows. These declaratively specify the set of constraints that a schedule needs to satisfy. Accordingly, several schedules may satisfy a process model.

Fig. 1 reports an excerpt of the process model of a real construction project for the construction of a hotel [5]. The hotel consists of four floors, one of which under ground (*f0*, *f1*, *f2*, *u1* in the picture). The model starts with the activity CONSTRUCTION SITE PREPARATION, which concerns the delimitation of the construction site. According to the specification, this task can be performed by a crew of 4 people working as SITE EQUIPPERS (SE) and requires 5 working days to be completed. It must be executed in all sections of all levels. The EXCAVATION can start after the site has been prepared and after the task CONCRETE POURING can be executed. At this point, a first ambiguity in the model arises. Indeed, according to the previous usage of precedence dependencies, it could seem that concrete has to be poured in all locations before the process can progress. It is also not clear which task should be performed next among: *i*) SCAFFOLDING INSTALLATION by the Scaffolder (Sc); *ii*) ELECTRICITY CONNECTION by the Electrician (El) and WATER AND GAS CONNECTION by the Plumber (Pl); and *iii*) PIPES INSTALLATION. The intention of the model is to capture that first the concrete has to be poured for the underground level. Then, at the same level, the connections for electricity, water and gas can be made, before the task EXCAVATION FILLING. After this, the scaffolding can be installed for the first floor, and then the concrete can be poured for the same floor. This sequence of SCAFFOLDING INSTALLATION and CONCRETE POURING repeats for all floors until the last is reached. At

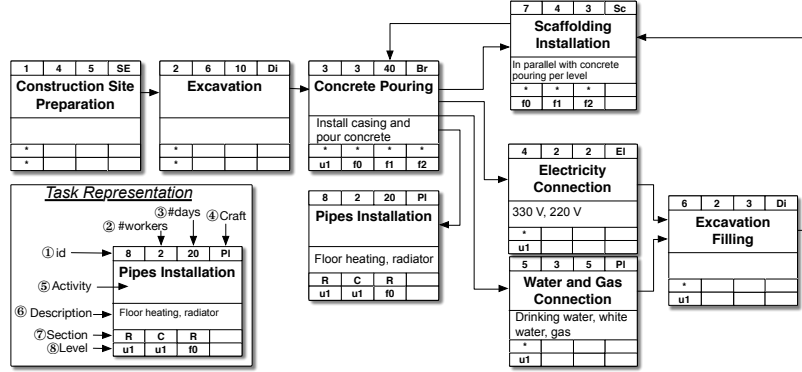


Fig. 1. Excerpt of a real process model for the realization of a hotel. The wild-card * represents all sections and/or levels. The project foresees four floors (u1, f0, f1, f2). The sections are Room (R) and Corridor (C). The involved crafts are Site Equipper (SE), Digger (Di), Brick Layer (Br), Plumber (PI), Electrician (El), and Scaffolder (Sc).

this point, the PIPES INSTALLATION can start in the room (R) and in the corridor (C) of the underground floor (u1), and in the room (R) of the ground floor (f0). Note that these ambiguities are solved by expressing further requirements in the *description* field of the tasks (e.g. as in SCAFFOLDING INSTALLATION description). However, a model should be a non-ambiguous agreement among the parties, so as to allow one to determine, for instance, whether it is satisfiable or whether a schedule is compliant with a model.

As highlighted by this example, some limitations of the language emerge. We describe them in the following.

(a) *Representation of locations.* The PRECISE modelling language does not foresee a unique representation of locations, which can be specified in different ways depending on the project. A standard way for representing them would allow for the implementation of tools for supporting the process modelling, and for the definition and reuse of common modelling patterns in different projects.

(b) *Ordering on the execution of a task in different locations.* A task is represented together with a set of locations where it is foreseen. In some cases, the order in which the task is executed in these locations matter. For instance, considering the task CONCRETE POURING: its execution has to be performed from the lowest level to the highest, but this is not captured in the model.

(c) *Scope of a dependency.* Currently, when two tasks are connected by a dependency, it is not clear whether the precedence constraint applies at the level of task or of location. For instance, the meaning of the dependency between the tasks CONSTRUCTION SITE PREPARATION and EXCAVATION is that the first task needs to be finished in all locations before the second one can start. The dependency between tasks SCAFFOLDING INSTALLATION and CONCRETE POURING, instead, applies at the level of floor.

(d) *Different kinds of precedence constraints.* When the scope of a dependency is at the level of location, it is not clear whether the first task has to wait for

the second one to be completed in the same location before it can progress somewhere else (like a *chain* execution), or this is not required. In the example, for instance, not only SCAFFOLDING INSTALLATION has to be performed before CONCRETE POURING, but the installation of the scaffolding cannot progress to a subsequent floor until the pouring of the concrete is finished in the previous one. Thus the two tasks have to be executed in chain. This is not always the case. If we consider tasks PAINTING and CLEANING, then the former task can start in every location without waiting for the second one to be finished.

3 Extending the PRECISE Modelling Language

In this section we propose an extension of the PRECISE modelling language, making it able to address all critical aspects discussed in Section 2.

(a) *Representation of locations.* From the real projects where the methodology has been applied we identified the following elements as suitable to represent a building in an abstract way [4]:

Sectors: identify parts of the building whose construction process progresses (almost) in an independent way one from the other (e.g., different wings).

Levels: the floors of the building (e.g., f0, f1, f2).

Sections: identify the technological content of certain areas (e.g., swimming pool, corridor, room and such like).

Units: a unit is a number used to enumerate locations of the same kind. For instance, different rooms at the same floor have different unit numbers.

We will refer to this representation as the *construction area hierarchy*. With slight abuse of terminology, we will refer to a *construction area (CA)* as a location specified at different levels of the hierarchy, that is a sector, a pair sector-level, a triple sector-level-section or a tuple sector-level-section-unit are all construction areas. The latter, in particular, provides the highest level of detail in representing a location. We refer to it as *construction unit*. For instance, the tuple $\langle A, f1, room, 2 \rangle$ represents the construction unit at sector A, floor 1, room 2.

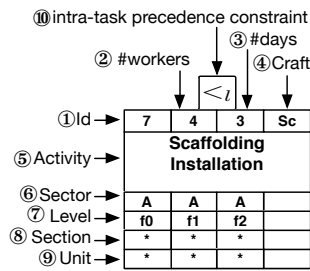


Fig. 2. Representation of a task.

A task is then foreseen to be performed in a number of construction units. To represent this information we extend the representation of a task, as reported in Fig. 2.

As shown, a construction unit is represented in terms of ⑥ sector, ⑦ level, ⑧ section and ⑨ unit. The wild-card * can be used instead of the *unit* number (e.g., $\langle A, 1, room, * \rangle$) to identify all units of a *section* at a particular *level* of a *sector*. The wild-card can also be used for the other elements of a construction unit, if all elements below in the CA hierarchy are also *.

(b) *Ordering on the execution of a task in different locations.* In some cases it is necessary to specify *intra-task precedence constraints*, expressing that the execution of a task in different construction areas must be performed following a given order (as for the task SCAFFOLDING INSTALLATION in Fig. 1, which has to be performed from the lower level up to the top one). An intra-task precedence constraint can be specified at different levels of the CA hierarchy to capture, for instance, that a task needs to be performed following an order on the construction units, on the floors (e.g., from the first to the last floor, regardless of the execution in the units within a floor) on the sections or on the sectors. To express this kind of requirements we introduce the notion of *scope* which identifies at which level of the CA hierarchy the constraint applies. Possible scopes are *sector*, *level*, *section* or *unit* and are graphically represented with the symbols \langle_{sr} , \langle_1 , \langle_{sn} and \langle_u resp., as depicted in $\textcircled{\text{a}}$ in Fig. 2.

(c) *Scope of a dependency.* To specify at which level a dependency applies we introduce a notion of scope which is similar to the one introduced for the intra-task precedence constraints. The scope allows one to specify that, for instance, a dependency between two tasks applies at scope *level*, i.e. in every floor where the two tasks are foreseen, the first task must be performed before the second. Additionally, we consider *task* to be a possible scope for the dependencies. This is used to express that a certain task must be completed (everywhere) before another task can start. Graphically, the scope of a dependency is specified by annotating an arrow with the symbols \mathfrak{t} , \mathfrak{sr} , \mathfrak{l} , \mathfrak{sn} and \mathfrak{u} to represent task, sector, level, section and unit scopes resp. (see Table 1).

(d) *Different kinds of precedence constraints.* By taking inspiration from Declare [1], we extend the possible kinds of dependency that can be expressed between two tasks and provide a graphical representation. The set of dependencies is reported in Table 1. The types of dependency that we identified are described in the following (note that the language can be easily extended).

EXISTENCE. Drawing a task in a model corresponds to an existence constraint, which requires the task to be sooner or later executed in all construction units specified in the task. If an intra-task precedence constraint is specified, as in Fig. 2, this means that the order in which the construction areas are specified matters. The dependencies that we present in the following can be equally applied to tasks with and without intra-task precedence constraints.

PRECEDENCE. This dependency involves two tasks and it is annotated with a scope. If the scope is *task*, then the TASK PRECEDENCE applies, requiring the first task to be finished in all specified construction units before the second one can start. If the scope is not task, then the CA PRECEDENCE relation applies. In particular, this applies to all construction areas (at the specified scope) that the two tasks share. In the areas that are not shared, the two tasks can be performed independently. Graphically, a precedence dependency is represented with an arrow annotated with the scope.

CHAIN PRECEDENCE. This dependency between two tasks A and B , is used to specify that in each construction area (at the specified scope) shared between A



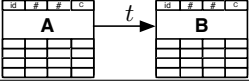
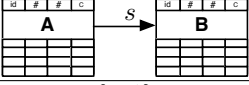
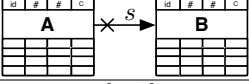
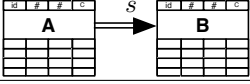
<u>EXISTENCE</u> : $existence(A:task, Lo:\{CA\})$	
Task A must be executed in all construction areas Lo	
<u>ORDERED EXISTENCE</u> : $ordered_existence(A:task, Lo:\{CA\}, <_s:order)$	
Task A must be executed in all construction areas in Lo following the order given by $<_s$	
<u>TASK PRECEDENCE</u> : $task_precedence(A:task, Lo_A:\{CA\}, B:task, Lo_B:\{CA\})$	
Task B cannot start in any construction area in Lo_B until task A is finished in all construction areas in Lo_A	
<u>CA PRECEDENCE</u> : $CA_precedence(A:task, Lo_A:\{CA\}, B:task, Lo_B:\{CA\}, s:scope)$	
Task B cannot start in a construction area that is shared with task A at scope level s , until A is finished	
<u>CHAIN PRECEDENCE</u> : $A_chain_B(A:task, Lo_A:\{CA\}, B:task, Lo_B:\{CA\}, s:scope)$	
For all construction areas shared between A and B at scope s , once A is started then B must be performed in the same construction area before A can progress	
<u>STRICT SEQUENCE</u> : $A_sequence_B(A:task, Lo_A:\{CA\}, B:task, Lo_B:\{CA\}, s:scope)$	
For all construction areas shared between tasks A and B at scope level s , task A must be done before B and B must start immediately after A is finished	

Table 1. Existence and Dependency Relations. (CA stands for Construction Area)

and B *i*) A must be executed before B , and *ii*) once A is started in a CA c_1 it cannot progress in another CA c_2 until B is finished in c_1 . Graphically, this is captured with an arrow with an X symbol closed to A , to capture that, in order to progress, A has to wait for B .

STRICT SEQUENCE. In some cases it is necessary to specify that a certain task must be started immediately after another one is finished. A *strict sequence* dependency allows to express this requirement and is represented with a double arrow. Similarly as before, it is applied only to construction areas shared among the two tasks and it is specified at a certain scope.

4 Conclusion and Future Work

In this paper we presented the PRECISE methodology for construction project management. In particular, we focused on the process modelling phase and highlighted some limitations of the original language, on the one hand related to the lack of abstractions needed for capturing relevant constraints, and on the other hand connected to the ambiguity of the language, which prevents the implementation of automatic supporting tools. To overcome these issues we proposed an extension of the language, which also paves the way towards the definition

of its formal semantics. In particular, we are now formalizing the different dependencies in terms of Linear Temporal Logic [3] over finite traces [8], taking inspiration from previous approaches focused on constraint-based, declarative process models [2,7]. Intuitively, the execution of a task in a certain construction unit is formalized with the “eventually” (\diamond) operator. Precedence dependencies, instead, are formalized in terms of the “until” (U) operator. The adoption of this well-known logic would allow to adopt existing tools and techniques [3,2,7] as a starting point for the development of automatic tools to support the process modelling. In particular, we will focus on: *i) Soundness of a model.* The aim is to check whether there exists a schedule that satisfies the model. This kind of check may be performed at the control flow level, checking whether the set of dependencies is satisfiable, and at the level of resources, checking whether given a certain number of available resources the model can be enacted. *ii) Compliance of a schedule.* A process model specifies the coordination among the different companies involved in a construction project. Checking that a schedule is compliant with the model corresponds to checking that the coordination is actually implemented as designed. This check can be performed by adopting well known LTL model checking techniques over finite traces [3,7].

Acknowledgements. This work was done within the research projects MAGIC, financed by the Province of Bolzano, and MoMaPC financed by the Free University of Bozen-Bolzano. The authors thank Ognjen Savković for the discussions.

References

1. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Dagstuhl Seminar Proceedings (2006)
2. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science-R&D* 23(2) (2009)
3. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: *Model Checking*. MIT Press (1999)
4. Dallasega, P., Marengo, E., Nutt, W., Rescic, L., Matt, D.T., Rauch, E.: Design of a Framework for Supporting the Execution-Management of Small and Medium sized Projects in the AEC-Industry. In: DCEE (2015)
5. Dallasega, P., Matt, D.T., Krause, D.: Design of the Building Execution Process in SME Construction Networks . In: DCEE (2013)
6. Dallasega, P., Rauch, E., Matt, D.T.: Sustainability in the Supply Chain Through Synchronization of Demand and Dupply in ETO-Companies. *CIRP Elsevier* (2015)
7. De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring Business Metaconstraints Based on LTL&LDL for Finite Traces. In: *BPM* (2014)
8. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In: *AAAI*. pp. 1027–1033 (2014)
9. Dumas, M.: From Models to Data and Back: The Journey of the BPM Discipline and the Tangled Road to BPM 2020. Keynote. In: *BPM. LNCS*, vol. 9253 (2015)
10. Object Management Group: *Business Process Modeling Notation Version 2.0*. Tech. Rep., Object Management Group Final Adopted Specification (2011)
11. Singh, M.P.: *Agent Communication Languages: Rethinking the Principles*. In: *Communication in Multiagent Systems, Agent Communication Languages and Conversation Polocies. LNCS*, vol. 2650 (2003)