

Figure 4: Process composition graph for the assembly line process.

Objects:

$Vp = [line : A1, interval : [-50, 50],$
 $anchored_processes : \{[machine : M1, anchor : 10, processes : \{Ep\}], [machine : M2, anchor : 3, processes : \{CUp\}]\}$
 $Ep = [interval : [-30, 40], anchored_processes : \{[anchor : 20, processes : \{CWp\}],$
 $[anchor : 10, processes : \{Gp\}], [anchor : 15, processes : \{Gp\}]\}$
 $CWp = [interval : [-10, 20], anchored_processes : \{[anchor : 10, processes : \{R2p\}],$
 $[anchor : -3, processes : \{D2p\}]\}$
 $R2p = [robot : R2, interval : [0, 10],$
 $anchored_processes : \{ [anchor : 5, processes : \{P2p\}], [anchor : 0, processes : \{CCp\}] \}$
 $D2p = [robot : D2, interval : [-5, 20],$
 $anchored_processes : \{ [anchor : 2, processes : \{P1p\}], [anchor : 13, processes : \{CCp\}],$
 $[anchor : 0, processes : \{DHp\}] \}$
 $P2p = [interval : [2, 4], from : (15, -4), to : (64, 9), object : chain]$
 $CCp = [interval : [4, 7]]$
 $DHp = [interval : [0, 5]]$
 $P1p = [interval : [-5, 3], from : (5, 4), to : (4, 9), object : wheel]$
 $Gp = [interval : [5, 7]]$
 $CUp = [interval : [-10, 30], anchored_processes : \{[anchor : -13, processes : \{Cp\}] \}$
 $Cp = [interval : [10, 40], anchored_processes : \{[anchor : 20, processes : \{Sp\}] \}$
 $Sp = [interval : [-10, 20], anchored_processes : \{[anchor : 0, processes : \{CWp\}] \}$

Classes:

$Assembly_line_process = \{Vp, Ep, CWp, Gp\}$
 $Robot_schedule = \{R2p, D2p\}$
 $Motion_process = \{P1p, P2p\}$
 $Shared_assembly_process = \{CCp\}$

Relations:

$robots = \{R2p, D2P\}$

Figure 3: An instance of the assembly line schema.

```

class Process
    type [ interval : Time-Interval ]
end;

class Complex_process inherit Process
    type [ anchored_processes : [ anchor : Time-Point, processes : {Process} ], ]
end;

class Assembly_line_process inherit Complex_process
    type [ line : Assembly_line,
          anchored_processes : {[ machine : Machine, anchor : Time-Point, process : Complex_process ]},
    ]
end;

class Shared_assembly_process inherit Complex_process, Shared_process
end;

class Robot_schedule inherit Complex_Process
    type [ robot : Robot, movements : [ anchor : Time-Point, processes : {Motion_process}]]
end;

class Motion_process inherit Process
    type [ from : Coordinate, to : Coordinate, object : Part]
end;

relation robots : Robot_schedule

```

Figure 2: Database schema for assembly line.

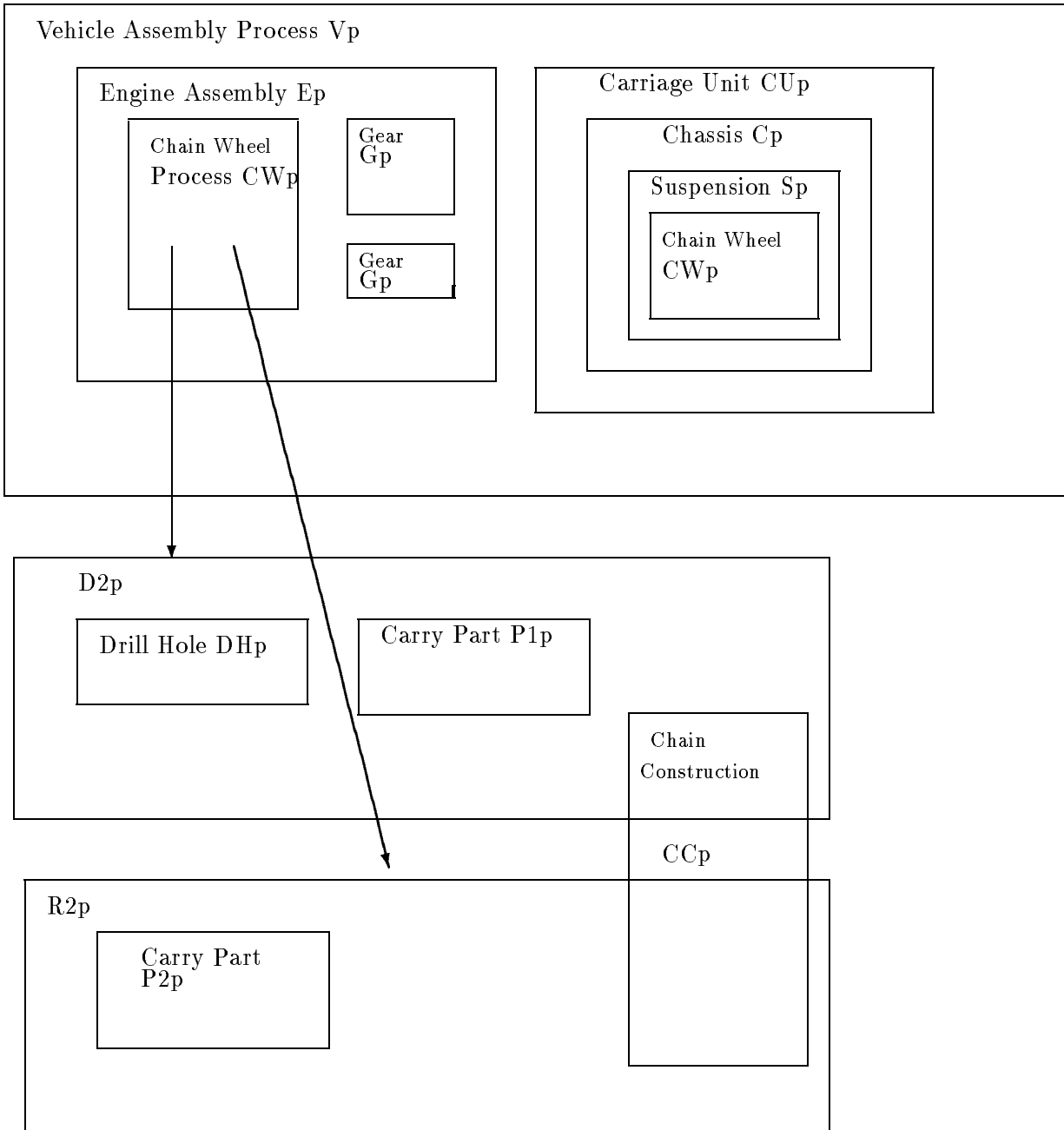


Figure 1: An assembly line process.

- [WD92] G.T.J. Wu and U. Dayal. A uniform model for temporal object-oriented databases. In *Proc. IEEE Data Engineering Conf.*, pages 584–593, 1992.

- [NA89] S.B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Systems*, 49:147–175, 1989.
- [PC92] J.M. Pratt and M. Cohen. A process-oriented scientific database model. *SIGMOD Record*, 21(3):17–25, September 1992.
- [Ram93] K. Ramamritham. Real time databases. *Distributed and parallel databases*, 1(2):199–226, April 1993.
- [RS91] E. Rose and A. Segev. TOODM - a temporal object-oriented data model with temporal constraints. In *Intl. Conf. on Entity-Relationship Approach*, pages 205–229, 1991.
- [SBE93] A. Shuster and D. Ben-Eliyahu. Rtime data model implementation for the food processors parts industry of kibutz mephalsim. Technical report, Department of Math. and CS, Ben-Gurion University, Beer Sheva, Israel, December 1993. In Hebrew.
- [SC91] S.Y.W. Su and H-H.M. Chen. A temporal knowledge representation model OSAM*/T and its query language OQl/T. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 431–442, 1991.
- [Sno87] R. Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, 1987.
- [Sno90] R. Snodgrass. Temporal databases: status and research directions. *ACM SIGMOD Record*, 19(4):83–89, 1990.
- [Tau91] B. Tauzovich. Towards temporal extensions to the Entity-Relationship model. In *Intl. Conf. on Entity-Relationship Approach*, pages 163–179, 1991.
- [TC90] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal completeness. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 13–23, 1990.
- [TG89] A.U. Tansel and L. Garnett. Nested historical relations. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 284–293, 1989.
- [TLW91] C. Theodoulidis, P. Loucopoulos, and B. Wangler. The Entity-Relationship time model and the conceptual rule language. In *Intl. Conf. on Entity-Relationship Approach*, pages 181–204, 1991.
- [Tou86] D. Touretzky. *The Mathematics of Inheritance Systems*. Morgan Kaufmann, 1986.

- [EW90] R. Elmasri and G. Wu. A temporal model and query language for ER databases. In *Proc. IEEE Data Engineering Conf.*, pages 76–83, 1990.
- [FD82] J. D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison Wesley, 1982.
- [For84] K.D. Forbus. *Qualitative Process Theory*. PhD thesis, Artificial Intelligence Laboratory, MIT, 1984.
- [Gad88] S.K. Gadia. A homogeneous relational model and query language for temporal databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.
- [Hay85] P.J. Hayes. The second naive physics manifesto. In J. Hobbs and B. Moore, editors, *Formal theories of the commonsense world*, pages 1–36. Ablex Publishing Corporation, 1985.
- [KG77] K. Kahn and G.A. Gorry. Mechanizing temporal knowledge. *Artificial Intelligence*, 9:87–108, 1977.
- [KM90] A. Kemper and G. Moerkotte. Access support relations in object bases. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 364–374, 1990.
- [KS92] W. Kafer and H. Schoning. Realizing a temporal complex-object data model. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 266–275, 1992.
- [LJ88] N. Lorentzos and R. Johnson. Extending relational algebra to manipulate temporal data. *Information Systems*, 13(3):289–296, 1988.
- [MMK93] M. Marefat, S. Malhotra, and R.L. Kashyap. Object-oriented intelligent computer-integrated design, process planning, and inspection. *IEEE Computer*, 26(3):54–65, March 1993.
- [MPB92] R. Maiocchi, B. Pernici, and F. Barbic. Automatic deduction of temporal information. *ACM Transactions on Database Systems*, 17(4):647–688, December 1992.
- [MS91] L.E. McKenzie and R.T. Snodgrass. Evaluation of relational algebras incorporating the time dimension in databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.
- [MTM91] J.R. Moyne, T.J. Teorey, and L.C. McAfee, Jr. Time sequence ordering extensions to the Entity-Relationship model and their application to the automated manufacturing process. *Data & Knowledge Engineering*, 6:421–443, 1991.

- [All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [BCD89] F. Bancilhon, S. Cluet, and C. Delobel. A query language for the O_2 object-oriented database system. In *Proc. Second Intl. Workshop on Database Programming Languages*, pages 122–138, 1989.
- [Bee89] C. Beeri. Formal models for object-oriented databases. In *Proc. First Intl. Conf on Deductive and Object-oriented databases*, pages 370–395, Kyoto, Japan, December 1989.
- [Bla91] H. Blanken. Implementing version support for complex objects. *Data & Knowledge Engineering*, 6:1–25, 1991.
- [BM89] M. Balaban and N.V. Murray. The logic of time structures: temporal and nonmonotonic features. In *Intl. Joint Conf. on Artificial Intelligence*, pages 1285–1290, 1989.
- [BM93] M. Balaban and N.V. Murray. Interleaving time and structure. Technical Report TR 93-14, Department of Mathematics and Computer Science, Ben-Gurion University, Israel, and Department of Computer Science, SUNYA, 1993. Submitted.
- [BS92] M. Balaban and C. Samoun. Object-oriented music pieces (oomp) — a first step in the formal conceptualization of music. In *Israeli Symposium on Artificial Intelligence and Computer Vision*, volume 9, 1992.
- [BS93] M. Balaban and C. Samoun. Hierarchy, time and inheritance in music modelling. *Languages of Design*, 1(2), 1993.
- [BS94] M. Balaban and S.E. Shimony. Structured plans with sharing and repetition. Technical Report FC 94-11, Department of Mathematics and Computer Science, Ben-Gurion University, Beer Sheva, Israel, 1994. Submitted.
- [CKO92] B. Curtis, M.I. Kellner, and J. Over. Process modelling. *Communications of the ACM*, 35(9):75–90, September 1992.
- [D⁺91] O. Deux et al. The O_2 system. *Communications of the ACM*, 34(10):34–48, October 1991.
- [Dav90] E. Davis. *Representations of Commonsense Knowledge*. Morgan Kaufmann, 1990.
- [Dea89] T. Dean. Using temporal hierarchies to efficiently maintain large temporal databases. *Journal of the ACM*, pages 687–718, 1989.

period in Vp does not include the time point under consideration (the relativized period is $[-7, 33]$, and the time point is -10), search of CUp 's subgraph is redundant, by the containment constraint. Note, however, that since the process composition graph is not necessarily a tree, parts of an eliminated subgraph can still be searched via other nodes (like searching CWp as a subprocess of Ep). Further optimization can be obtained by keeping a list of nodes that are roots of graphs that were already searched.

7 Conclusions

We have presented the Rtime model whose major contribution is an explicit representation of temporal processes, their hierarchical composition specified with relative times, and the possibility to describe shared and repeated process execution. The direct representation of the temporal hierarchy permits using an extension of standard object-oriented query languages for manipulating process information in Rtime. The data model has been implemented on top of the O_2 DBMS [D⁺91]. It has been applied to record the processes' timing in an industrial plant for producing parts of a food processor, in Kibutz Mephalsim ([SBE93]).

The processes handled in this paper are assumed contiguous, and hence their periods of activities are modeled by time intervals. This assumption can be waived in the future by replacing temporal elements for time intervals. It will not require any change in the Rtime model itself; only the algebra of interval operations and relations should be replaced by that of temporal elements. An important direction for future research is incorporating inferences on processes into the data model such as inferring the set of agents participating in a shared process from its containing processes. Such inferences demand a notion of inheritance in terms of the temporal aggregation hierarchy [Tou86, BS92].

Acknowledgment

We are indebted to an anonymous referee, that suggested the usage of temporal elements, thereby generalizing our original notion of contiguous processes into interruptible ones.

References

- [A⁺89] M. Atkinson et al. The object-oriented database system manifesto. In *Proc. Intl. Conf. on Deductive and Object-Oriented Databases*, pages 40–57, Kyoto, Japan, December 1989.

```

SELECT  p
FROM    Process p, Time-Interval IntpVp
WHERE   p IN components(Global.Vp.process) AND IntpVp IN relativize(Global.Vp.process, p) AND
        IntpVp.lb + 2.30.1900 ≤ 14 : 23@2.30.1900 ≤ IntpVp.ub + 2.30.1900

```

4. Queries comparing the component processes of a parent process. For example,

Find all subprocesses of Vp that start before Ep .

```

SELECT  p
FROM    Process p, Time-Interval IntpVp, Time-Interval IntEp
WHERE   p IN components(Vp) AND IntpVp IN relativize(Vp, p) AND
        IntEp IN relativize(Vp, Ep) AND IntpVp.lb ≤ IntEp.lb

```

The four kinds of temporal queries form the basis for expressing more complex temporal queries, and for combining temporal and nontemporal queries. Following is an example of a query for finding all shared subprocesses of any two global occurrences of processes $R2p$ and $D2p$ (see Figure 3).

```

SELECT  p
FROM    Shared_process p, Global-Time-Interval Int, Global-Process gr, Global-Process gd
WHERE   gr.process = R2p AND gd.process = D2p AND p IN components(R2p) AND p IN components(D2p)
        AND Int IN relativize(R2p, p) + gr.anchor AND Int IN relativize(D2p, p) + gd.anchor

```

The query looks for a shared process which is a subprocess of both $R2p$ and $D2p$, and has a single global interval when relativized with respect to any two global occurrences of $R2p$ and $D2p$.

6.4 Efficient Computation of Temporal Queries

Evaluation of the **components** operator requires the computation of transitive closure on the *processes* attribute. This can be performed efficiently by means of path indices [KM90].

Computation of temporal queries that involve relativization of time lines can be accelerated by taking the containment constraint into consideration. The idea is to use the containment constraint as a *filter* for avoiding redundant search in the process composition graph. Consider, for example, the second query above. For a process like CUp , whose relativized

The temporal queries presented in the following section use the **relativize** operator, which is a special case of the **translate** operator, defined as follows:

For processes $subp$ and $parentp$ such that $subp$ is a component of $parentp$ (i.e., $Paths(parentp, subp)$ is not empty),

$$\mathbf{relativize}(parentp, subp) = \mathbf{translate}(parentp, subp, subp.interval)$$

While in the queries below we do not use **translate** in its full generality, it should be noted that it is useful in manipulations and updates involving components of complex processes.

6.3 Temporal Queries

The relative time specification of the internal composition of processes suggests that there are four kinds of temporal queries inherent in the Rtime data model. These temporal queries are expressed, using the **relativize** operator, defined in the previous subsection, and the operator **components** that was defined in Subsection 6.1.

1. Queries on component processes of a parent process, in terms of the components' time order. For example,

Find all subprocesses of Vp with duration greater than 20.

```
SELECT  p
FROM    Process p
WHERE   p IN components(Vp) AND (p.period.ub - p.period.lb) > 20
```

2. Queries on component processes of a parent process, in terms of the parent process' time order. For example,

Find all subprocesses of Vp that are active 40 seconds after its start (i.e., active at $Vp.period.lb + 40$).

```
SELECT  p
FROM    Process p, Time-Interval IntpVp
WHERE   p IN components(Vp) AND IntpVp IN relativize(Vp, p)
        AND IntpVp.lb ≤ Vp.period.lb + 40 ≤ IntpVp.ub
```

3. Queries on global occurrences of subprocesses of a global process. For example,

Find all subprocesses of $Global_Vp$ (from Section 5) that have global occurrences active at 14 : 23 on Feb 30th, 1900.

to the three possible values of its third argument it yields the sets:

$$\begin{aligned} & \{ t + t' \mid t' \in \text{disp}(\text{Paths}(p_1, p_2)) \} \\ & \{ [t_1, t_2] + t' \mid t' \in \text{disp}(\text{Paths}(p_1, p_2)) \} \\ & \{ p' \mid p' = \mathbf{displace}(p_2, t') \wedge t' \in \text{disp}(\text{Paths}(p_1, p_2)) \} \end{aligned}$$

where $\mathbf{displace}(p, t)$ is a function that maps a process value p and a time point t to a process value, defined as follows:

1. If p is atomic:

$$\mathbf{displace}(p, t).period = p.period + t,$$

and $\mathbf{displace}(p, t)$ has all other attributes of p .

2. If p is complex: $\mathbf{displace}(p, t).period = p.period + t$,

If $p.anchored_processes = \{ [anchor : t_1, processes : P_1], \dots, [anchor : t_n, processes : P_n] \}$, where P_i 's are sets of processes, then

$$\mathbf{displace}(p, t).anchored_processes = \{ [anchor : t_1 + t, processes : P_1], \dots, [anchor : t_n + t, processes : P_n] \}.$$

and $\mathbf{displace}(p, t)$ has all other attributes of p .

- If there is some path from p_2 to p_1 in the process composition graph, then it translates the temporal information in its third argument from p_2 's time line to p_1 's time line, i.e. going down in the composition graph. In that case it yields the sets:

$$\begin{aligned} & \{ t - t' \mid t' \in \text{disp}(\text{Paths}(p_2, p_1)) \} \\ & \{ [t_1, t_2] - t' \mid t' \in \text{disp}(\text{Paths}(p_2, p_1)) \} \\ & \{ p' \mid \mathbf{displace}(p_2, -t') \wedge t' \in \text{disp}(\text{Paths}(p_2, p_1)) \} \end{aligned}$$

For instance, consider Vp in our running example (Figures 3, 4). Then:

$$\mathbf{translate}(Vp, CWp, 20) = \{ 50, 30 \},$$

$$\mathbf{translate}(Gp, Vp, 20) = \{ -5, 0 \},$$

$\mathbf{translate}(Vp, Ep, \cdot)$ converts Ep to the time line of Vp , yielding $\{E'p\}$, where :

$$\begin{aligned} E'p = & [period : [-20, 50], anchored_processes : \{ [anchor : 30, processes : \{CWp\}], \\ & [anchor : 20, processes : \{Gp\}], [anchor : 25, processes : \{Gp\}] \}] \end{aligned}$$

Note that the periods of the components of Ep were not changed because they are specified relative to a time line nested within Ep .

- To find all machine processes used in assembly line $A1$, we write the query:

```

SELECT  m
FROM    Assembly_line_process a, Complex_process m
WHERE   a.line = A1 AND m IN components(a)

```

- To find all common sub-processes of robots $R2$ and $D2$, we write the query:

```

SELECT  sp
FROM    Robot_schedule r2, Robot_schedule d2
WHERE   r2.robot = R2 AND d2.robot = D2 AND sp IN components(r2)
        AND sp IN components(d2)

```

Note that this query does not inquire about shared components, since sharing exists only in the context of global processes.

An example of querying about shared components appears in 6.3.

6.2 Translation Between Time Lines

Because the timing information associated with processes is relative, evaluation of temporal queries demands conversion of timing information from one temporal frame of reference to another. Such conversion would be done along a time path from a process to its components. The translation of temporal information is computed with the **translate** operator that converts timing information from one time line to another. These two time lines should be related in the process composition graph. Namely, they should be associated with two processes where one is referenced from or contained in the other. Formally, we use **translate**, for processes p_1 and p_2 , as:

$$\mathbf{translate}(p_1, p_2, \left\{ \begin{array}{l} t \\ [t_1, t_2] \\ - \end{array} \right\})$$

where braces denote choice. The semantics of **translate** is that of a partial function defined as follows:

- If there is some path from p_1 to p_2 in the process composition graph, then it translates the temporal information in its third argument from p_2 's time line to p_1 's time line, i.e. going up in the composition graph. Corresponding

process value is *sp*. The type *SHARED-PROCESS* is defined as a subtype of the *PROCESS* type:

type *SHARED-PROCESS* = [*period* : *Time-interval*]

The sharing constraint: For *sp*, a shared process, and a global time point *t*, there exists at most a single global process *gp* with value [*anchor* : *t*, *process* : *sp*].

Note that because temporal information on processes is relative, it is enough to globally time for every real world activity, only the corresponding highest level complex process. Thus, the description above of the global timing of subprocesses would be used only for explicating their semantics.

6 Querying Processes

The direct representation of process semantics in the Rtime data model enables using standard object-oriented query languages to query process information. Temporal queries on processes may require temporal information expressed relative to a variety of time lines. Thus, they may query on component processes in terms of their own time line, or in terms of the time line of an enclosing complex process. To facilitate the expression of such queries in terms of standard query languages, we introduce a *translation* operator for translating temporal information from one temporal frame of reference to another. Temporal queries are expressed as standard queries including temporal operators.

For brevity of the discussion, we express queries in a notation borrowed from the *O₂Query* query language of the *O₂* system [BCD89, D⁺91]. The central query construct that we use is the **SELECT-FROM-WHERE** block made of three parts: the **SELECT** clause defines the output of a query which is a relation, the **FROM** clause introduces the range of the variables which is a collection of classes and relations, and the **WHERE** clause defines a conditional expression.

6.1 Queries on Process Composition

To express queries on process composition we use an operator, **components**, that when applied to a process *p*, as in **components(*p*)** yields its set of (direct or indirect) components. For an atomic process *p*, **components(*p*)** is empty. The operator is evaluated by computing the transitive closure of the *processes* attribute in the *anchored_processes* attribute of a *Complex-process* object or value.

We demonstrate these queries through examples:

occurrences can be grouped together into a relation of global processes that share the same process component with different global anchors.

5.1 Shared Processes

The need to account for sharing of process objects arises in the context of global processes, where we need to know which component processes occurring within a global process(es) are *repeated*, thereby corresponding to the distinct real world activities, and which are *shared*, thereby corresponding to the same real world activity.

In the example described in Figure 1, the Chain Wheel process CWp is *repeated* within the Ep and the Sp subprocesses. That is, every global occurrence of Vp , anchored at global time point t , implies two global occurrences of CWp , anchored at global time points $t + 30$ ($= t + 10 + 20$) and $t + 10$ ($= t + 3 + (-13) + 20 + 0$). In contrast, the Chain Construction process CCp is *shared* between the subprocesses $D2p$ and $R2p$, in every global occurrence of Vp . This implies that in every global occurrence of CWp , there is just a single occurrence of CCp . Hence, a global occurrence of Vp , anchored at global time point t , implies only two global occurrences of CCp , anchored at global time points $t + 30 + 10$ and $t + 10 + 10$. These occurrences result from the two global occurrences of CWp , as described above. (In Figure 4, dashed edges denote sharing.)

The distinction between *repetition* and *sharing* is further sharpened when considering simultaneous repetition of a subprocess, which arises in case where a subprocess has two different time paths, with the same displacement value. In that case, if the subprocess is not shared, every global occurrence of a parent process would imply a distinct simultaneous global occurrence of the subprocess; if the process is shared, then there are no simultaneous global occurrences.

A full account for the *sharing* phenomenon in the Rtime data model requires imposing global *sharing specifications* on a database instance. A sharing specification with respect to a subprocess p' of a process p , is a subset of the set of time paths $Paths(p, p')$, with a common displacement value. The sharing specification says that in every global occurrence of p , all global occurrences of p' obtained through paths in the sharing specification, are shared¹. The problem with this approach lies in the global character of sharing specifications. In this paper we introduce a more restricted account, where *sharing* is conceived as a local property of a process. Under the restricted view, a *shared process* sp has no simultaneous global occurrences. That is, for every global time point, the database has at most a single global process object whose

¹Sharing specifications in the context of structured plans, are fully investigated in [BS94].

To check the containment constraint, it is sufficient to check it with respect to direct components alone, since containment is transitive. That is, for every displacement t_2 in $disp(Paths(p', p''))$, and every anchor value t_1 such that $[t_1, p']$ is an element in the *anchored_processes* value of p :

$$p''.period + t_2 \subseteq p'.period \wedge p'.period + t_1 \subseteq p.period \Rightarrow p''.period + t_1 + t_2 \subseteq p.period$$

Note that if the type of process periods is generalized to *Temporal-Element*, the containment constraint stays intact. The only difference is that the inclusion relation is *Temporal-Element's* inclusion.

5 Global and Shared Processes

Timing information in the Rtime model is purely relative, and its processes are potential real world activities. A potential process turns into a real world one, once it is anchored to global time. Such a process is a *global process*. The type *GLOBAL-PROCESS* is defined as follows:

$$\mathbf{type\ } GLOBAL_PROCESS = [anchor : Global-Time-Point, process : Process]$$

The period of activity of a global process gb is the global time interval:

$$[gp.process.lb + gb.anchor, gp.process.ub + gb.anchor]$$

For example, suppose Vp occurred at Feb. 30th, 1900. This occurrence is the global process *Global_Vp*, with value $[anchor : 2.30.1900, process : Vp]$, and period of activity $[-50, 50] + 2.30.1900$.

The no global embedding constraint: Since global processes are real world activities, while embedded anchored processes are potential processes, we impose a constraint that global processes do not appear as subcomponents of other processes.

Subprocesses of a process anchored within a global process receive also global timing, thereby turning from potential to real world activities. Assume that a process p is anchored, at global time point t , in global process gp . A component p' of p with a time path σ , has a global period of activity $p'.period + disp(\sigma) + t$. Such a combination of global time and relative times enables a description of historical information on complex processes. The implied global occurrence of the component process Ep of Vp has the global time period $[-30, 40] + 10 + 2.30.1900$.

A process may occur as a component in many global processes. For example, Vp might have been used several times over the last month, yielding several global processes, all having Vp as their single anchored process. These different

rectly), the process composition hierarchy is *acyclic*. Hence, in any given hierarchy, time paths have finite length. A process value may be included several times within a complex process, possibly with different anchor values. Thus, in our example, CWp is included twice within Vp . Once, through the processes Vp, Ep , and once through the processes Vp, CUp, Cp, Sp .

The set of all time paths of p' relative to p is denoted $Paths(p, p')$. If p and p' are process objects, then $Paths(p, p')$ is defined similarly, using their values. The *displacement* of a time path σ , denoted $disp(\sigma)$, is the sum of time points in the path. It denotes the cumulative displacement between the origins of the time orders corresponding to the start and end of the path. The displacement of a set of time paths P , denoted $disp(P)$, is the set $\{ disp(\sigma) \mid \sigma \in P \}$. In what follows, we denote time paths by $\sigma_1, \sigma_2, \dots$, and sets of time paths by P_1, P_2, \dots .

In the example of Figure 4 we have that: $Paths(Vp, Ep) = \{ \langle 10 \rangle \}$, and $Paths(Vp, Gp) = \{ \langle 10, 10 \rangle, \langle 10, 15 \rangle \}$, with corresponding displacements: $disp(Paths(Vp, Ep)) = \{ 10 \}$, and $disp(Paths(Vp, Gp)) = \{ 20, 25 \}$.

The semantics of an anchored process p' that is a component of a complex process value p , with a time path σ , is that of a component process active at the period: $p'.period + disp(\sigma)$, with respect to the time line of p . Thus, in our running example, if the period of Ep is specified as $[-30, 40]$ (see Figure 3), its period with respect to the time order of Vp is $[-30 + 10, 40 + 10]$. The process Gp that has a period $[5, 7]$, is repeated twice within Vp , and occurs at two periods in the time line of Vp : $[5 + 20, 7 + 20]$, and $[5 + 25, 7 + 25]$.

Using the notion of a time path, we can similarly define the translation of a time point or interval specified at the time line of a process p' to the time line of an enclosing process p . For such a time point t , its possible values at the time order of p are $\{ t + t' \mid t' \in disp(Paths(p, p')) \}$; for an interval $[t_1, t_2]$ its possible values at that time order are $\{ [t_1, t_2] + t' \mid t' \in disp(Paths(p, p')) \}$. We define the translation of process intervals, time points, and time intervals in the reverse direction, from the time order of a parent complex process to one that is nested within it, using subtraction in place of addition in the above definitions.

4.5 The Containment Constraint

Since anchored processes are subprocesses of their containing process, we have to ensure that their periods relative to the containing process are within its period of activity. This constraint is called the *containment constraint*. Formally, for a process value p' that is a component of p , with a set of time paths $Paths(p, p')$, we require that:

$$\forall t \in disp(Paths(p, p')) : p'.period + t \subseteq p.period$$

4.4 Process Semantics and Time paths

The semantics of an atomic process ap is that of a potential process having no internal structure. A complex process cp is a potential process having anchored processes as its components. The period of activity of a process, $ap.period$, or $cp.period$, relates to the time line of the process itself. The period of activity of an anchored process nested as a component within cp , can be related to the time line of cp via the anchor value.

The semantics of processes is fully defined in [BM93], where processes and their temporal properties are laid out as a world of *structured histories*. According to that semantics, a complex process (a structured history) is a time function, that maps time points to finite multisets of processes. For example, the complex process value of the object Vp in Figure 4, is the time mapping:

$$\begin{aligned} 3 &\longrightarrow \{ Cup \}, \\ 10 &\longrightarrow \{ Ep \}, \\ &\textit{other time points are mapped to the empty multiset .} \end{aligned}$$

Since the processes' construction is defined inductively, process values cannot reference themselves (directly or indirectly). For a detailed study of structured histories we refer the reader to [BM93]. In this paper we restrict the semantics' presentation to a characterization of central concepts that derive from the full formal semantics, and underly the temporal queries suggested below (see section 6). The multiset semantics suggested in [BM93], is simulated, in the Rtime model, by repetitions of anchored processes.

A process in the Rtime model denotes a hierarchy of time lines. The main semantical operation involves translation of temporal information between levels of the process composition hierarchy. This is done using the notion of a *time path*, which is inductively defined as a sequence of anchor values from a complex process to a nested component.

A *time path* is any sequence of time points that can be constructed using a finite number of applications of the following two rules (where $\langle \rangle$ encloses sequences):

1. For an anchored process value $[t, p']$, that is a direct component of a complex process value p , $\langle t \rangle$ is a *time path* of p' relative to p .
2. For an anchored process value $[t, p']$, that is a direct component of a complex process value p , and a process value p'' with a time path σ relative to p' , $\langle t \rangle || \sigma$ is a time path of p'' relative to p (where $||$ denotes sequence concatenation).

Note that since the semantics of processes guarantees that process values cannot reference themselves (directly or indi-

distinction between solid and dashed edges.

Insert Figure 4 here.

The graph shows the vehicle assembly process Vp , with two component processes: CUp , for carriage unit assembly, and Ep , for engine assembly. The anchor of Ep in Vp is 10; hence $(10, Ep)$ is a direct component of Vp . Process Gp is anchored twice in Ep , at times 15 and 10, and process CWp is anchored twice at two different processes: Ep and Sp . Hence, $(20, CWp)$ is a direct component of Ep , and $(0, CWp)$ is a direct component of Sp . Note that although the process composition graph is on the instance level of processes, it describes potential processes, not global ones. Hence, a process object can be anchored at different times. Multiple anchor times are realized by multiple global processes, that have the same value for their *process* attribute.

Figure 3 describes an instance database that corresponds to this process composition graph. It is an instance of the database schema presented in Figure 2. We emphasize, again, that all time periods are relative, e.g., the period $[-30, 40]$ associated with Ep is relative to its own time line. *Negative anchors* are useful for modeling processes like scheduling activities, where the process has an expected natural beginning (e.g., a flight's takeoff, or a semester's start), which is preceded by some *preparation* subprocesses.

Real-time databases [Ram93] employ timed transactions (global processes) whose scheduling involves a representation of approximate times or intervals. These can be included in our model by using such times and intervals (instead of exact ones) for timing information on processes. Thus, processes with inexact intervals would be a subclass of the general *Process* class, provided that appropriate methods have been defined to test and manipulate inexact time points and intervals. For example, an inexact motion process can be described as follows:

```
class Inexact-Motion-Process inherit Process
    type [period : Approximate-Interval, from : Coordinate, to : Coordinate, object : Part]
end;
```

where an approximate interval is defined as:

```
type Approximate-Interval = [lb : [below, above : Time-Point], ub : [below, above : Time-Point ]
```

The lower bound t of such an interval, satisfies $lb.below \leq t \leq lb.above$ and similarly for its upper bound.

COMPLEX_PROCESS type augments the *PROCESS* type with an *anchored_processes* attribute whose values are sets of tuples of an anchor and a set of anchored processes. The anchored processes have, each, their own time line. For a complex process p , a process p' enclosed in its attribute *anchored_processes*, together with the associated anchor value is called a *direct component* of p . Note that the type *COMPLEX_PROCESS* is a subtype of the type *PROCESS*. We assume also a *Complex_process* class, whose type is the *COMPLEX_PROCESS* type just defined. Hence, the class *Complex_process* is a subclass of the class *Process*.

For example, consider a complex process p starting at 3 : 00 and ending at 14 : 00, that has two components specified to be active at periods [4 : 00, 6 : 00] and [7 : 00, 9 : 30]. If these processes were specified with respect to a time line different from that of p , whose origin lies at 3 : 00 with respect to p 's time line, the anchor value for these processes is 3 : 00. Accordingly, p might have the complex process value:

```
[ period :          [3 : 00, 14 : 00],
  anchored_processes : { [anchor : 3 : 00, processes : {[period : [4 : 00, 6 : 00]], [period : [7 : 00, 9 : 30]]}] }].
```

If the component processes of p were complex, then the nesting of *anchored_processes* would be deeper. For example, if the component process whose period is [4 : 00, 6 : 00] is complex, having component processes anchored at times -1 : 00, 0 : 00, and 1 : 00, then p might have the value:

```
[ period :          [3 : 00, 14 : 00],
  anchored_processes : { [anchor :    3 : 00,
                        processes : { [period :          [4 : 00, 6 : 00],
                                      anchored_processes : { [anchor : -1 : 00,
                                                                processes : { [period : [2 : 50, 3 : 00]] } ]
                                      [anchor : 0 : 00,
                                                                processes : { [period : [0 : 30, 5 : 00]] } ]
                                      [anchor : 1 : 00,
                                                                processes : { [period : [-3 : 30, 5 : 00]] } ] } ]
                        [period :          [7 : 00, 9 : 30]] } ] } ]].
```

Relative temporal composition can be visualized by a *process composition graph*, a directed acyclic graph whose nodes stand for process objects and whose edges point from a node of a process to the nodes of its direct components. Edges are labeled by the anchor of the component with respect to the containing process. The process composition graph for our running example is shown in Figure 4. It describes an instance of class *Complex_process*. For the moment ignore the

type *PROCESS*. Note that the 3 : 00 and 5 : 00 time points are relativized to the origin (zero) of the process's time line. Hence, the semantics of an atomic process *p* is that of a stand-alone process active at the period *p.period*. As noted above, we restrict ourselves to processes that are not interruptible. This restriction is realized by assuming the type *Time-Interval* for the *period* attribute of atomic processes. In what follows we denote processes by p_1, p_2, \dots .

Process values can carry additional information, beyond their temporal properties, such as the resources used by the process in a manufacturing application. For this purpose we use subtypes of *PROCESS* types and corresponding subclasses of the *Process* class in which non-temporal attributes are added. For instance, in Figure 1, *DHp* is an atomic process object. It can be defined as an instance of:

```

class Drill_hole inherit Process
    type [agent : Robot, tools : {Drill} ]
end;

```

4.3 Complex Processes

Complex process types describe processes such as the vehicle assembly process in our example, that have other processes as their components. A process component is specified along its own time line. To relate this time line to that of the complex process, we give the displacement between the origins of the two time lines. We term the displacement an *anchor*, and the associated process, which may have an arbitrarily complex process value, an *anchored process*. Thus, a complex process value forms a hierarchy of temporal coordinate systems formed by the recursive nesting of anchored processes within other processes. Each level of anchoring entails a relativization of the temporal coordinates of the contained anchored processes with respect to the containing process. Note that as component processes may have been constructed independently, and may have complex internal structures, timing these component processes, including their subprocesses, in the global time order, would be inefficient and could undermine the benefits of a relative time specification. Accordingly, we leave the internal temporal description of these processes intact, and just supply the temporal displacements between the origins of their respective time lines, and the origin of the containing process' time line.

Assume that there is a class *Process*, whose type is the *PROCESS* type defined above. The prototypical *COMPLEX_PROCESS* type is defined as follows:

```

type COMPLEX_PROCESS = [ period : Time-Interval,
    anchored_processes : {[anchor : Time-Point, processes : {PROCESS}}]

```

where instead of a *PROCESS* value in the *processes* attribute, we could have a *Process* object. The

is well defined. Note that time points may precede the origin. Thus, *Time-Point* defines a temporal frame of reference (coordinate system, time line) with origin 0. Different granularities of time units are defined as subtypes of *Time-Point*, e.g. years, months.

Corresponding to each Time-Point type *TP*, a *Time-Interval* type *TI* is defined, whose instances are time intervals described as pairs $[t_1, t_2]$ of time points of type *TP* such that $t_1 \leq t_2$. We assume the predefined operations *lb* and *ub* returning the lower and upper bounds of a time interval, respectively. Addition and subtraction of an interval $[t_1, t_2]$ and a time point t are defined as addition and subtraction of t from its bounds, respectively. The containment relationship between two intervals, denoted \subseteq , is defined as: $[lb, ub] \subseteq [lb', ub']$ iff $lb' \leq lb$ and $ub \leq ub'$. Familiar temporal predicates such as *overlap* or *during* on intervals [All83] are assumed to be available. In what follows we denote time points by t_1, t_2, \dots , and intervals by $[t_1, t'_1], [t_2, t'_2], \dots$.

Global-Time-Point is a subtype of the *Time-point* supertype, and *Global-Time-Interval* is its corresponding *Time-Interval* subtype. We assume that every database application using the Rtime data model singles out one *Time-Point* type as the *GTP* type, and its corresponding *Time-Interval* type as the *GTI* type. Processes whose time line is *GTP* are *Global-Processes* (see below).

Temporal elements were introduced in [Gad88], for modeling temporal information by a data structure that is closed under union, intersection, and complementation. For the purpose of the Rtime model, the Time-Interval type can be generalized into a *Temporal-Element* type *TE*, whose instances are finite unions of time intervals over *TP*. Hence, *TE* can account for non contiguous time periods. For example, a single temporal element can account for the period of time $[3 : 00, 5 : 00]$ and $[6 : 00, 8 : 00]$. In this paper we assume that the period of activity of a process is an interval, rather than a temporal element. This way we avoid the need to introduce the more complex operations associated with temporal elements. This assumption can be generalized in the future, to processes with non-contiguous periods of activities, using temporal elements instead of intervals.

4.2 Atomic Processes

An atomic process value is a period of activity. Hence we define the *PROCESS* type as follows:

$$\mathbf{type} \text{ PROCESS} = [\textit{period} : \textit{Time-Interval}]$$

The *start* and *end* times of an atomic process value are extracted with the *lb* and *ub* operations on its *period* attribute, respectively. For example, a process starting at 3 : 00 and ending at 5 : 00, has a value, $[\textit{period} : [3 : 00, 5 : 00]]$, of

from atomic process values and process objects, using the type constructors of the data model. Thus, complex process types are the analogue of the standard complex values. Process objects whose values are complex are called *complex process objects*. A constraint put on complex processes is that the period of activity of a complex process must contain the periods of its component processes, when relativized to the complex process' time line.

The temporal description of a process (atomic or complex) is inherently relative to its own time line. The period of activity of an atomic process does not refer to a particular occurrence of the process, since the process can be repeated indefinitely, but describes it in terms of its own time line. Similarly, a complex process is specified in terms of its own time line. The periods of the component subprocesses of a complex process are related to its time line by specifying the relative displacement between the origins of their time lines and that of the complex process.

A *Global process* is a process associated with a global time point. It describes a real world, possibly structured activity, whose structure is given by the structure of the process (if it is complex). The global time point specifies a historical occurrence time for the origin of the time line of the process. The rest of the process' component subprocesses are globally timed, using their relative displacements. Global processes are instances of class *Global_process*, which is introduced in Section 5.

Processes can be annotated as *shared* processes, with the meaning that simultaneous occurrences of their global processes are not repeated. That is, for a given shared process p , and a given global time point t , there can be at most a single *global process* object whose process component is p , and whose global time point is t . For example, in Figure 1, the Chain Construction process *CCp* is a *shared* process. Shared processes are instances of class *Shared_process*, which is introduced in Section 5.

To introduce the formal constructs for processes, we first describe the required notions for describing time points, intervals and elements. As our running example we use the vehicle assembly line shown in Figure 1.

4.1 Time Types

We regard time values as ADTs whose concrete representation is encapsulated from the rest of the data model [WD92], thereby allowing for a wide range of application requirements. Since different types of time are based on the common notion of being sets of ordered points, they are generalized into a supertype *Time-Point*. The *Time-Point* type defines an equality comparison operator, $=$, an ordering relationship, $<$, and arithmetic addition and subtraction functions, e.g. $16 : 00 + 5 \text{ hours}$. The constant 0 is assumed to denote a distinguished zero time point, that in each process stands for its origin (e.g., midnight). Given an origin and a time point, the meaning of time expressions such as $16 : 00 + 5 \text{ hours}$

An object or a value may refer to other objects via their identity. Similarly, an object or a value may be composed of subvalues. In the latter case, however, a subvalue is part of its container and cannot be shared by other objects. Sets of objects may be stored in named sets, called *relations*, consisting of values having the same type, or of objects that are instances of the same class. A relation may store objects from subclasses of its declared class. Relations serve as outputs of queries. An example of a relation is *robot* in Figure 2. A *database schema* includes the definitions of types, classes, and relations. Figure 2 is an example of a database schema. A database is an *instance* of its schema. It consists of: (a) the set of object identities belonging to each class, called its *extent*, and (b) a set of values (or objects) of the appropriate type for each relation. An example of a database instance is given in Figure 3.

In what follows, we denote values by v_1, v_2, \dots , objects by o_1, o_2, \dots , and types by T_1, T_2, \dots . Tuple values are denoted by $[A_1 : v_1, \dots, A_k : v_k]$, and set values by $\{v_1, \dots, v_m\}$.

Insert Figure 2 here.

Insert Figure 3 here.

4 Processes

The Rtime data model basic construct is the *process*. A process is an *object*, which is an instance of class *Process*. A process value is of type *PROCESS*. A process describes a potential real world activity, as a structure of potential real world activities, hierarchically organized along their own time lines. The hierarchical organization is based on repeated relativization of origins of time lines of processes to time lines of other processes. The semantics of such relativization is that of *shifting in time*. In Figure 1, for example, Vp is a (complex) process object, whose value includes relativization of the (complex) process objects Ep and CUp to the time line of Vp .

A process value consists of a *temporal element*, which specifies the time period (relative to the process' own time line) at which the process is active, and of possibly additional information such as the agent (e.g., robot) performing the process. Such values are called *atomic process values*. Process objects whose values are atomic are called *atomic process objects*. *Complex process values* describe a temporal composition (aggregation) hierarchy of processes. They are built

basic temporal representation mechanism is that of a mapping of a time point to some value which results in representing a process by its history.

The naturalness of the notion of a process is well appreciated in artificial intelligence research, especially qualitative physics [For84, Hay85], where it is evident that representing history by means of sequences of states is awkward and often inefficient. Temporal hierarchies based on different granularities of time units have been employed in temporal reasoning and knowledge maintenance systems [Dea89, KG77] in order to facilitate efficient reasoning. A general abstract framework and a representation language for temporal hierarchies is introduced in [BM89, BM93]. An application of that framework for the development of music is described in [BS92, BS93].

Available process models [MTM91, PC92] provide constructs for modeling temporal ordering between processes (e.g., which subprocess occurs before another). However, they do not provide constructs for describing the internal temporal composition of complex processes or for timing and using shared and repeated processes.

3 An Object-Oriented Data Model

The Rtime data model is based on the object-oriented modeling paradigm and accordingly includes the familiar notions of object-oriented data models [A⁺89, Bee89]. We briefly recall these concepts and refer the reader to the references for a detailed description.

The data model includes *values*, *objects*, *types*, and *classes*. A *value* has a *type*. A type is recursively definable from atomic types and abstract data types (ADTs) using type constructors such as set and tuple (Extending the data model to other type constructors such as array and list is direct and omitted for brevity of the exposition.). An example of a complex type is:

```
type Assembly_line_process = [ line : Assembly_line,  
                                anchored_processes : { [ anchor : Time-point, process : Assembly_line_process ] } ]
```

An object has an *identity*, a value (state) and a behavior, defined by its *attributes* and *methods*. An object belongs to (is an instance of) a class. A class is defined by its *type* and its *behavior*. Figure 2 provides examples of the classes *Process* and *Complex_process*. Classes are related by an inheritance (subclass – superclass) relationship. A class inherits its type, attributes and methods from its superclasses. In that case the class type is a subtype of its superclasses' types, and it has all attributes and methods of its superclasses. The type of the class may declare additional attributes and methods. For example, in Figure 2, *Complex_process* is a subclass of *Process*.

of object-oriented data models. Processes and temporal types are presented in Section 4. Global processes and sharing are introduced in Section 5. Section 6 describes support for querying processes in the Rtime model. We conclude in Section 7.

2 Related Work

Temporal data models regard time as a timestamp attached to values of objects such as tuples or attributes in order to describe their evolution over time [Sno90]. Temporal relational data models describe the history of a relation by recording attribute values over time, i.e., by associating values of attributes with the intervals at which they held [MS91, NA89, Sno87, TC90]. Temporal data models incorporating complex objects [Bla91, KS92, TG89] similarly record the history of a complex value having parts, by storing the histories of the individual parts of the object. The obtained composition (aggregation) hierarchy derives from the structure of objects alone, and is not affected by temporal information. Temporal object-oriented models [EW90, RS91, SC91, Tau91, TLW91, WD92] describe the evolution of objects by recording their lifetime, and the history of values they had for each of their parts. Different granularities of time units, e.g., month, day, second, are, indeed, supported by some temporal models ([LJ88, MPB92]). However, describing activities such as the breakdown of a process into its components requires flexible granularities of time, that are not captured by these models.

Temporal data models can be characterized as describing histories as sequences of snapshots of the state of an (possibly complex) object. However, processes are continuous complex activities, whose representation as sequences of snapshots is conceptually cumbersome, and inefficient to store and manipulate. Since there are no natural points at which to record such snapshots of complex processes, the common approach implies storing snapshots of all possible combinations of the subprocesses executing in parallel. Moreover, these snapshots are necessarily global, making it difficult to manipulate complex processes and understand their subprocesses in isolation. Using independent histories for each subprocess of a complex process encodes process information by repeating and sharing corresponding histories. Such a representation fragmentizes the process description, losing the direct description of process composition, and the notion of a process as a stand alone entity which is more than a sequence of snapshots.

While some temporal models such as that of Wu and Dayal [WD92] which include ADTs can represent any composite structure, and in particular that of processes, the semantics of the representation would be encapsulated in the ADT representation and would not be visible to the user or more importantly to the database system and its query processor. Note that their model cannot represent process hierarchies, even though it regards time points as ADTs, because the

its hierarchical composition from subprocesses. The hierarchy of time lines forms a directed acyclic graph (DAG) of time lines of subprocesses; the root of the DAG corresponds to the complex process itself. To relate these different time lines, we specify with component processes the temporal displacement between the origins (zero points) of the time lines in terms of which these components were described, and the origin (zero point) of the complex process. The displacement is called an *anchor*, and the component process associated with it an *anchored process*.

Process objects have a value (state) which is a process and may be referred to through their identity, representing their occurrence within (possibly several) other processes. References to process objects are distinguished semantically into two kinds: (a) repeated occurrence of the same process value, which corresponds in the real world to distinct occurring activities having the same internal structure, and (b) shared subprocesses, as in the example of the shared Chain Construction process of $R2p$ and $D2p$, in Figure 1. In the latter case, the shared reference to the process identity corresponds to a single real world occurrence of the process. Such a semantic distinction is not available in standard object-oriented data models.

The benefits of the relative time specification employed in the Rtime model are evident when one considers repeated processes. Specifying these with absolute timing information would demand an explicit recalculation of the timing information every time a process serves as a component of another process. Besides avoiding redundant computation and storage of all the different occurrences of the same process, a single description of the internal composition of these processes leads to a greater degree of information encapsulation, increasing consistency, and facilitating its efficient maintenance. The relative time approach is especially useful when considering, for example, a library of manufacturing plans, even before they are realized into actual manufacturing processes. The benefits of relative coordinates are well known in representations of complex spatial information [Dav90, FD82]. However, we are not aware of a corresponding treatment of composite temporal information within temporal data models.

Using a virtual global process, we allow for timing occurrences of complex processes in a global frame of reference, providing for representation and manipulation of their real world timing. The global timing turns a potential process into an actual world one.

The explicit representation of process values and objects in the Rtime data model permits the usage of standard query languages to retrieve information on process composition. We provide an operator for converting temporal information between different temporal frame of references, and use it to support efficient evaluation of temporal queries within such languages.

The rest of the paper is organized as follows: Related work is discussed in Section 2. Section 3 recalls some definitions

1 Introduction

Advanced database applications, such as automated manufacturing, scheduling, and computer-aided software engineering [CKO92, MMK93], demand an explicit representation of processes, including their decomposition into subprocesses, where subprocesses may be naturally repeated or shared. For example, a complex assembly process in an automated factory is hierarchically composed of subprocesses, which may be repeated in case several machines perform the same function. As an example, consider an assembly process in an automated factory as visualized in Figure 1. In the figure, boxes denote processes, and containment of processes within other processes is indicated by either enclosure, or through arrows from a process to a contained process. A box overlapping several other boxes denotes a process that is shared by other processes. The process Vp performed at some assembly line, includes processes Ep and CUp performed by subordinate machines. Processes Ep and CUp both include a Chain Wheel construction process CWp , that has the same internal structure (i.e. it is the same manufacturing process). Thus, CWp is *repeated* within processes Ep and CUp . Subprocesses can also be *shared*, if they are performed jointly by several agents. In Figure 1, process Ep includes the schedules $R2p$ and $D2p$ of two robots, say $R2$ and $D2$, respectively, that jointly perform a Chain Construction process, CCp . Temporal information on these processes is inherently hierarchical due to their composite structure, and is most naturally specified using relative timing of a subprocess with respect to a containing process.

Insert Figure 1 here.

We suggest the Relative Time (Rtime) object-oriented data model for processes, in which processes are first-class values and objects, standing for potential real world activities. The processes in our model, extract the (possibly complex) structures of real world activities, and turn them into independent entities, that can be relativized with respect to different temporal frames of reference. Such a relative description allows for the incremental construction of complex processes from reusable libraries of components, e.g. production plan libraries, without the need for modifying the internal temporal description of these processes with each reuse.

Processes are represented by their period of activity. Process types may include additional information such as the agent (e.g., machine) performing the process. Complex processes are built using type constructors (e.g., set, tuple) from other processes. To provide for modular construction of complex processes, we describe each process in a temporal frame of reference (time line) natural to it, and make a complex process into a nested hierarchy of time lines, corresponding to

A Data Model for Processes Based on Relative Time

Mira Balaban Yoram Kornatzky

Dept. of Mathematics and Computer Science

Ben-Gurion University of the Negev

P.O.B. 653, Beer-Sheva 84105, Israel

Abstract

Advanced database applications such as automated manufacturing, scheduling, and computer-aided software engineering, demand an explicit representation of processes, including their decomposition into subprocesses, where subprocesses may be repeated or shared. Temporal information on these processes is inherently relative to particular temporal frames of reference, that may be different from that of a complex process containing them. We suggest the Rtime object-oriented data model in which processes are first-class citizens and complex processes are built, using standard type constructors, from their component processes. The relative timing of component processes is a key feature of the suggested model. It allows for a modular construction of complex process objects that may be repeated and shared. Standard object-oriented query languages can be used for temporal queries on processes, by providing an operator for translating timing information between different temporal frames of reference.

keywords: temporal database, relative time, process, object-oriented database