A Modeling Environment for Reified Temporal-Causal Networks: Modeling Plasticity and Metaplasticity in Cognitive Agent Models

Jan Treur(✉)

Social AI Group, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
j.treur@vu.nl
https://www.researchgate.net/profile/Jan_Treur

Abstract. Plasticity is a crucial adaptive characteristic of the brain. Relatively recently mechanisms have been found showing that plasticity itself is controlled by what is called metaplasticity. In this paper a modeling environment is introduced to develop and simulate reified temporal-causal network models that can be applied for cognitive agent models. It is shown how this environment is a useful tool to model plasticity combined with metaplasticity.

1 Introduction

Real-world cognitive agents are often adaptive, described by adaptation principles. For example, mental or neural networks equipped with a Hebbian learning mechanism [5] are able to adapt connection weights over time and learn in this way. This is usually called *plasticity* (modeled by the middle layer in the example in Fig. 1). In some circumstances it is better to learn fast, but in other circumstances it is better to stay stable and persist what has been learnt in the past. To control this, a type of (higher-order) adaptation called *metaplasticity* is used (highest layer in Fig. 1); e.g., [1, 6].

In [8, 11] any form of adaptation had to be added by specific procedural program code like usually is done for adaptive networks; there was no standard or principled way to explicitly specify adaptive causal relations. To offer a more principled way to specify adaptive networks, recently the notion of *network reification* was proposed as an addition to the temporal-causal network modeling approach, and illustrated by some case studies that were implemented in a more or less ad hoc - proof of concept – manner [9, 10]. These initial explorations suggest that this notion of network reification could be useful to model in a systematic and transparent manner from a network-oriented perspective, cognitive and social agent processes that are adaptive of any order, and in particular those involving plasticity and metaplasticity (e.g., see Fig. 1).

Following this, the current paper introduces a specification format based on declarative mathematical relations and a modeling environment for reified temporal-causal networks, implemented in Matlab in a principled and structure-preserving manner. Due to this dedicated overall Network-Oriented Modelling approach for adaptive networks, no procedural, algorithmic or programming skills are needed to

M. Baldoni et al. (Eds.): PRIMA 2019, LNAI 11873, pp. 487–495, 2019.
https://doi.org/10.1007/978-3-030-33792-6_33

design cognitive agents or social networks which show complex adaptive behaviour of any order.

In the paper, in Sect. 2 the reified temporal-causal network architecture is explained in some detail. After this, more details are described of the specification format (Sect. 3) and the implemented modeling environment and its computational reified network engine (Sect. 4) developed. Finally, Sect. 5 is a discussion.

2 Modeling Adaptive Processes by Reified Networks

A conceptual representation of the network structure of a temporal-causal network model involves three main characteristics of the network structure; see [8], Chapter 2, or [11]. First, for the *connectivity characteristics* of the network, connection weights $\boldsymbol{\omega}_{X,Y}$ are used as a labels for connections from X to Y. Second, for the *aggregation characteristics* of a network, for each state Y a combination function $\mathbf{c}_Y(..)$ is used to aggregate (and modulate) causal impacts on state Y; they can contain parameters $\mathbf{p}$. Third, for the *timing characteristics* of a network, for each state Y a *speed factor* $\boldsymbol{\eta}_Y$ is used for timing of the causal effects. The difference equations used for simulation and mathematical analysis incorporate these three types of network characteristics $\boldsymbol{\omega}_{X,Y}$, $\mathbf{c}_Y(..)$, $\boldsymbol{\eta}_Y$:

$$Y(t+\Delta t) = Y(t) + \boldsymbol{\eta}_Y[\mathbf{c}_Y(\boldsymbol{\omega}_{X_1,Y}X_1(t), \ldots, \boldsymbol{\omega}_{X_k,Y}X_k(t)) - Y(t)]\Delta t$$

Here X_1, …, X_k are the states from which state Y gets its incoming connections. For aggregation a library with a number (currently 35) of standard combination functions are available as options, but also own-defined functions can be added.

Modeling adaptive networks asks for a dedicated network architecture in which different levels of adaptivity or plasticity can be modeled. Such an architecture has been proposed based on the notion of network reification [9, 10]. Reification (e.g., [4]), in general means making an abstract notion concrete. For network models this is done by introducing additional states in the network that explicitly represent characteristics of the network such as *connectivity*, *aggregation*, and *timing*, and makes them adaptive:

- **Adaptation of a connection weight $\boldsymbol{\omega}_{X,Y}$**: reified connection weight representations $\mathbf{W}_{X,Y}$
- **Adaptation of a speed factor $\boldsymbol{\eta}_Y$**: reified speed factor representations $\mathbf{H}_Y$
- **Adaptation of a combination function $\mathbf{c}_Y(..)$**: reified combination function weight representations $\mathbf{C}_{i,Y}$ (for the i^{th} combination function used)
- **Adaptation of a combination function parameter $\mathbf{p}_Y$**: reified combination function parameter representations $\mathbf{P}_{i,j,Y}$ (for the j^{th} parameter of the i^{th} combination function for Y)

In a graphical representation in a 3D format these new states are depicted in a second plane above the plane for the base network; see the blue plane in the example model depicted in Fig. 1, also indicated as the first reification level. This step can be repeated so that a third plane is added for second-order reification (see the purple third plane in Fig. 1). Three types of causal connections are distinguished: upward causal connections, downward causal connections and leveled (horizontal) causal connections. The downward causal connections have their own fixed role and meaning in the sense that they are causally effectuating one of the four types of adaptations listed above.

Combination functions are built as a weighted average from a number of basic combination functions $\mathrm{bcf}_i(..)$ available in a library; these weights can be prespecified as constant values or can be adaptive based on reification states. Examples of basic combination functions often used are the *euclidean combination function* $\mathbf{eucl}_{n,\lambda}(\ldots)$ with order $n > 0$ and scaling factor $\lambda > 0$ (generalising the linear scaled sum function for $n = 1$) and the *advanced logistic sum* combination function $\mathbf{alogistic}_{\sigma,\tau}(\ldots)$ with steepness parameter $\boldsymbol{\sigma} > 0$ and excitability threshold parameter $\boldsymbol{\tau}$:

$$\mathbf{eucl}_{n,\lambda}(V_1,\ldots,V_k) = \sqrt[n]{\frac{V_1^n + \ldots + V_k^n}{\lambda}}$$

$$\mathbf{alogistic}_{\sigma,\tau}(V_1,\ldots,V_k) = \left[\frac{1}{1+\mathrm{e}^{-\sigma}(V_1+\ldots+V_k-\tau)} - \frac{1}{1+\mathrm{e}^{\sigma\tau})}\right](1+\mathrm{e}^{-\sigma\tau})$$

Here the V_i denote the single impacts $\boldsymbol{\omega}_{X_i,Y}X_i(t)$ on state Y for each of the incoming connections from states $X_1, \ldots, X_k$. Moreover, for Hebbian learning ('neurons that fire together, wire together'), among others the following combination function is available (used for the reification state $\mathbf{W}_{X,Y}$ in the middle layer in Fig. 1):

$$\mathbf{hebb}_{\boldsymbol{\mu}}(V_1, V_2, W) = V_1V_2(1-W) + \boldsymbol{\mu}W$$

where V_1, V_2 indicate the single impacts from the connected states (base states at the bottom layer in Fig. 1) and W the connection weight (represented by reification state $\mathbf{W}_{X,Y}$ in the middle layer in Fig. 1), and $\boldsymbol{\mu}$ is a persistence parameter. In Fig. 1:

- $\mathbf{W}_{X,Y}$ plays the role of connection weight for the connection from X to Y
- $\mathbf{H}_Y$ the role of speed factor for Y
- $\mathbf{C}_{i,Y}$ the role of combination function weight of $\mathrm{bcf}_i(..)$ for Y
- $\mathbf{P}_{i,,j,Y}$ the role of combination function parameter value; examples of such reified parameters used in Fig. 1 are the excitability parameters $\boldsymbol{\tau}$ (reified by the two **T** states in the middle plane) and the persistence parameter $\boldsymbol{\mu}$ (reified by the **M** state in the upper plane)

These values are used in the computations for base states Y depending on their role. For any base state Y the following *universal combination function* $\mathbf{c}^*_Y(..)$ is used:

$$\mathbf{c}^*_Y(H, C_1, \ldots, C_m, P_{1,1}, P_{2,1}, \ldots, P_{1,m}, P_{2,m}, W_1, \ldots, W_k, V_1, \ldots, V_k, V) =$$
$$H\frac{C_1\mathrm{bcf}_1(P_{1,1},P_{2,1},W_1V_1,..,W_kV_k) + \ldots.. + C_m\mathrm{bcf}_m(P_{1,m},P_{2,m},W_1V_1,..,W_kV_k)}{C_1 + \ldots.. + C_m} + (1-H)\,V$$
where

- H is used for the speed factor reification $\mathbf{H}_Y(t)$
- C_j for the combination function weight reification $\mathbf{C}_{i,Y}(t)$

- $P_{i,j}$ for the combination function parameter reification $\mathbf{P}_{i,j,Y}(t)$
- W_i for the connection weight reification $\mathbf{W}_{X_i,Y}(t)$
- V_i for the state value $X_i(t)$ of base state X_i
- V for the state value $Y(t)$ of base state Y

This universal combination function is used in the following *universal computational (difference) equation* (leaving t out of most of the notation):

$$Y(t+\Delta t) = Y(t) + [\mathbf{c}*_Y(\mathbf{H}_Y, \mathbf{C}_{1,Y}, \ldots, \mathbf{C}_{m,Y}, \mathbf{P}_{1,1,Y}, \mathbf{P}_{2,1,Y}, \ldots, \mathbf{P}_{1,m,Y}, \mathbf{P}_{2,m,Y}, \mathbf{W}_{X_1,Y}, \ldots, \mathbf{W}_{X_k,Y}, X_1, \ldots, X_k, Y(t)) - Y(t)]\Delta t$$

$$= Y(t) + \mathbf{H}_Y$$

$$[\frac{\mathbf{C}_{1,Y}\mathrm{bcf}_1(\mathbf{P}_{1,1,Y}, \mathbf{P}_{2,1,Y}, \mathbf{W}_{X_1,Y}X_1, .., \mathbf{W}_{X_k,Y}X_k) + \ldots\ldots + \mathbf{C}_{m,Y}\mathrm{bcf}_m(\mathbf{P}_{1,m,Y}, \mathbf{P}_{2,m,Y}, , \mathbf{W}_{X_1,Y}X_1, .., \mathbf{W}_{X_k,Y}X_k)}{\mathbf{C}_{1,Y} + \ldots\ldots + \mathbf{C}_{m,Y}} - Y(t)]\Delta t$$

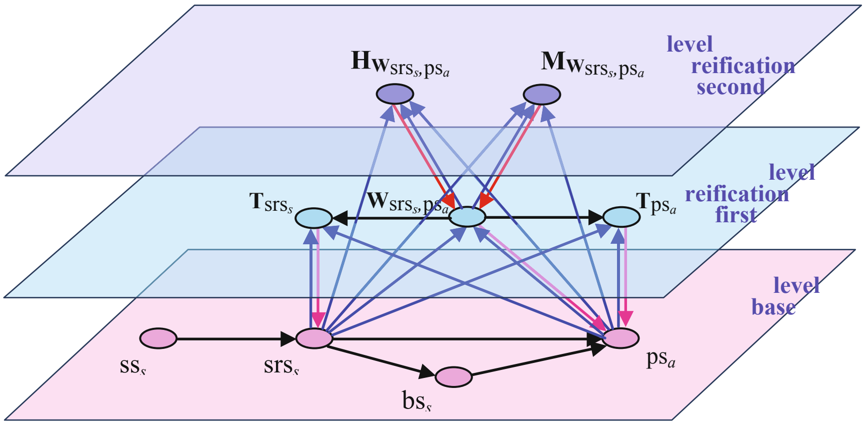

Fig. 1. Overview of an example reified network architecture addressing plasticity and metaplasticity for a cognitive agent model, with: (1) *base level* (lower plane, pink), (2) first reification level (middle plane, blue) for *plasticity* of the weight **ω** of the base connection from srs_s to ps_a and the excitability thresholds **τ** of these two base states (by the **W** state and the two **T** states), and (3) second reification level (upper plane, purple) for *metaplasticity* for the first-order adaptation speed **η** and the persistence **μ** (by the **H** state and **M** state). The upward causal connections (blue) and downward causal connections (red) define the interlevel relations. For more explanation of this example network, see [13]. (Color figure online)

In these formulas, by its place in the formula, each role indeed contributes a different type of effect according to its intended semantics. In Sect. 3 it is shown how in a network model design, the roles of these reification states are specified by *role matrices* **mb** (base connection role), **mcw** (connection weight role), **ms** (speed factor role), **mcfw** (combination function weight role), and **mcfp** (combination function parameter role).

Note that in a reified network the specific names of the reification states are computationally irrelevant: in this network modeling style the connections and their roles define meaning and processing, not the state names. This may be considered in contrast to reification in logic-based languages like (meta)Prolog [2, 7] where usually syntactical structures of names are processed.

3 Specification Format for a Reified Temporal-Causal Network

In role matrices it is specified which other states have impact on a given state (the incoming arrows in Fig. 1), but distinguished according to their role: *base* or *non-base* connections, from which for the latter a distinction is made for the roles *connection weight*, *speed factor*, *combination function weight* and *combination function parameter reification* (see also Fig. 1). Role matrices enable to apply structure-preserving implementation. The matrices all have rows according to the numbered states X_1, X_2, X_3, …..

For a given application a limited sequence of combination functions is specified by **mcf** = [….], for the example **mcf** = [1 2 3], where the numbers 1, 2, 3 refer to the numbering in the function library which currently contains 35 combination functions, the first three being $\mathbf{eucl}_{n,\lambda}(\ldots)$, $\mathbf{alogistic}_{\sigma,\tau}(\ldots)$, $\mathbf{hebb}_{\mu}(\ldots)$. In Box 1 the role matrices **mcfw** and **mcfp** (3D matrix) are shown. The first role matrix **mb** for *base connectivity* specifies on each row for a given state from which states at the same or a lower level it has incoming connections; see Box 1. For example, in the third row it is indicated that state X_3 (= bs_s) only has one incoming base connection, from state X_2 (= srs_s). As another example, the fifth row indicates that state X_5 (= $\mathbf{W}_{\text{srs}_s,\text{ps}_a}$) has incoming base connections from X_2 (= srs_s), X_4 (= ps_a) and from X_5 itself, and in that order, which is important as the Hebbian combination function $\mathbf{hebb}_{\mu}(\ldots)$ used here is not symmetric.

In a similar way the four types of role matrices for *non-base connectivity* (i.e., connectivity from reification states at a higher level of reification: the downward arrows in Fig. 1), were defined: role matrices **mcw** for connection weights and **ms** for speed factors, and role matrices **mcfw** for combination function weights and **mcfp** for combination function parameters (see Box 1).

Within each role matrix a difference is made between cell entries indicating (in red) a reference to the name of another state that as a form of reification represents in a dynamic manner an adaptive characteristic, and entries indicating (in green) fixed values for nonadaptive characteristics. Indeed, in Box 1 it can be seen that the red cells of the non-base role matrices are filled with the (reification) states X_5 to X_9 of the first and second reification levels. For example, in Box 1 the name X_5 in the red cell row-column (4, 1) in role matrix **mcw** indicates that the value of the connection weight from srs_s to ps_a (as indicated in role matrix **mb**) can be found as value of the fifth state X_5. In contrast, the 1 in green cell (5, 1) of **mcw** indicates the static value of the connection weight from X_2 (= srs_s) to X_5 (= $\mathbf{W}_{\text{srs}_s,\text{ps}_a}$). Similarly, role matrix **ms** indicates (in red) that X_8 represents the adaptive speed factor of X_5, and (in green) that the speed factors of all other states have fixed values. For more explanation about this role matrix specification format and the above example, see [12, 13] or the forthcoming book [14].

mb	**base connectivity**	1	2	3	4
X_1	ss$_s$	X_1			
X_2	srs$_s$	X_1			
X_3	bs$_s$	X_2			
X_4	ps$_a$	X_2	X_3		
X_5	**W**srs$_s$,ps$_a$	X_2	X_4	X_5	
X_6	**T**srs$_s$	X_2	X_4	X_6	
X_7	**T**ps$_a$	X_2	X_4	X_7	
X_8	**HW**srs$_s$,ps$_a$	X_2	X_4	X_5	X_8
X_9	**MW**srs$_s$,ps$_a$	X_2	X_4	X_5	X_9

mcw	**connection weights**	1	2	3	4
X_1	ss$_s$	1			
X_2	srs$_s$	1			
X_3	bs$_s$	1			
X_4	ps$_a$	X_5	1		
X_5	**W**srs$_s$,ps$_a$	1	1	1	
X_6	**T**srs$_s$	-0.4	-0.4	1	
X_7	**T**ps$_a$	-0.4	-0.4	1	
X_8	**HW**srs$_s$,ps$_a$	1	1	-0.4	1
X_9	**MW**srs$_s$,ps$_a$	1	1	1	1

mcfw	**combination function weights**	1 **eucl**	2 **alogistic**	3 **hebb**
X_1	ss$_s$	1		
X_2	srs$_s$		1	
X_3	bs$_s$		1	
X_4	ps$_a$		1	
X_5	**W**srs$_s$,ps$_a$			1
X_6	**T**srs$_s$		1	
X_7	**T**ps$_a$		1	
X_8	**HW**srs$_s$,ps$_a$		1	
X_9	**MW**srs$_s$,ps$_a$		1	

ms	**speed factors**	1
X_1	ss$_s$	0.5
X_2	srs$_s$	0.5
X_3	bs$_s$	0.2
X_4	ps$_a$	0.5
X_5	**W**srs$_s$,ps$_a$	X_8
X_6	**T**srs$_s$	0.3
X_7	**T**ps$_a$	0.3
X_8	**HW**srs$_s$,ps$_a$	0.5
X_9	**MW**srs$_s$,ps$_a$	0.1

mcfp	**function**	1 **eucl**		2 **alogistic**		3 **hebb**	
	parameter	1 n	2 λ	1 σ	2 τ	1 μ	2
X_1	ss$_s$	1	1				
X_2	srs$_s$			5	X_6		
X_3	bs$_s$			5	0.2		
X_4	ps$_a$			5	X_7		
X_5	**W**srs$_s$,ps$_a$					X_9	
X_6	**T**srs$_s$			5	0.7		
X_7	**T**ps$_a$			5	0.7		
X_8	**HW**srs$_s$,ps$_a$			5	1		
X_9	**MW**srs$_s$,ps$_a$			5	1		

Box 1. Specification in role matrices format for the example reified network for plasticity and metaplasticity

4 The Computational Reified Network Engine

The computational reified network engine developed takes a specification in the format as described in Sect. 3 and runs it. First each role matrix (which can be specified easily as table in Word or in Excel) is copied to Matlab in two variants: a *values matrix* for the static values (adding the letter **v** to the name) in the green cells, and an *adaptivity matrix* for the adaptive values represented by reification states (adding the letter **a** to the name) in the red cells. For example, from **mcw** two matrices **mcwa** (adaptivity matrix) and **mcwv** (values matrix) are derived in this way. The numbers in **mcwa** indicate the state numbers of the reification states where the values can be found, and in **mcwv** the numbers indicate the static values directly. States X_j are represented in Matlab by their index number j. Empty cells are filled with NaN (Not a Number) indications. During a simulation, for each step from k to $k + 1$ (with step size Δt, in Matlab dt) based on the above role matrices first for each state X_j the right values (either the fixed value, or the adaptive value) are assigned to:

`s(j, k)`	speed of X_j
`b(j, p, k)`	value for the p^{th} state connected to state X_j
`cw(j, p, k)`	connection weight for the p^{th} state connected to state X_j
`cfw(j, m, k)`	weight for the m^{th} combination function for X_j
`cfp(j, p, m, k)`	the p^{th} parameter value of the m^{th} combination function for X_j

Then, as a second part of the computational reified network engine, for the step from k to $k + 1$ the following is applied; here `X(j,k)` denotes $X_j(t)$ for t = `t(k) = kdt`:

```
for m=1:1:nocf
cfv(j,m,k) = bcf(mcf(m), squeeze(cfp(j, :, m, k)),
                 squeeze(cw(j, :, k)).*squeeze(b(j, :, k)));
end
   % This calculates the combination function values cfv(j,m,k)for
each combination function mcf(m) for state j at k
aggimpact(j, k) =
dot(cfw(j, :, k), cfv(j, :, k))/sum(cfw(j, :, k));
   % The aggregated impact for state j at k as inproduct of com-
bination function weights and combination function values, scaled
by the sum of these weights
X(j,k+1) =
X(j,k) + s(j,k)*(aggimpact(j,k) - X(j,k))*dt;
   % The iteration step from k to k+1 for state j
t(k+1) = t(k)+dt;
   % Keeping track of time
```

Note that functions with multiple groups of arguments here in Matlab get vector arguments where groups of arguments become vectors of variable length. For example, the basic combination function $\text{bcf}_i(P_{1,i}, P_{2,i}, W_1V_1, \ldots, W_kV_k)$ as expressed in Sect. 3 becomes bcf(i, p, v) in Matlab with vectors p = $[P_{1,i}, P_{2,i}]$ for function parameters and v = $[W_1V_1, \ldots, W_kV_k]$ for the values of the function arguments. This format bcf(i, p, v) is used as the basis of the combination function library developed (currently numbered by i = 1 to 35). As can be seen, the structure of the code of this computational reified network engine is quite compact, based on the universal difference equation discussed in Sect. 3: structure-preserving implementation. The combination function library used contains 35 functions at the time of writing. To obtain a general format easily usable within the simulations these functions were numbered and rewritten in the standard *basic combination function* form `bcf(i, p, v)` where i is the number of the function, **p** is its vector of parameters an **v** is a vector of values. A more detailed description of the software and a complete specification of the current combination function library can be found at [12].

5 Discussion

In this paper a modeling environment for reified temporal-causal networks was introduced, and applied to model a cognitive agent with plasticity and metaplasticity known from neuroscientific literature; e.g., [1, 6]. The environment includes a new specification format for reified networks and comes with a newly implemented dedicated computational reified network engine, which can simply run such specifications. Moreover, a library of currently 35 combination functions is offered, which can be used; this library can also be extended easily. Using this software environment, the development process of a model can focus in a declarative manner on the reified network specification and therefore is quite efficient, while still all kinds of complex (higher order) adaptive dynamics are covered without being bothered by implementation details. In a forthcoming book [14], more details and many more examples for this modeling approach will be presented.

Application may extend well beyond the neuro-inspired cognitive agents area, as also in Social Science cases are reported where network adaptation is itself adaptive; for example in [3] the second-order adaptation concept called ‘inhibiting adaptation’ for network organisations is described. For further work, it would be interesting to explore the applicability of the introduced modeling environment for such social agent domains as well.

References

1. Abraham, W.C., Bear, M.F.: Metaplasticity: the plasticity of synaptic plasticity. Trends Neurosci. **19**(4), 126–130 (1996)
2. Bowen, K.A., Kowalski, R.: Amalgamating language and meta-language in logic programming. In: Clark, K., Tarnlund, S. (eds.) Logic Programming, pp. 153–172. Academic Press, New York (1982)
3. Carley, K.M.: Inhibiting adaptation. In: Proceedings of the 2002 Command and Control Research and Technology Symposium, pp. 1–10. Naval Postgraduate School, Monterey, CA (2002)
4. Galton, A.: Operators vs. arguments: the ins and outs of reification. Synthese **150**, 415–441 (2006). https://doi.org/10.1007/s11229-005-5516-7
5. Hebb, D.O.: The organization of behavior: A neuropsychological theory. Wiley, New York (1949)
6. Magerl, W., Hansen, N., Treede, R.D., Klein, T.: The human pain system exhibits higher-order plasticity (metaplasticity). Neurobiol. Learn. Mem. **154**, 112–120 (2018)
7. Sterling, L., Beer, R.: Metainterpreters for expert system construction. J. Logic Program. **6**, 163–178 (1989)

8. Treur, J.: Network-Oriented Modeling: Addressing Complexity of Cognitive, Affective and Social Interactions. UCS. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45213-5
9. Treur, J.: Network reification as a unified approach to represent network adaptation principles within a network. In: Fagan, D., Martín-Vide, C., O'Neill, M., Vega-Rodríguez, M.A. (eds.) TPNC 2018. LNCS, vol. 11324, pp. 344–358. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04070-3_27
10. Treur, J.: Multilevel network reification: representing higher order adaptivity in a network. In: Aiello, L.M., Cherifi, C., Cherifi, H., Lambiotte, R., Lió, P., Rocha, Luis M. (eds.) COMPLEX NETWORKS 2018. SCI, vol. 812, pp. 635–651. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05411-3_51
11. Treur, J.: The ins and outs of network-oriented modeling: from biological networks and mental networks to social networks and beyond. In: Nguyen, N.T., Kowalczyk, R., Hernes, M. (eds.) Transactions on Computational Collective Intelligence XXXII. LNCS, vol. 11370, pp. 120–139. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-58611-2_2
12. Treur, J.: Design of a software architecture for multilevel reified temporal-causal networks (2019). https://www.researchgate.net/publication/333662169
13. Treur, J.: Network-oriented modeling of plasticity and metaplasticity (2019). https://www.researchgate.net/publication/335473145
14. Treur, J.: Network-Oriented Modeling for Adaptive Networks: Designing Higher-Order Adaptive Biological, Mental and Social Network Models. Studies in Systems, Decision and Control, vol. 251, pp. 314. Springer, Heidelberg (2020, to appear). https://doi.org/10.1007/978-3-030-31445-3, https://www.researchgate.net/publication/334576216