Analysis of Design Process Dynamics*

Tibor Bosse[1], Catholijn Jonker[1] and Jan Treur[1,2]

Abstract. To enable the development of automated support for design, a challenge is to model and analyse dynamics of design processes in a formal manner. This paper contributes a declarative, logical approach for specification of dynamic properties of design processes, supported by a formal temporal language. [1]This language is used to specify dynamic properties of a design process as a whole, or of parts thereof. At the most detailed level, in an executable sublanguage also simulation models are specified in a declarative, logical manner, which allows to use these specifications in logical analysis as well. The approach is illustrated by an example component-based agent-system design process.

1 INTRODUCTION

Providing automated support to manage the dynamics of a design process is in most cases not trivial. For example, in [6] some of the requirements put forward are that (1) a complete design process representation is needed, (2) with sufficient detail to allow for direct execution. Also by [1], [5] it is put forward that supporting the management of the dynamics of a design process is an important challenge to be addressed. This indeed is the aim of the current paper. The type of design considered is the design of component-based (e.g., software) systems for dynamic applications. In such application areas often components can be (re)used for which the properties are known. By composing a number of such components in a component-based design, the required overall dynamics is obtained. As holds for many design processes, designing component-based systems can be a rather complex and dynamic process, for which a number of tasks play a role, for example in this specific case:

* In: Proc. of the 16th European Conference on Artificial Intelligence, ECAI'04. IOS Press, 2004.

1 Vrije Universiteit Amsterdam, Department of Artificial Intelligence, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. E-mail: {tbosse, jonker, treur}@cs.vu.nl. URL: http://www.cs.vu.nl/~{tbosse, jonker, treur}

2 Universiteit Utrecht, Department of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

1. maintaining of specifications of properties of (reusable) components
2. maintaining of requirements on the overall system to be designed (usually in close contact with a stakeholder)
3. refinement and revision of requirements
4. determination of reusable components based on their properties, to find a system that satisfies the requirements
5. checking whether a system (a design object description) satisfies the requirements
6. revision of a design object description that does not satisfy the requirements

Most of these tasks essentially involve the dynamics of design as a process. The analysis of this *design process dynamics* is the subject of the current paper.

During a design process, two important concepts play a role: a design problem statement and a solution specification. A *design problem statement* consists of:

- a set of requirements in the form of dynamic properties on the overall system behaviour that have to be fulfilled
- a partial description of (prescribed) system architecture that has to be incorporated
- a partial description of (prescribed) dynamic properties of elements of the system that have to be incorporated; e.g., for components, for transfers, for parts, for interactions between parts.

A *solution specification* for a design problem is a specification of a design object (both structure and behaviour) that fulfils the imposed requirements on overall behaviour, and includes the given (prescribed) descriptions of structure and behaviour. Here 'fulfilling' the overall behaviour requirements means that they are implied by the dynamic properties for components, transfers and interactions between parts within the specification.

In this paper, in Section 2 a formalisation of design process dynamics will be discussed in terms of design states, design transitions and design traces. Section 3 addresses some dynamic properties of design processes. Section 4 gives an overview of an example design process. In Section 5, a relevant requirement will be given for the example system to be designed. It will be shown how this global requirement for the overall system can be refined to local requirements for parts of the system. Section 6 will

describe a simulation model of the example design process, and shows an example simulation trace. In Section 7 the example design process is analysed in terms of dynamic properties. Finally, Section 8 is a conclusion.

2 DESIGN PROCESS DYNAMICS

To analyse dynamics of a design process, a formalisation is needed of such dynamics. Such a formalisation is introduced in this section, inspired by [4]. It is based on the notion of design process state and design trace.

The state of a design process at a certain time point can be described as a combined state consisting of two states, S = <S1, S2> with:

- S1 *requirements manipulation state* (RM-state),
  including the current requirements set
- S2 *design object description manipulation state* (DM-state),
  including the current design object description state

A particular design process shows a sequence of transitions from one state S to another (next) state S'. Design *traces* are time-indexed sequences of design states, where each subsequent pair of states is a design transition. To describe such sequences a fixed *time frame* T is assumed which is linearly ordered. A *trace* γ over a state ontology Ont (including ontology for design objects and requirements) and time frame T is a mapping $\gamma : \mathsf{T} \rightarrow \mathsf{STATES(Ont)}$, i.e., a sequence of states γ_t ($t \in \mathsf{T}$) in $\mathsf{STATES(Ont)}$. The set of all traces over state ontology Ont is denoted by $\mathsf{TRACES(Ont)}$. Depending on the application, the time frame T may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form, as long as it has a linear ordering.

3 DYNAMIC PROPERTIES OF DESIGN

To formally specify dynamic properties that express characteristics of dynamic processes (such as design) from a temporal perspective an expressive language is needed. To this end the *Temporal Trace Language* TTL is used as a tool; cf. [7], which is briefly defined as follows. The set of *dynamic properties* $\mathsf{DYNPROP(Ont)}$ is the set of temporal statements that can be formulated with respect to traces based

on the state ontology Ont in the following manner. Given a trace γ over state ontology Ont, a certain state during a design process at time point t is denoted by state(γ, t), which as a TTL-expression refers to γ_t. These states can be related to state properties via the formally defined satisfaction relation |=, comparable to the Holds-predicate in the Situation Calculus: state(γ, t) |= p denotes that state property p holds in trace γ at time t. Based on these statements, dynamic properties can be formulated in a formal manner in a sorted first-order predicate logic with sorts T for time points, Traces for traces and F for state formulae, using quantifiers over time and the usual first-order logical connectives such as $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$.

To be able to perform some automated experiments with design processes, a simpler language has been used. This language (the *leads to* language) enables to model direct temporal dependencies between two state properties in successive states, as occur in specifications of a simulation model (for example, if in the current state, state property p holds, then in the next state, state property q holds). This language enables the automatic generation of simulated traces. The executable format is defined as follows. Let α and β be state properties of the form 'conjunction of ground atoms or negations of ground atoms. In the leads to language the notation $\alpha \rightarrow\!\!\!\rightarrow_{e, f, g, h} \beta$, means:

If state property α holds for a certain time interval with duration g, then after some delay (between e and f) state property β will hold for a certain time interval of length h.

For a formal definition of the leads to language in terms (as a sublanguage) of the language TTL, see [8].

Two different types of dynamic properties can be distinguished: Local Properties and Global Properties. Local properties only concern the smallest steps (taken into account in the conceptualisation of the process) in the process under analysis; for example:

At every point in time,
if a requirement **r** is imposed on the object to be designed,
and this requirement can be refined to sub-requirement **q**
then at the next point in time, sub-requirement **q** will be imposed
on the object to be designed

In contrast, a global property is a property that concerns the overall process (taken into account) in the process under analysis, for example:

> Eventually there is a committed requirement set **R** and
> a design object description **D** such that, for each requirement **r** in **R**,
> the design object description **D** satisfies requirement **r**

More complex Local and Global dynamic properties for design processes will be introduced in Sections 6 and 7, respectively.

4 AN EXAMPLE DESIGN PROCESS

To address in more detail the analysis of design process dynamics, an example design process was taken. The analysis approach is described and evaluated for this example design process. The example design process concerns the design of a cooperative information gathering agent system (see Section 4.2). The design approach is by requirements refinement (see Section 4.1).

4.1 Design by Requirements Refinement

A design process of a complex system (e.g., a software system) usually starts by specifying requirements for the overall system behaviour. They express the dynamic properties that should 'emerge' if appropriate components are designed and combined in a proper manner. Given these requirements on overall system behaviour, the system is designed in such a manner that the requirements are fulfilled.

Between dynamic properties at different levels of aggregation within a complex system (to be) designed, certain interlevel relationships can be identified; overall behaviour of the design object can be related to dynamic properties of parts of the design object and properties of interaction between these parts via the following pattern:

> dynamic properties for the parts &
> dynamic properties for interaction between parts
> ⇒ dynamic properties for the design object.

The process to identify new, refined requirements for behaviour of parts of the system and their interaction is called *requirements refinement*. Subsequently, the required dynamic properties of parts can be refined to dynamic properties of certain components and transfers, making use of:

dynamic properties for components &
dynamic properties for transfer between components
⇒ dynamic properties for a part.

4.2 An Example Design Problem

As a case study, the process of designing a multi-agent system for cooperative information gathering [7] will be analysed in more detail. To get the idea, assume the system to be designed has to consist of three agents: A, B and C. The resulting behaviour of the system must be as follows: agent A and B are able to do some investigations and make up a report on some topic, and communicate that to the third agent C. Both A and B have access to useful sources of information, but this differs for the two agents. By co-operation they can benefit from the exchange of information that is only accessible to the other agent. If both types of information are combined, conclusions can be drawn that would not have been achievable for each of the agents separately. To make the example more precise: the example agent model is composed of three components: two information gathering agents A and B, agent C, and environment component E representing the external world, see Figure 1.

Each of the agents is able to acquire partial information from an external source by initiated observations. For reasons of presentation, this by itself quite common situation for co-operative information agents is materialised in the following more concrete form. The world situation consists of an object that has to be classified. One agent can observe only the bottom view of the object, the other agent only the side view. By exchanging and combining observation information they are able to classify the object. For example, if an agent knows that the views are a circle and a square, it is concluded that the object is a cylinder.

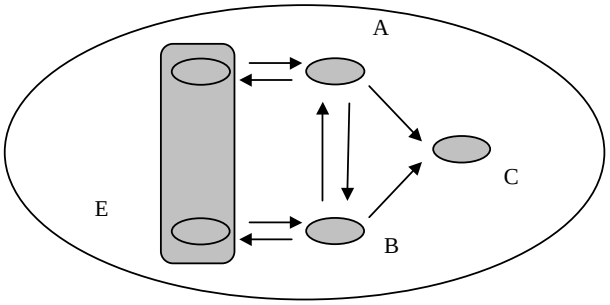

Figure 1. The example Agent System

In most multi-agent systems it is common that each agent has its own characteristics or attitudes. In the current system to be designed, the agents used as components in the design can differ, for instance, in their attitudes towards observation and communication: an agent may or may not be *pro-active*, in the sense that it takes the initiative with respect to one or more of:

- performing observations
- communicate its own observation results to the other agent
- ask the other agent for its observation results
- draw conclusions about the classification of the object

Moreover, an agent may be *reactive* to the other agent in the sense that it responds to a request for observation information:

- by communicating its observation result as soon as they are available
- by starting to observe for the other agent

The successfulness of the system to be designed will depend on the combination of attitudes of the agents. For example, if both agents are pro-active and reactive in all respects, then they can easily come to a conclusion. However, it is also possible that one of the agents is only reactive, and still the other agent comes to a conclusion. So, successfulness can be achieved in many ways and depends on subtle interactions between pro-activeness and reactiveness attitudes of both agents.

5 REQUIREMENTS OF THE EXAMPLE

In this section the example agent system to be designed as discussed in the previous section is further elaborated in terms of relevant requirements. Therefore, it is necessary to define the design problem statement, consisting of the requirements on the overall agent system behaviour. To simplify the example, it is assumed that just one main requirement is imposed on the current agent system, namely is whether or not a result will be generated. This requirement is called DODGP (Design Object Description Global Property):

DODGP Successfulness

For any trace of the system, there exists a point in time such that in this trace at that point in time agent C will receive a correct conclusion, either from A or from B (or from both).

In virtue of which combination of dynamic properties of the agents can success be achieved? In other words, which dynamic properties for the agents imply the property successfulness? Such a requirements refinement process can be managed more effectively if the overall requirements are not directly related to agent behaviour requirements, but one or more intermediate levels are created, as explained in Section 4.1. The idea is that for the agent system to be successful it is needed that

- both information sources within the environment E are addressed,
- if they are addressed, they provide the relevant information, and
- if the relevant information is provided by the information sources, a conclusion is drawn.

This first requirements refinement provides the dynamic properties DODGP1, DODGP2, DODGP3:

DODGP1 Information request effectiveness

At some points in time A and B will start information acquisition to E.

DODGP2 Information source effectiveness

If at some points in time A and B start information acquisition to E,
then E will generate all the correct relevant information for both.

DODGP3 Concluding effectiveness

If at some points in time E generates all the correct relevant information, then C will receive a correct conclusion.

These properties are logically related to DODGP (see also Table 1) by the implication:

$$\text{DODGP1} \;\&\; \text{DODGP2} \;\&\; \text{DODGP3} \Rightarrow \text{DODGP}$$

A next step in the requirements refinement process is to relate each of the dynamic properties DODGP1, DODGP2 and DODGP3 to agent behaviour properties. The complete refinement of these properties is elaborated in [2]. Due to space limitations, in this paper we only present a table with logical relationships between dynamic properties, without showing the exact definitions of all of the properties.

Table 1. Overview of all possible requirement refinements

DODGP1 ∧ DODGP2 ∧ DODGP3	⇒	DODGP
B1 ∨ B2 ∨ B3	⇒	DODGP1
DODI1(A) ∧ DODI1(B)	⇒	DODGP2
DODI2(X,Y) ∧ DODI3(X,Y,C)	⇒	DODGP3
DODBP1(A) ∧ DODBP1(B)	⇒	B1
DODBP1(X) ∧ DODBP2(X) ∧ DODBP4(Y) ∧ DODTP(X,Y)	⇒	B2
DODBP2(A) ∧ DODBP4(A) ∧ DODBP2(B) ∧ DODBP4(B) ∧ DODTP(A,B) ∧ DODTP(B,A)	⇒	B3
DODEP(A) ∧ DODTP(A,E)	⇒	DODI1(A)
DODEP(B) ∧ DODTP(B,E)	⇒	DODI1(B)
B4 ∨ B5	⇒	DODI2(X,Y)
DODBP3(X) ∧ DODTP(Y,X) ∧ DODTP(E,X) ∧ DODTP(X,C)	⇒	DODI3(X,Y,C)
DODBP6(Y) ∧ DODTP(E,Y)	⇒	B4
DODBP2(X) ∧ DODBP5(Y) ∧ DODTP(X,Y) ∧ DODTP(E,Y)	⇒	B5

In Table 1, in the form of logical implications an overview is shown of all possible refinements as discussed. Here X and Y are variables over the set {agent A, agent B}, where $X \neq Y$. Note that the different alternatives (*branches*) are indicated by the names B1 to B5. Moreover, to be able to distinguish the properties concerning the system to be designed (presented in this section) from the properties concerning the

design process itself (presented in Section 7), the names of the former have been slightly modified with respect to [2]. To be specific, to the name of each property, the prefix “DOD” has been added.

6 A SIMULATION MODEL

Making use of the formal approach described in Section 3, the dynamics of the example design process have been simulated by means of *local properties*. Two types of local properties are distinguished: those that model the dynamics of requirements states, and those that model the dynamics of the Design Object Description states. Due to space limitations, only a subset of the Local Properties used for the simulation are shown.

The process concerning *requirements* takes into account whether or not the stakeholder asserts that certain requirements are undesirable.

LP4 Requirement Refinement

Local property LP4 expresses that, if currently a requirement p exists that can be refined to a subrequirement q, and it has not been refined yet, then this should be done by refining via the best branch b (e.g. the one with the lowest costs). Formalisation:

is_a_current_requirement(p) ∧ is_a_subrequirement_of_via(q,p,b) ∧
not(requirement_refined(p)) ∧ best_branch_for(b,p) ∧ not(undesirable_branch(b))
→ is_a_current_requirement(q) ∧ requirement_refined(p) ∧ requirement_refined_via(p,b)

The process concerning *Design Object Descriptions* determines design object descriptions for sets of requirements given as input. Within this process it is taken into account whether or not the stakeholder asserts that certain components are undesirable as part of a design object.

LP6 DOD Generation

This property expresses that each local requirement l should be satisfied by adding the best component c for that requirement to the current DOD dod(x). Formalisation:

‘DOD_counter’(x) ∧ is_a_current_requirement(l) ∧ best_component_for(c,l) ∧
not(undesirable_component(c)) ↠ current_DOD(dod(x)) ∧ part_of_DOD(c,dod(x))

LP8 Local Requirement Satisfaction Determination

This property determines when a local requirement l is satisfied by a DOD. This is the case when the current DOD contains a component c for which this requirement holds. Formalisation:

current_DOD(d) ∧ part_of_DOD(c,d) ∧ holds_for(l,c) ∧ is_a_current_requirement(l)
→ local_requirement_satisfied(l)

Using the simulation model, a number of experiments have been performed. In such experiments, different types of revision might be needed with an increasing impact on the design process:

- revision of the *design object description* for given requirements based on the stakeholders judgement that a component used in the DOD is undesirable.
- revision of the refined *requirements* based on the stakeholder's judgement that one of these requirements is undesirable.
- revision of a whole *branch* based on the calculation that the costs of the design object description found are higher than expected.

An example trace of a design process in which the last type of revision is needed is depicted in Figure 2. Time is on the horizontal axis, the derived state properties are on the vertical axis. In this simulation, for all local properties the values (0,0,1,1) have been chosen for the timing parameters e, f, g, and h. Due to space limitations, only a subset of the derived atoms is shown, but the overall dynamics of the process are clear:

When the process starts, first the initial requirement dodgp is identified. After this, this requirement is refined into sub-requirements dodgp1, dodgp2 and dodgp3 (based on the logical relationships of Table 1). This process continues until the most elementary requirements (i.e. those that have no subrequirements) have been reached. Then a new design object description (called dod(1)) is created which consists of a number of components that satisfy all local requirements. Based on the costs of these components, the system calculates the total costs for each branch (i.e., for each collection of subrequirements, see Table 1). In case this number is higher than the predicted costs for that branch, the branch is marked as undesirable. This turns out to be the case at, for example, time point 17. Here, the refinement of requirement dodi2(x,y) via branch b4 turns out to be too expensive. As a result, the system starts

backtracking in the table of logical relationships in order to select another branch. In total, three branches are revised in this trace (namely b4, b1 and b2, respectively). Finally, the system succeeds in finding a satisfactory DOD. This resulting DOD is then evaluated and its total costs are calculated.

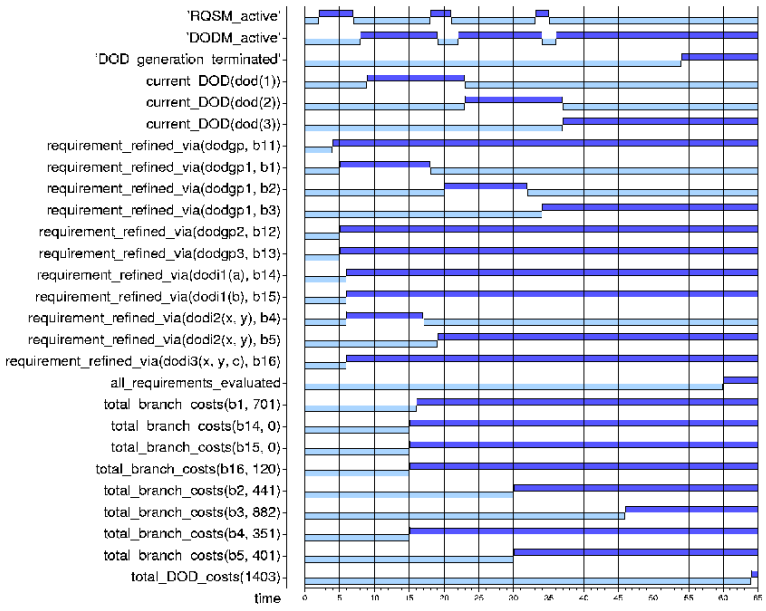

Figure 2. Simulation trace

7 GLOBAL DYNAMIC PROPERTIES

For design processes like the one described above, a number of global dynamic properties can be identified. For example:

- During (or after termination of) the design process, the design process objectives are fulfilled
- After termination of the design process the final design object description satisfies the requirements of the final RM-state
- After termination of the design process the requirements in the final RM-state have been declared sufficient by the stakeholder at some point during the process
- If one of the design process objectives is that the design process should be fast and cheap, then any design object description generated during the process solely consists of standard components

In this section a number of such dynamic properties, expressed as TTL statements, are presented. These properties as listed are relevant to be considered and checked for a design reasoning trace. They need not be satisfied by all design reasoning traces; they may be used to distinguish between different types of design reasoning traces as well.

GP1 Local Requirement Satisfaction

Eventually there is a DOD that contains a satisfactory component for each local requirement that exists at that moment. Formalisation:

$\exists t\ \exists d{:}DOD \quad state(\gamma,t,DM\text{-}state) \mid\!\!=\!\!= current\_DOD(d) \wedge$
$\forall r{:}localreq\ [state(\gamma,t,RM\text{-}state) \mid\!\!=\!\!= is\_a\_current\_requirement(r) \Rightarrow$
$\exists\ c{:}component\ state(\gamma,t,DM\text{-}state) \mid\!\!=\!\!= part\_of\_DOD(c, d) \wedge$
$state(\gamma,t,DM\text{-}state) \mid\!\!=\!\!= holds\_for(r, c)]$

GP2 DM Successfulness

For each local requirement, if there is a component that satisfies it, then such a component will be added to the DOD. Formalisation:

$\forall t\ \forall r\ \forall{:}localreq\ \forall c{:}component$
$state(\gamma,t,DM\text{-}state) \mid\!\!=\!\!= is\_a\_current\_requirement(r) \wedge$
$state(\gamma,t,DM\text{-}state) \mid\!\!=\!\!= holds\_for(r, c) \Rightarrow$
$\exists t' \geq t\ \exists d{:}DOD\ \exists c'{:}component\ state(\gamma,t',DM\text{-}state) \mid\!\!=\!\!= part\_of\_DOD(c', d) \wedge$
$state(\gamma,t',DM\text{-}state) \mid\!\!=\!\!= holds\_for(r, c')$

GP3 RM Successfulness

At a certain point in time, all nonlocal requirements will be refined. Formalisation:

$\exists t\ \forall n{:}nonlocalreq \quad state(\gamma,t,RM\text{-}state) \mid\!\!=\!\!= is\_a\_current\_requirement(n) \Rightarrow$
$state(\gamma,t,RM\text{-}state) \mid\!\!=\!\!= requirement\_refined(n)$

The global properties presented above have been checked automatically against the simulation trace discussed in Section 6. They all turned out to hold, which confirms the fact that the simulated design processes satisfied the desired properties such as termination and successfulness.

In addition to the above, logical relationships can be and have been identified between dynamic properties at different abstraction levels. Such relationships relate the Global Properties presented in this section to some of the Local Properties presented in Section 6. They can be specified by means of logical implications or

graphically by means of AND/OR trees. In these relationships, also properties at an intermediate level of aggregation (*Intermediate Properties*) occur, addressing smaller steps than Global Properties do, but bigger steps than Local Properties do. In combination with the automated checks described above, the interlevel relationships can play an important role in the analysis of design processes, because of their hierarchical structure. I.e., if a certain Global Property turns out not to hold for a given design process trace, then the table of logical relationships can be consulted in order to pinpoint which local properties are candidates for causing the failure.

8 CONCLUSION

In order to develop automated support for the dynamics of nontrivial design processes, the challenge of modelling and analysing such dynamics in a formal manner has to be addressed; cf. [1], [5], [6]. This paper offers an approach to do so. The complex dynamics of a design process has been analysed in such a precise way that properties of the process as a whole can be specified and, moreover, part of the analysis contains enough detail to allow for simulation. The result of simulation has been checked against the properties of the design process as a whole.

Compared to the references mentioned above, the approach put forward is a declarative, logical approach supported by a formal language TTL for specification of dynamic properties of design processes, which has a high expressivity. Furthermore, also simulation models are specified in a declarative, logical manner, which allows using these specifications in logical analysis as well.

The paper shows the potential of this formal analysis as a technique for analysis at a high level of abstraction, and for constructing simulations at an abstract level to experiment with dynamics of a design process. The simulation actually is entailed by the analysis and requires no additional programming.

The analysis approach that is for the first time applied to design processes here, has previously been applied to complex and dynamic reasoning processes other than design, such as reasoning based on multiple representations [3]. In these cases in addition to simulated traces, also empirical (human) reasoning traces have been formally analysed. For further research it is planned to formally analyse protocols of human design processes in a similar manner.

REFERENCES

[1] Baldwin and Chung (1995). A Formal Approach to Managing Design Processes. IEEE Computer, Feb. 1995, pp. 54-63.

[2] Bosse, T., Jonker, C.M., and Treur, J., (2003). Requirements Analysis in Design of Agent Systems. Vrije Universiteit Amsterdam, Department of Artificial Intelligence. Technical Report. URL: http://www.cs.vu.nl/~wai/Papers/BDBCh3.pdf

[3] Bosse, T., Jonker, C.M., and Treur, J., (2003). Simulation and analysis of controlled multi-representational reasoning processes. *Proc. of the Fifth International Conference on Cognitive Modelling, ICCM'03*. Universitats-Verlag Bamberg, 2003, pp. 27-32.

[4] Brazier F.M.T., Langen P.H.G. van, Treur J., (1996). A logical theory of design. In: J.S. Gero (ed.), *Advances in Formal Design Methods for CAD, Proc. of the Second International Workshop on Formal Methods in Design*. Chapman & Hall, New York, 1996, pp. 243-266.

[5] Brown, D. C., and Chandrasekaran, B., (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*, Pitman, London.

[6] Corkill, D.D. (2000). When Workflow Doesn't Work: Issues in managing dynamic processes, *Proceedings of the Design Project Support using Process Models Workshop*, Sixth International Conference on Artificial Intelligence in Design, Worcester, Massachusetts, June 2000, pp. 1-13.

[7] Jonker, C.M., and Treur, J. (2002). Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. *International Journal of Cooperative Information Systems*, vol. 11, 2002, pp. 51-92.

[8] Jonker, C.M., Treur, J., and Wijngaards, W.C.A. (2003). A Temporal Modelling Environment for Internally Grounded Beliefs, Desires and Intentions. *Cognitive Systems Research Journal*, vol. 4, 2003, pp. 191-210.