Under consideration for publication in Formal Aspects of Computing

A Tale of Two Graph Models: A Case Study in Wireless Sensor Networks

Blair Archibald[1] and Géza Kulcsár[2] and Michele Sevegnani[1]

[1]School of Computing Science, University of Glasgow, UK
[2]Real-Time Systems Lab, Technische Universität Darmstadt, Germany

Abstract.

Designing and reasoning about complex systems such as wireless sensor networks (WSN) is hard due to highly dynamic environments: sensors are heterogeneous, battery-powered, and mobile. While formal modelling can provide rigorous mechanisms for design/reasoning, they are often viewed as difficult to use. Graph rewrite-based modelling techniques increase usability by providing an intuitive, flexible, and *diagrammatic* form of modelling in which graph-like structures express relationships between entities while rewriting mechanisms allow model evolution.

Two major graph-based formalisms are Graph Transformation Systems (GTS) and Bigraphical Reactive Systems (BRS). While both use similar underlying structures, how they are employed in modelling is quite different. To gain a deeper understanding of GTS and BRS, and to guide future modelling, theory, and tool development, in this experience report we compare the *practical* modelling abilities and style of GTS and BRS when applied to topology control in WSNs. To show the value of the models, we describe how analysis may be performed in both formalisms.

A comparison of the approaches shows that although the two formalisms are different, from both a theoretical and practical modelling standpoint, they are each successful in modelling topology control in WSNs. We found that GTS, while featuring a small set of entities and transformation rules, relied on entity attributes, rule application based on attribute/variable side-conditions, and imperative control flow units. BRS on the other hand, required a larger number of entities in order to both encode attributes directly in the model (via nesting) and provide tagging functionality that, when coupled with rule priorities, implements control flow. There remains promising research mapping techniques between the formalisms to further enable flexible and expressive modelling.

Keywords: rewriting-based modelling, graph transformation systems, bigraphical reactive systems, wireless sensor networks

Correspondence and offprint requests to: Blair Archibald, email: blair.archibald@glasgow.ac.uk

1. Introduction

Reasoning about the design of large-scale and complex systems, such as wireless sensor networks (WSNs), is hard because they involve highly dynamic environments: devices are heterogeneous, requirements change, devices fail, there is lots of self-adaptation etc. Constructing suitable models of such systems allows rigorous design and analysis to take place, providing increased confidence in the correctness of a proposed solution [CCC+18]. Such analysis is essential in, for example, safety critical systems, and more generally as systems continue to pervade our everyday lives. In the case of WSNs, a model needs to specify not only the relationships between entities but also how these relationships evolve over time.

Graph-like structures are well suited to provide the foundational structure for these models. There is a natural correspondence: graph nodes represent domain entities and graph edges represent relations between them. This correspondence is further strengthened if the domain we are modelling has itself an inherent graph structure *e.g.* network topologies. Temporal evolution is typically modelled through some form of rewrite system over the graph structure. The diagrammatic nature of graph models provide an intuitive approach to modelling that requires only limited knowledge of mathematical modelling making them accessible to a wide range of practitioners.

In this paper we compare two graph based formalisms when applied to the same problem: communication and routing in wireless sensor networks. The two formalisms are (1) *Graph Transformation Systems* [EEPT06] (GTS), introduced by Ehrig and others, and (2) *Bigraphical Reactive Systems* [Mil09] (BRS), as proposed by Milner. Both formalisms are underpinned by a well-defined mathematical framework, and have been thoroughly investigated in their practical use. Historically, the development of GTS has been driven by research in software engineering and systems design, while development of BRS has followed the process calculi tradition of focusing on concurrency, *e.g.* through behavioural equivalence theory.

While our comparison focuses on GTS and BRS due to previous practical and theoretical experience with these graphical modelling formalisms, *e.g.* [KSS+14, KLS18, SKCM18, SC16, ASH+20], we acknowledge there are many equally valid tools for modelling different aspects of WSNs including process algebra [LS10, FvGH+12], (coloured) Petri-nets [DRM14], and probabilistic reactive modules [WBD+18].

Research into both GTS and BRS formalisms has led to a range of variants. Specifically, we compare the typed attributed graph transformations with control units/strategies found in Henshin [SBG+17] with *Bigraphs with Sharing* featured in BigraphER [SC16].

Both formalisms have been employed successfully in a diverse range of practical modelling problems. GTS are common in model driven engineering and software development [GdLW+13, PKLS16], while BRS have been used to model biological systems [KMT08], security in cyber-physical spaces [TPGN18, APN19] and IoT systems [SKCM18, ASH+20]. Both have seen use in networking [CKSS14, KSV+18, KVS15, KSS+14, CS14] due to the close relationship between graph models and graph based networks. While many models have been developed, no practical comparison between the formalisms has been undertaken.

The aim of this paper is to gain a deeper understanding of the *practical* modelling abilities of GTS and BRS to (1) guide application modelling *i.e.* by providing a tutorial-like construction of the two models, (2) to aid in choosing between the two modelling formalisms, *i.e.* we summarise (Table 3) the core capabilities/features of each approach, and (3) to guide future theory and tool development by detailing areas of promising development *e.g.* automated tools for inductive reasoning (Section 6.2), and highlighting where the approaches might learn from each other *e.g.* the benefits of adding application conditions to BRS or allowing hyperedges in GTS.

While there are tutorial-like papers describing GTS [KNPR18, Hec06], no similar works exist for BRS with the main reference [Mil09] focusing on theoretical aspects rather than practical application development. We address this here: providing practical examples of BRS construction, tips for overcoming common issues, *e.g.* fixed arity constraints, and also cover the *bigraph with sharing* case.

GTS and BRS share a number of basic principles and formal design characteristics: Models are *instances* of graph-based mathematical structures over a previously specified (and application-specific) *meta-model* that details domain entities and their relationships. To express model evolution, both GTS and BRS rely on declarative *rule-based rewriting*. *Rules* specify transformations of one (sub)-graph to another. An instantiation of a rule in a given input model results in a transformed output model. From this we can analyse the model by, for example, deriving transition systems or performing simulations.

The theoretical correspondence between GTS and BRS was elaborated early on [Ehr02, SS04], with a

particular focus on combining the reactive semantics of both approaches through cospan categories[1]. More recently, Gassara *et al.* [GRJD19] show a practical correspondence by converting BRS to GTS for execution.

We chose to compare GTS and BRS as, although they share basic principles and have a theoretical correspondence, the respective communities remain largely disjoint. Given the similarities we believe there is still much the GTS and BRS can learn from each other. For example: bringing modelling features from one to the other, *e.g.* adding inheritance support to BRS; improved tooling design, *e.g.* co-design of automated inductive reasoning tools; and new theory *e.g.* researching how application conditions from GTS might increase the expressiveness of BRS. To enable cross-fertilisation of ideas, we identity similarities/differences (from a practical modelling perspective) as a key first step.

To allow the similarities/differences to be identified, we summarise and generalise the practical experience of modelling and verification using both GTS and BRS. We use an appropriately simplified, yet realistic case study from the domain of wireless sensor networks that allows a demonstration and comparison of relevant aspects of the practical use of these formalisms. While we focus on WSNs, the approaches are applicable to a much wider range of systems, *e.g.* other networking scenarios [LYW+19]. Following [KSS+14] we model two notions of network topology (1) the *underlay* network that describes whether sensor nodes are within *physical* range of each other *should* communicate, and (2) the *overlay* network describes *virtual* links between sensors nodes indicating a communication path between two sensors is *required* by some application. Both layers cooperate to ensure a valid routing path between virtually linked sensors (if one exists). We model both layers and show how they evolve over time in response to environmental changes *e.g.* dropping of links. Although the focus is on modelling, we show how the models allow analysis/verification of properties such as ensuring the design removes redundant links whenever possible.

We do not intend to establish a correspondence between GTS and BRS models by forcing the exact same implementation details into different frameworks. Instead, we focus on the domain-specific requirements of the WSN scenario and highlight the inherent differences in modelling style by deliberately diverging in details where appropriate. Likewise we do not claim this is the only way to express these models, and acknowledge that model writing is often a creative endeavour in its own right.

To the best of our knowledge, this is the first comparison study of GTS and BRS in terms of practical modelling ability. We hope this encourages further cross-fertilisation between GTS and BRS communities and we make the following research contributions:

- We model the underlay and overlay networks, in particular topology control aspects, of a wireless sensor system using both Graph Transformation Systems (GTS) and Bigraphical Reactive Systems (BRS). Complete reference models are provided[2]
- We provide a detailed comparison of the modelling capabilities and approaches of both GTS and BRS, highlighting their key differences and providing insight for model, theory and tool development.
- We show how formal analysis/verification techniques, in particular model checking and inductive reasoning, apply to both formalisms; this allows network properties to be checked/proved at design time.

Overview

The paper is structured as follows. Section 2 describes the WSN case study: introducing the *underlay* and *overlay* network layers, and analysis goals. Section 3 provides background on GTS and BRS theory. Section 4 and Section 5 present the GTS and BRS models for the underlay and overlay network. Discussion and comparison of approaches is interspersed throughout these sections. Section 6 shows the state-of-the-art analysis capabilities for both GTS and BRS. Section 7 brings together our discussions of GTS/BRS, compares the two formalisms, and highlights areas of future work. We conclude in Section 8.

[1] These results apply mainly to input linear bigraphs that differ slightly from Milner's original formalisation in their treatment of interfaces/sites.

[2] GTS: `https://github.com/timofr/sosywsn/tree/master/modeling/henshin/WSN_Henshin`; BRS: `http://www.dcs.gla.ac.uk/~michele/wsn.big`

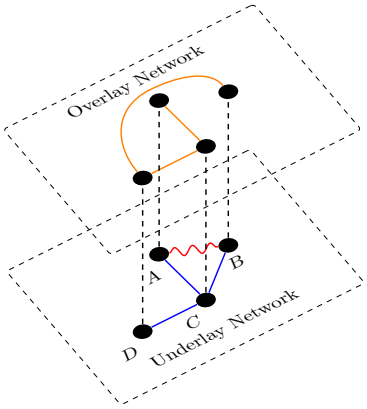


Fig. 1. Case study overview. Underlay network shows (one possible) configuration of link statues (red wave = inactive, blue straight = active). Overlay network shows virtual links between sensors

2. Modelling Scenario

To show the differences in modelling with GTS and BRS, we focus on a *wireless sensor network* (WSN) scenario. WSNs consist of a set of sensor *nodes* (sensors) that are deployed into a physical space in order to sense (and perhaps modify through actuators) real world environments. WSNs are employed in a wide range of contexts such as data collection, smart buildings and cities, biological experiments, military applications, etc. [LML08].

We chose WSNs as a representative example of the type of system we, and others – with topology control systems being modelled using GTS previously [KSV$^+$18, KVS15, KSS$^+$14] – wish to model. They are highly dynamic: sensors may move, *e.g.* utilising mobile phone sensors, communication links change over time, *e.g.* in respond to environmental degradation of the network, and node failure is not uncommon [BTM18]. WSNs are also ideal for graph based modelling formalisms that can accurately model the (graph based) underlying network.

We focus specifically on the communication/networking aspects of WSNs. In particular, we model how topology control algorithms allow WSNs to dynamically organise their communication to optimise the distribution of collected data even in highly dynamic environments. We model WSN networking at two layers as shown graphically in Fig. 1. The operation of each layer is as follows:

Underlay Network: The underlay network models *physical communication* links between sensors. A communication link can be established when a sensor is in the sending/receiving range of another. To save power, the network can choose to not use a link even if it is physically possible to do so. That is, links between specific sensors may be either *active*, *i.e.* available for communication [KSV$^+$18], or *inactive*. For example, in Fig. 1, A is within range of both C and B. However, the underlay network has determined that A should communicate only with C directly, and never with B. We assume all communication links are bidirectional.

Overlay Network: The overlay network defines the links required between sensors for *application* correctness, even if they have no physical link. For example, in Fig. 1, a virtual link specifies that nodes D and B must communicate. As there is no physical link between D and B they must communicate through an active path, *i.e.* through C.

Note that the underlay and overlay networks are not fully independent and, for example, the overlay network may influence the underlay network to form new paths between sensors (if physically possible) such that application requirements are met.

While we model at the level of the network topology between sensors, different, and potentially more complex, models are possible *e.g.* modelling individual packets. There is always a trade-off between getting the right level of abstraction while ensuring we can still reason about the properties of interest.

2.1. Underlay Network

Physical communication is performed through wireless channels, *i.e.* radio links between sensors. To define the range of a wireless channel we rely on the widely used unit-disk assumption. Here, each sensor node has a uniform transmission range and there is a potential communication link between two sensors whenever the two sensors are in range of one-another. As sensor positions are not necessarily fixed, *e.g.* body sensors, links between sensors may vary over time.

Rather than broadcast in all directions, we allow the sensors to use directional radio links to reduce transmission power. Given this, we treat communication as direct bi-directional links between specific sensors that may be either *active*, *i.e.* available for communication [KSV+18], or *inactive*. Management of links relies on a marking principle, and, we assign an intermediate edge status *unclassified* to those links yet to be assigned a definitive state.

We model three key behaviours of the underlay network:

Sensor behaviour The movement of sensors is modelled through the changes they induce in the topology. In particular, a *Create Link* operation occurs when two sensors move close enough to communicate, while a *Delete Link* operation occurs when two sensors move out of communication range. Links are created in the unclassified state.

Link behaviour Independently of sensor movement, environmental events may make it necessary to revise the status of a link. This is represented by the *Unclassify* operation, that turns an active or inactive link to unclassified, allowing topology control to reassign the link status to optimise data movement.

Topology control Topology control performs status revision of unclassified links to reduce the number of active links required. In particular, the *Resolve Unclassified Link* operation inactivates an unclassified link if there is an alternative active 2-hop path between its end nodes, or activates it otherwise.

2.2. Overlay Network

The overlay network maintains a *virtual* set of links between sensor nodes that are required to communicate *e.g.* for a particular application. A virtual link represents the requirement that there is *at least one* underlay network path between the end nodes with no inactive link. To avoid confusion, we refer to such a path as an `a-u` path (meaning a path consisting of active and unclassified links). We model the following behaviour of the overlay network:

Requirement Management The *Create Virtual Link* operation creates a new *requirement* for two sensors to communicate by creating a virtual link between them. Likewise, the *Delete Virtual Link* removes unneeded virtual links. These two operations model a user, *e.g.* administrator/supervisor, updating the required high-level network specification.

2.3. Interaction between Underlay and Overlay Network

Both network layers interact to ensure the virtual link requirements, set by the overlay layer, are met (if possible) by the underlay layer. This is represented using the following behaviour:

Routing Maintenance Given a set of virtual links, a routing maintenance algorithm maintains/creates `a-u` paths in the *underlay network* to ensure the expected connectivity (if possible). In particular, given a virtual link, the *Search Active Path* operation determines if there is a `a`-only path between the end nodes. If there is no such path but there is an `a-u` path, the *Activate `a-u` Path* operation should iteratively activate all the unclassified links in order to achieve an `a`-path. If there was neither an `a`-path nor an `a-u` path available, then the *Mark Inactive Path* operation should unclassify inactive links in one of the appropriate path candidates to allow the underlay network topology control algorithm to activate them if possible.

2.4. Analysis Goals

A benefit of constructing formal models of systems is the ability to verify rigorously properties of the modelled behaviour. We verify properties of the model through state space analysis where, given an initial model instance (the *start state*), we induce the full state space/transition system by executing transformation/rewrite rules in all possible orders. We can then formulate and check predicates about the states reachable from a given start state via *Reachability analysis*.

These properties are commonly found in topology control scenarios [San05]. Ideally the system provides: 1. connectivity between nodes, 2. eventual consistency of the routing (*e.g.* eventually discovering all nodes), 3. optimisations to reduce the number of paths. These properties are not exhaustive, and other properties could be explored. A key advantage of a formal model is the ability to explore additional properties quickly without requiring expensive implementation and testing.

2.4.1. Underlay Properties

We say that a topology is *connected* if there is at least one `a-u` path between each pair of sensors, *i.e.* no sensor is unreachable. An *active-triangle* occurs where there are three active links between three nodes. Deactivating any one of the links still allows all three nodes to communicate through multi-hop communication. Given these definitions, we consider the following underlay properties:

Connectedness: Given a connected start state, on any execution trace in the full transition system, where no link gets deleted, all reachable states are connected.

Redundancy Reduction: In a state with no unclassified links, there are no active triangles, *i.e.* topology control successfully reduces the number of active links required.

Liveness Resolution: From a state with unclassified links, given no operations that add new unclassified links (*e.g. Unclassify*, *Create Link*, *Mark Inactive Path*) occur, then we eventually reach a state – through repeated application of *Resolve unclassified link* – containing no unclassified links.

2.4.2. Overlay Properties

We aim to verify the following overlay property:

Virtual Link Fulfillment: Given a state where any virtual links have no `a-u` path, we reach a future state where all virtual links obtain an `a-u` path.

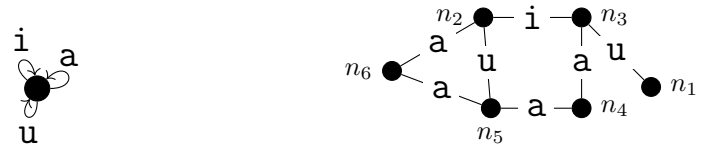
That is, we ensure topology control satisfies application requirements. Note that this is not always possible, for example when we require communication with an isolated sensor, in which case we expect the property to be false.

3. Technical Background

Graph rewriting systems are often best understood from a categorical standpoint and as such this section assumes knowledge of elementary category theory. We have ensured the rest of the paper does not require an understanding of the categorical details, and note that users of graph rewriting systems likewise do not require knowledge of category theory to make use of the formalisms for modelling.

3.1. Graph Transformation Systems (GTS)

Algebraic graph transformation [EEPT06] is formalised in a categorical setting, where objects are graphs and arrows are graph morphisms, *i.e.* structure-preserving functions on nodes and edges. For modelling, the category of all graphs is too loose, and instead we wish to form only graphs that are valid for a given modelling domain. To achieve this we constrain the graph objects of interest using a *type graph*. Type graphs specify a *meta-model* that determines the model entities (as nodes) as well as relationships between them (as one-to-one edges) *e.g.* entity e_0 may connect to n entity e_1's. A graph that adheres to a particular type graph T is said to be typed over T.



(a) Type Graph

(b) Example topology instance based on the type graph of Fig. 2a

Fig. 2. GTS Type Graph example

Definition 1 (Graphs and Typed Graphs)**.** *A* (directed) graph *is a tuple* $G = \langle N, E, s, t\rangle$*, where* N *and* E *are finite sets of nodes and edges, and* $s, t : E \to N$ *are the source and target functions on edges. We denote the components of a graph* G *as* N_G*,* E_G*,* s_G*,* t_G*.*

A graph morphism $f : G \to H$ *is a pair of functions* $f = \langle f_N : N_G \to N_H, f_E : E_G \to E_H\rangle$ *such that* $f_N \circ s_G = s_H \circ f_E$ *and* $f_N \circ t_G = t_H \circ f_E$*. A graph morphism is an isomorphism if both* f_N *and* f_E *are bijections. A* partial graph morphism $g : G \to H$ *is a graph morphism* $dom(g) \to H$ *where* $dom(g)$ *is a subgraph of* G*. We denote the category of graphs and partial graph morphisms as* **GraphP***.*

A type graph *is a distinguished graph* $T = \langle N_T, E_T, s_T, t_T\rangle$ *over node and edge types* N_T *and* E_T*.*

The category of typed graphs *over a* type graph T *is the slice category* (**GraphP** $\downarrow T$)*, also denoted* $\mathbf{Graph}_T$ *[CMR96]. That is, objects of* $\mathbf{Graph}_T$ *are pairs* (G, t) *where* $t : G \to T$ *is a* typing *morphism, and an arrow* $f : (G, t) \to (G', t')$ *is a morphism* $f : G \to G'$ *such that* $t' \circ f = t$*.*

For example, Fig. 2a shows a type graph for representing a simple WSN topology. It specifies that a node entity (shown as a black circle) may connect to other nodes (including itself) through `i`, `a`, or `u` labelled links. Labels on edges represent their type, and the type encodes the status of the link (active, inactive, unclassified). An *instance* of this typed graph is in Fig. 2b.

3.1.1. Rewriting

Graph transformation usually takes the form of a pushout in **GraphP**. The main constructions are double-pushout (DPO) and single-pushout (SPO) rewriting that, from a practical standpoint, differ mainly in their treatment of dangling edges, *i.e.* in SPO deleting a node also deletes dangling connected edges. As we do not delete nodes in our WSN model the differences are minimal. We introduce SPO rewriting here as the DPO-approach can be embedded within the SPO-approach [EHK+97], *i.e.* there are notions of equivalence between them. Practically, tools choose a preferred rewriting approach, but the rules provided by the user are similar in both approaches. We assume rules presented here are well formed.

Intuitively, an *SPO graph transformation rule* $p : L \to R$, describes how a sub-graph L may be rewritten into a sub-graph R. A rule consists of a left-hand side L, a right-hand side R, and a partial morphism p describing how they are connected, *i.e.* which nodes/edges are the same. As p is partial we can express nodes that are deleted from L, *i.e.* have no corresponding node in R, and new nodes created by R, *i.e.* with no corresponding node in L. Transforms are type-safe in that L and R must both be instances of the same type graph.

Applying a rule to a graph G consists of 3 steps:

1. Find an occurrence/match of L in G
2. Delete elements of the match not in R
3. Creates elements of R that do not appear in L

A formal definition of SPO rewriting is given below.

Definition 2 (SPO Rule)**.** *An* SPO rule $r = p : L \to R$ *consists of a rule name* r *and a partial graph morphism* $p : L \to R$*. An* SPO rule application *applying rule* r *at* match $m : L \to G$ *(total morphism) such that* $f : G \to H, h : R \to H$ *is a pushout over* p, m *in* **GraphP** *according to the diagram:*

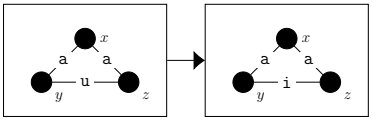


Fig. 3. Rule INACT: Inactivate Unclassified Edge

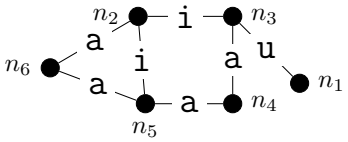


Fig. 4. Figure 2b after applying INACT

$$\begin{array}{ccc} L & \xrightarrow{p} & R \\ \downarrow m & (PO) & \downarrow h \\ G & \xrightarrow{f} & H \end{array}$$

An SPO rule with negative application condition *(NAC)* $n = N \xleftarrow{n} L \xrightarrow{p} R$ *consists of a rule name* r *and a span of morphisms with* n *total and* p *partial. An* SPO rule application *with NAC transforms the graph* G *to* H*, as a result of applying rule* $n = N \xleftarrow{n} L \xrightarrow{p} R$ *at* match *(total morphism)* $m : L \to G$ *so that* $f : G \to H, h : R \to H$ *is a pushout over* p, m *in* **GraphP**. *Additionally, there is no morphism* $c : N \to G$ *with* $c \circ n = m$*, as seen in the diagram:*

$$\begin{array}{ccccc} N & \xleftarrow{n} & L & \xrightarrow{p} & R \\ & \searrow & \downarrow m & (PO) & \downarrow h \\ & & G & \xrightarrow{f} & H \end{array}$$

For example, the rule INACT in Fig. 3 represents the inactivation of an unclassified link as specified by the *Resolve Unclassified Link* operation: whenever a triangle with two a-edges and one u-edge can be found in the input graph, the u-edge gets inactivated (set to i).

The result of applying INACT to the instance in Fig. 2b is shown in Fig. 4. Here we match $n_6 = x$, $n_2 = y$, and $n_3 = z$, allowing the link in between n_2 and n_5 to be made active.

A rule featuring a NAC is shown in Fig. 5. ACT_NAC is applied conditionally only if there is *no* active triangle of links between y and z, *e.g.* through x. NACs are only specified on the left hand side of the rule as they are checked during the matching process and never the creation process.

Formally, *graph transformation systems* (GTS) consist of a set of rules and (optionally) a start graph. In practice, the term GTS often simply refers to the whole formalism.

Although, according to the definition of GTS, a transformation rule may be applied whenever a match is found, this is often undesirable from a practical standpoint, *e.g.* when we wish a rule to apply only if another rule has been applied previously. To overcome this, the use of imperative *control units* [ABJ+10] is common in practical GTS. Control units capture imperative behaviour such as: *looping*, by applying a given rule as many times as possible (or n times); *conditionals*, where a choice of two rules are applied based on the existence/absence of a given match; *sequentially* applying rules one after another; and applying rules in a *priority* order. That is, control units determine the conditions for rule application.

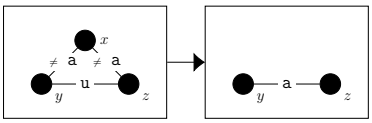


Fig. 5. Negative Application Condition Rule ACT_NAC: Active Unclassified Edge

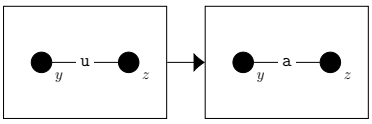


Fig. 6. Rule Act: Active Unclassified Edge

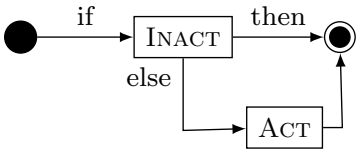


Fig. 7. Conditional unit example. Rule Act should only be applied if the pattern specified by the graph of the left-hand-side of rule Inact does not occur in the current model instance

An example control unit is in Fig. 7. It specifies that rule Act (Fig. 6) should only be applied when there are no matches of the pattern graph specified by the left-hand-side of Inact. Using this control unit, we allow the left-hand-side of Inact to operate like a negative application condition for Act. This allows us to replace the Act_NAC rule with the simpler Act rule.

While control units are flexible, unlike rewriting, they are not included in the formal GTS theory. Throughout this paper we use GTS to refer to GTS with control units.

GTS enjoy a variety of implementations ranging from domain agnostic tools (*e.g.* Groove [GdMR+12], AGG [Tae03]) to tools focusing on software engineering practice (*e.g.* Henshin [SBG+17], eMoflon [LAS14]). The GTS model presented here is implemented using Henshin and is available online[3].

3.2. Bigraphical Reactive Systems (BRS)

Bigraphs are a universal mathematical model for representing the spatial configuration of physical or virtual objects and their interaction capabilities. They were initially introduced by Milner [Mil09] and then extended to *bigraphs with sharing* in [SC15] to accommodate spatial locations that can overlap. For brevity, we use the term 'bigraphs' to refer to 'bigraphs with sharing'. A bigraph consists of a pair of relations over the same set of entities: a directed acyclic graph, called the *place graph*, representing topological space in terms of containment, and a hyper-graph, called the *link graph*, expressing the interactions and (non-spatial) relationships among entities. Each entity is assigned a *type* that determines its *arity*, *i.e.* number of links, and whether it is *atomic* *i.e.* it cannot contain other entities.

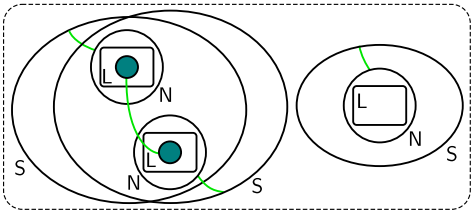
Bigraphs can be described in algebraic terms or with an equivalent diagrammatic representation. We focus on the diagrammatic representation here as an intuitive notation, and the equivalent algebric terms may be found in the provided model[4].

Example bigraphs are in Figs. 8a and 8b. In the diagrammatic representation of bigraphs we permit any kind of coloured shape as shorthand notation for particular entities – indicated by labels S, N, L, etc.

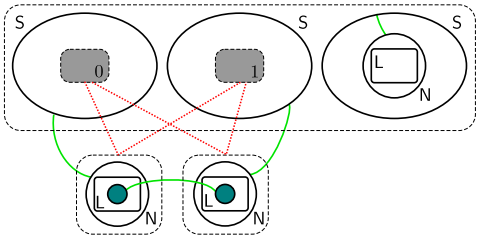
Spatial placement of entities is described in three ways: *Nesting* defines the containment relation on entities, *e.g.* an L within an N; *merge product* places two entities side-by-side at the same spatial location, *e.g.* two L's within the *same* N; *parallel product* is similar but entities are placed in two different locations. Filled grey rectangles denote *sites* that indicate parts of the model that have been abstracted away, *i.e.* an unspecified bigraph (including the empty bigraph) may exist there. Unfilled dashed rectangles drawn around sets of entities indicate *regions* of adjacent parts of the system. Intuitively, to create larger bigraphs from smaller bigraphs we compose them by placing *regions* inside *sites* as shown by the red lines in Fig. 8b (in this case the regions appear in both sites due to sharing).

Connectivity between entities is represented by green edges called *links*. Links may be only partially specified, in which case they connect a *name* which is usually drawn above the bigraph. Entities must always have as many links as their arity, however links may be *closed* (disconnected).

[3] https://github.com/timofr/sosywsn/tree/master/modeling/henshin/WSN_Henshin
[4] http://www.dcs.gla.ac.uk/~michele/wsn.big



(a) Bigraphical model of an example WSN of three nodes (N) and one active communication link shown as the link between the two small teal circles



(b) Equivalent graphical form highlighting sharing relation in red

Fig. 8. Example Bigraph representing a WSN

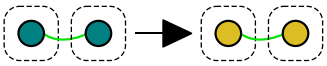


Fig. 9. `unclass_A`

Share expressions are a specialised version of nesting allowing to specify *shared* entities, *i.e.* entities situated in the intersection of other entities. An example of shared entities is in Fig. 8a where the N entities are nested below *two* different S entities. The sharing can be seen more clearly in the equivalent bigraph shown in Fig. 8b. Here, red dashed lines highlight the sharing relation between nodes and signals.

The bigraph of Fig. 8a models a WSN of three nodes with their overlapping wireless signals, denoted by types N and S, respectively. The full typing hierarchy, including allowed nesting relationships, will be introduced in the following sections (see Tables 1 and 2). When there is no ambiguity, we will omit explicit labels from the diagram. We also use shorthand shapes, for example, the teal circles are a shorthand for the bigraph created from entity E – representing a link end – nesting an A that acts as a token that this link end is *active*.

Bigraphs are *abstract* or *concrete*. In this paper we work primarily with abstract bigraphs, where entities do not have identifiers, *e.g.* N matches *any* physical sensor node in the system.

3.2.1. Rewriting

The dynamic aspects of the theory are expressed by means of rewriting through reaction rules.

Definition 3 (Reaction rule)**.** *A* reaction rule *is a pair* (R, R') *with* R *and* R' *bigraphs that can be inserted into the same host bigraph. A reaction rule, indicated by* $R \longrightarrow R'$*, is applicable to a bigraph* B *when* R *is an occurrence in* B*. The result of the application is bigraph* B' *which is obtained by substituting (in* B*) an occurrence of* R *with* R'*. Such a* reaction *is indicated with* $B \longrightarrow\!\!\!\!\triangleright\, B'$*.*

An example reaction rule, `unclass_A` is given in Fig. 9. This rule describes a reaction on underlay links to change their status from active (shown as the teal circles) and unclassified (shown as the yellow circles). Placing the link ends in separate regions states that they need not share a single parent. If only one region is used then they must have the same parent node. The result of applying `unclass_A` to the bigraph shown in Fig. 8a is shown in Fig. 10.

Reaction rules may also be equipped with *instantiation maps* to specify how the parameter of R should be composed with the parameter of R'. This powerful extension allows to easily duplicate or discard parts of

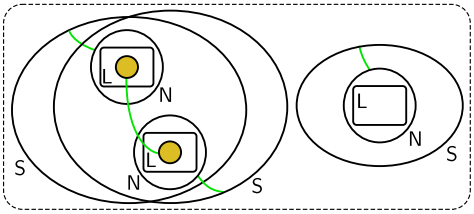


Fig. 10. Figure 8a after applying `unclass_A` (Fig. 9)

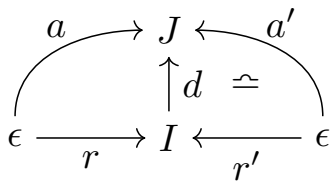


Fig. 11. Applying reaction rule $r \longrightarrow\!\!\blacktriangleright r'$ to a bigraph a

a bigraph upon reaction rule application. Graphically we represent this using blue arrows from the right-hand to the left-hand side (an example using instantiation maps is in Fig. 14b).

Finally, a *Bigraphical Reactive System* (BRS) consists of a set of reaction rules together with an initial bigraph on which the rules operate. Rule priorities, in the style of [BBKW89], can be introduced by defining a partial ordering on the reaction rules of a BRS. We write $\mathtt{r} < \mathtt{r}'$ to indicate that reaction rule $\mathtt{r}'$ has higher priority than reaction rule $\mathtt{r}$. As we will see, combing rule priorities and tagging schemes are a common method for controlling rule application, *i.e.* allowing features similar to GTS with control units. While we allow rule priorities to be directly specified for BRS, it is possible to encode priorities directly in BRS models allowing us to remain in the core BRS formalism.

3.2.2. Bigraph Categories

Like GTS, bigraphs may also be treated from a categorical perspective, this provides the basis for their rewriting semantics. The category of bigraphs takes *interfaces*, *i.e.* sites/regions and names, as objects and *bigraphs* as arrows. This is in contrast to GTS that has graphs as objects and graph homomorphisms as arrows.

Any bigraph can be treated as the composition (and tensor product) of arrows in the category of bigraphs[5]. The categorical models are high-level in that you cannot see the internal structure of the bigraph (*i.e.* nodes and edges), only how they glue together. Aside from re-writing the categorical framework provides powerful tools for deriving bisimulations (see [Mil09][Chapter 6]).

Rewriting is described categorically through the diagram shown in Fig. 11. In the diagram, I and J represent interfaces, with ϵ the empty interface implying no sites and names, while a, a', r, r', and d are bigraphs. The diagram shows how a bigraph a may be written as the composition of r and d. By replacing r with r', leaving the context d unchanged, we obtain a rewrite of a to a new bigraph a'. The use of ϵ as the domain of bigraphs a and r forces any rule parameters to be determined before matching occurs, *e.g.* sites are filled[6].

While the rules are specified for abstract bigraphs, rewriting itself happens on *concrete* bigraphs, where entities have identifiers. That is, we find a subgraph in the (concrete) bigraph that matches the pattern of the (abstract) rewrite rule. The operator $\simeq$ allows us to rewrite into a set of *support equivalent* a''s, *e.g.* we can change the identifiers during a reaction. We do not discuss support equivalence in detail here, but it allows, for example, a renaming of entities, while maintaining the structure of the bigraph itself.

This form of rewriting differs from GTS by rewriting entire sub-(bi)graphs at once as opposed to performing

[5]There are multiple categories of bigraphs covering the place-graphs/link-graph, abstract/concrete structures, lean-equivalent bigraphs etc; however these are all constructed a similar manner.

[6]This implies that a single reaction rule gives rise to a family of rules matching any given parameters.

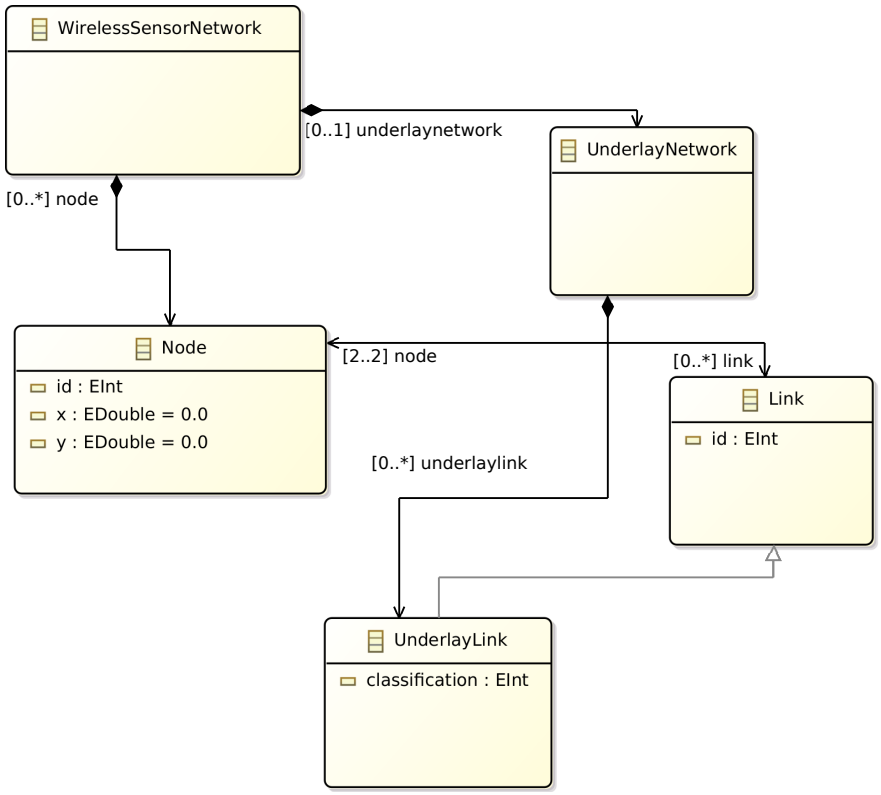


Fig. 12. GTS (Henshin) type graph for the underlay network showing entities and their relationships

deletions, mappings, and additions as three steps. Recent work explores a similar style of direct sub-graph rewriting applied to GTS [KR18].

Unlike GTS that enjoys a variety of implementations, there are only two implementations of BRS: BigRed [PDH13] and BigraphER [SC16]. The BRS model presented here uses BigraphER, which is available online[7].

4. Modelling WSN Underlay Networks

The underlay network has two main entities: sensor nodes and communication links between them. For modelling, we abstract away the physical connection details, *e.g.* protocols, and focus only on the network aspects.

For the remainder of the paper we use the notation – *Create Link* to refer to the model requirement, CREATE to refer to a GTS transformation rule modelling the requirement, and `create` to refer to a BRS reaction rule modelling the requirement.

The type graph for the GTS underlay model is given in Fig. 12. Here we use the Henshin format for type graphs that resembles UML class diagrams. As is common for practical modelling, the type graph allows for entity *attributes* [EEPT06, Chapter 8] to be specified. Entity attributes can be used for matching, *e.g.* match a Node only if it has a particular id, and may be updated during rewrites, *e.g.* moving a Node by updating the x, y coordinates. A sensor network consists of any number of Nodes and an (underlay) Link may exist between any two nodes. Different link statuses are represented by adding attributes to the links, *i.e.* via a classification attribute. Graphically we draw nodes as circles and link entities as graph links labelled with their classification (cf. Fig. 2b for a sample topology).

[7] `http://www.dcs.gla.ac.uk/~michele/wsn.big`

Table 1. Entities for the BRS of WSN underlay model

Description	Type	Arity	Atomic	Parent	Notation
Signal range	S	1			oval
Node	N	1		S	circle
Link	L	0		N	rounded box
End (any type)	E	1		L	small circle
Active end	E.A	1	✓	L	small teal circle
Inactive end	E.I	1	✓	L	small purple circle
Unclassified end	E.U	1	✓	L	small yellow circle
Tagged end	E.U′	1	✓	L	small amber circle

To create new links between nodes we need to know when they are close-enough to form a connection. This requires a basic encoding of the physical location of nodes. In the GTS model we achieve this through node attributes, where each node has an x, y location.

The BRS entities are given in Table 1. Bigraphs adopt a similar approach to GTS with both sensors and links being distinct entities. Using bigraphs with sharing, the model also expresses wireless signal ranges allowing additional checks that connected nodes meet the physical requirements. In our model, we use ovals to denote wireless signals (type S) and circles (type N) for nodes. Each node is linked to its signal (to allow it to be identified later) as in Fig. 8a. Communication links between nodes are modelled by bigraphical links between pairs of ends. These are entities of type E represented graphically as small circles. The type of a link is specified by the type of atomic entity contained by its ends. This is shown by the use of different coloured circles in the diagrammatic notation. All ends of a node are grouped within an entity of type L, indicated by a rounded box. This modelling strategy allows nodes to be connected to an arbitrary number of links while keeping the same type, overcoming the fixed arity limitation of bigraphs.

Unlike the graph model, bigraphs have a built-in notion of locality and we use this to describe the physical topology of the system. In this case we say two nodes are *close-enough* to form a connection if they share the same parent node. As we may always subdivide space, *e.g.* by splitting it into two smaller spaces, we can model arbitrary physical topologies.

Discussion 1 GTS excel in representing systems that themselves are graph-like, *e.g.* the underlay network. However, this flat, non-hierarchical structure is also a drawback of graph-based meta-models: frequent modelling patterns such as hierarchy layers, multiple connections, etc have to be encoded in the type graph and this potentially confuses different modelling concerns. On the other hand, a BRS model supports hierarchical structures allowing intuitive modelling of patterns such as layers, and physical aspects of domains to be directly modelled as topological spaces, *e.g.* two nodes sharing a single region. The physical topology, *i.e.* closeness, fits naturally into the bigraph model due to the built-in notion of locality, while for GTSs this must be encoded separately.

In terms of number of entities, GTS requires less entities[8] (4) than BRS (8). This is due to GTS allowing arbitrary numbers of links between nodes without a wrapper entity, and BRS requiring additional entities to tag the link statues as opposed to using attributes in the GTS model[9].

4.1. Dynamics

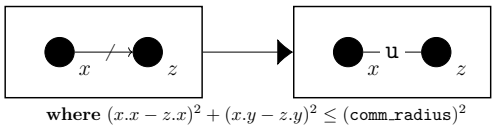
The dynamics of the underlay model is specified through graph transformation rules/bigraph reaction rules. In particular we specify: *node behaviour*, *e.g.* the creation of new links; *link behaviour*, *e.g.* unclassifying links; and *topology control*, *e.g.* (de-)activating links as required.

4.1.1. Node Behaviour

In GTS, underlay link creation/deletion corresponds to adding/removing a link entity to/from the graph and are specified using the transformation rules shown in Fig. 13. Create Link (Fig. 13a) makes use of a

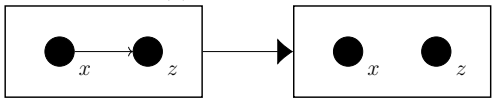
[8] Assuming all links are currently UnderlayLink

[9] BigraphER allows a basic form of attributes via *parameterised* entities. For example, Node(i) could describe a node with id i. We choose to use distinct entities here as this is closer to the original bigraph formalism.



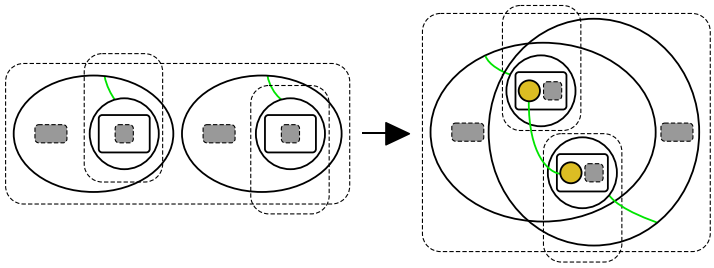
where $(x.x - z.x)^2 + (x.y - z.y)^2 \leq (\texttt{comm\_radius})^2$

(a) Create Link

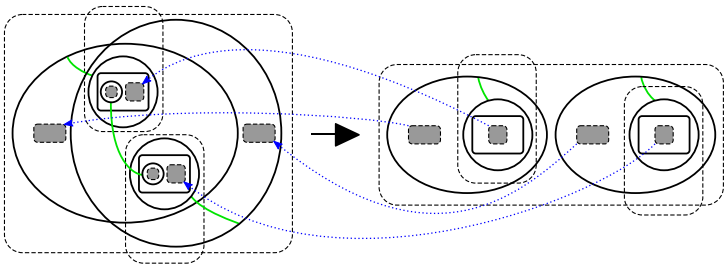


(b) Delete Link

Fig. 13. Node behaviour rules for GTS



(a) Reaction rule `create`



(b) Reaction rule `delete`

Fig. 14. Node behaviour rules for BRS

negative application condition (cf. Definition 2) to avoid creating duplicate links between a pair of nodes. We show these graphically as struck-out edges to indicate that an edge does not exist. Create Link also requires a *side-condition* that applies the rule only if the node attributes, *i.e.* x and y, ensure the signal ranges of the nodes overlap (via a disk assumption with constant radius `comm_radius`). The physical topology is modelled in the *side-conditions* because the lack of built-in notation of physical topology in GTS makes it difficult to encode directly.

Delete Link (Fig. 13b) performs the opposite operation to Create Link, removing a link between two nodes if one exists. By not explicitly specifying the link type, this rule applies to any edge classification.

Node behaviour in BRS is given by the reaction rules in Fig. 14. `create` (Fig. 14a) models explicitly (through sharing) that an unclassified link is established whenever communication between two nodes is physically possible, *i.e.* when two nodes are within each others signal range, in this case when the nodes share a common parent (are in the same region). Like the GTS model, this avoids duplicate links as the signals between two nodes are no longer disjoint after an initial application.

The inverse operation, `delete` (Fig. 14b), models the nodes out of signal range causing their overlap to split. Here, we use the instatiation map indicated by blue dotted arrows. This rule can be applied to delete *any type of link* between two nodes as the contents of the two `E` entities, shown as small circles in the left-hand side, are discarded by the instantiation map in the right-hand side.

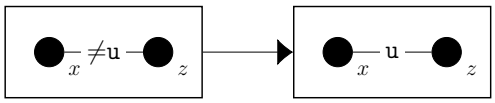


Fig. 15. UNCLASSIFY LINK

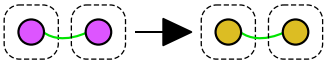


Fig. 16. `unclass_I`

Discussion 2 While the GTS model provides simpler methods for modelling node behaviour, *e.g.* negative conditions to handle duplicate links, it relies on side-conditions, based on entity attributes, to determine when rules can be applied, which requires a more complex theory, *e.g.* one that includes mathematical operations on attributes. The BRS model instead uses the built-in notion of locality to determine when links may be created, but requires more complex and less intuitive rules, *e.g.* instatiation maps to handle deletion.

In this case, the reaction rules for BRS are less intuitive than those for GTS, due to the complex diagrams required to indicate sharing.

From a pragmatic point of view, the use of sites as *placeholders* in BRS rules increases the applicability of a rule by allowing matching of structures whose exact content is not known *a priori*. In GTS, handling unknown structures often involves using additional control units or adding attributes into the model, and the operation may consist of several steps.

4.1.2. Link Behaviour

The GTS rule UNCLASSIFY LINK is shown in Fig. 15. This cannot be specified straightforwardly as a rule that allows *any* edge type in the left-hand side also matches links that are already unclassified, hence introducing unwanted identity transformations (*i.e.* rule applications which leave the graph unchanged). Instead, we use a side-condition, based on the link *attributes*, to specify the link status must not be already unclassified, allowing a single-rule specification.

An additional approach is to specify two distinct rules for unclassifying active and inactive edges, respectively. The BRS model adopts this same approach by introducing two distinct rules, `unclass_A` and `unclass_I`, as shown in Figs. 9 and 16.

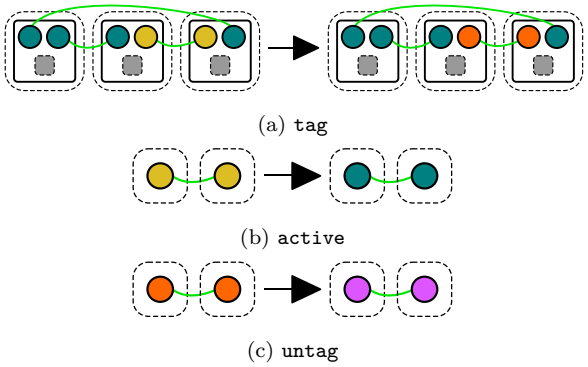
Discussion 3 In this case, both the GTS and BRS are similar (and could be written the same with multiple GTS rules), requiring small changes to link attributes/entity types. By using a variable condition, the GTS rule is open to extension, *e.g.* if a new link classification is added, while the BRS requires adding additional (yet trivial) rules.

4.1.3. Topology Control

Topology control revises the status of unclassified links to reduce the number of active links required. In GTS, topology control consists of the INACT rule shown in Fig. 3 that inactivates a link if another path between a triple of nodes exists. This rule only specifies the inactivation case and a complete topology control specification also contains requires a counterpart rule ACT, which turns unclassified nodes to active if required. As activation relies on the *non-existence* of an alternative active 2-hop path, rule ACT involves a negative condition.

In BRS, the *Resolve Unclassified Link* operation implementing *Topology control* is modelled using *tagging*, as introduced in [CKSS14]. Diagrams for three reaction rules implementing topology control are in Fig. 17. Firstly, we tag with U' (small amber circle) all the redundant unclassified links. Then, `active` is applied to activate all the remaining U-links that were not tagged during the previous step. Finally, the tags are removed and types changed to I by applications of reaction rule `untag`. Sites within L-boxes allow for `tag` to be applied when nodes have more than two links.

As bigraph theory lacks the control units of GTS, we instead use priorities that ensure, for example, `active` may only apply if there are no remaining occurrences of the left hand side of `tag`. The reactions should fire in the following order: `untag` < `active` < `tag`.



(a) `tag`

(b) `active`

(c) `untag`

Fig. 17. BRS reaction rules for topology control

Discussion 4 The rules for topology control are similar in both cases, with the BRS model requiring one additional rule to handle untagging. The key difference between GTS and BRS is how they determine when a specific rule(s) should be applied.

GTS systems often rely on control units to control rule applications (*e.g.* Fig. 7). Although this approach has its merits in relating GTS-based modelling to widely known imperative techniques, adding external control units obstructs the declarative nature of a GTS specification. Furthermore, almost all of the semantics proposed so far for GTS with control units focus on input-output model pairs. Only very recently, have there been proposed theoretical approaches which consider GTS control units using established theoretical frameworks like process algebrae or Petri nets [KLS18, KCL18].

In BRS practice, structures needed to control rule application are typically encoded within the bigraph itself using some form of tagging. This potentially adds confusion to the model, *e.g.* by adding entities that do not exist in the system being modelled. BRS often couples these structures with *rule priorities*, with partial ordering being theoretically more founded than the *ad-hoc* imperative control units used in GTS.

Finally, GTS provides a direct, inherent technique to formulate *negative conditions* both on rule as well as control level; this is currently out-of-scope in BRS (but, again, might be encoded by help structures). Recent work by Tsigkanos *et al.* [TKG17] narrowed the gap between GTS and BRS in this area by extending reaction rules with positive/negative application conditions specified in a spatial logic for closure spaces, while Archibald *et al.* have added a limited form of application condition directly into the bigraph theory [ACS20].

5. Modelling WSN Overlay Networks

The overlay network (Section 2) provides a set of virtual links over the sensor nodes to detail and manage routing pathways. As both the underlay and overlay share the same set of nodes, the main additional entity required is virtual links.

Ideally we should be able to treat the overlay and underlay networks as two separate models, allowing them to be verified both separately and together. This highlights the issue of whether the modelling frameworks provide a means for creating *modules*, *i.e.* subsets of rules/entities that provide a separation of concerns. Unfortunately, modules do not exist in either GTS and BRS, highlighting a future research direction. One promising approach in BRS is the use of multi-perspective modelling [BCRS16, SKCM18], where distinct bigraph regions are used to separate modelling concerns, *e.g.* into an underlay and overlay perspective. It remains unclear however how the multi-perspective approach should best maintain a single node set over multiple regions. We do not attempt a multi-perspective approach here, and instead opt to extend the existing model by adding additional entities and rules without making distinction between overlay and underlay entities.

Figure 18 shows the full GTS type graph containing the entities for both the underlay and overlay networks. Two new entities are added: OverlayLink that models a virtual link between two nodes, and OverlayNetwork that operates as a wrapper to manage the virtual link set. An interesting feature, available only in GTS, is the use of *inheritance* to allow a Link between two nodes to be either an UnderlayLink or an OverlayLink.

Table 2 shows the full set of entities for the BRS model. The main entity added is the virtual link V. As

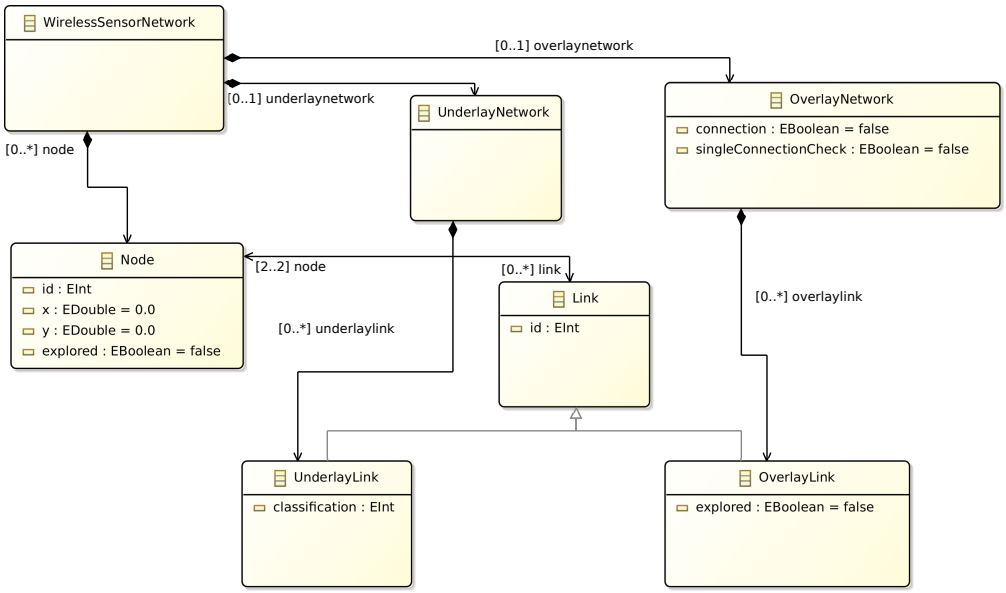


Fig. 18. GTS (Henshin) full type graph showing entities and their relationships

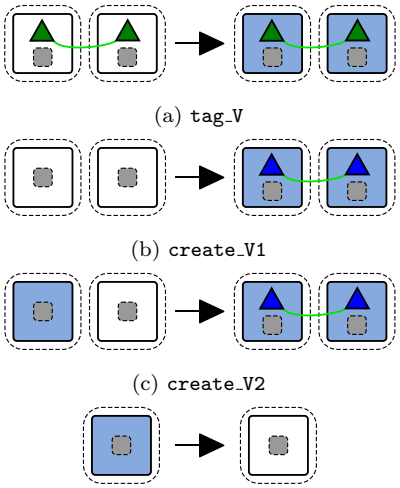
Table 2. Full set of entities for the BRS model

Description	Type	Arity	Atomic	Parent	Notation
		Underlay			
Signal range	S	1			oval
Node	N	1		S	circle
Link	L	0		N	rounded box
End (any type)	E	1		L	small circle
Active end	E.A	1	✓	L	small teal circle
Inactive end	E.I	1	✓	L	small purple circle
Unclassified end	E.U	1	✓	L	small yellow circle
Tagged end	E.U′	1	✓	L	small amber circle
		Overlay			
Virtual Link	V	1		L	triangle
Active Virtual Link	V.A	1	✓	L	green triangle
Tagged Virtual Link	V.A′	1	✓	L	blue triangle
Error Virtual Link	V.Err	1	✓	L	red triangle

with the link-end (E) entities in the underlay network, we use additional entities to tag the status of a virtual link, *e.g.* an error tag if there is no valid `a-u` underlay path meeting the virtual link requirement.

Discussion 5 As with the underlay network, GTS requires fewer additional entities (2 instead of 4) than BRS to represent the overlay network. Again, this is mainly due to additional *tagged* entities that are required to implement control sequences in BRS.

GTS makes use of *inheritance* in order to allow both underlay and overlay links to work in rules that match on abstract Links. Inheritance is not directly supported in BRS, however it can be mimicked through nesting. For example, we may define an entity Link that contains either a Underlay or Overlay entity. This allows rules that match on any link type, while still allowing rules that only work on specific types, *e.g.* by matching on Link.Overlay.



(a) `tag_V`

(b) `create_V1`

(c) `create_V2`

(d) `untag_V`

Fig. 19. BRS reaction rules to create virtual links

5.1. Dynamics

We specify two dynamic behaviours for the overlay network: *Requirement management* that allows a user to create/delete a virtual link between sensors, and *Routing maintenance* that attempts to ensure there is a valid underlay path for each virtual link.

5.1.1. Requirement Management

In the GTS model, the rules to create/delete a virtual link are similar to those that create an underlay link (cf. Fig. 13). They differ by no longer requiring a side condition to detect when the sensors are *close-enough* to communicate as in the overlay network *any* pair of sensors may be (virtually) connected/disconnected.

Due to the lack of wireless signals in the overlay network, it is not possible for the BRS to reuse rules similar to those for creating underlay links as we risk creating duplicate virtual links between pairs of sensors. As the BRS lacks negative application conditions, we avoid duplicating links through a multi-step tagging technique show in Fig. 19. Priorities enforce the order of rule application: first, sensors already linked by an active virtual link are tagged using `tag_V`. Second, `create_V1` or `create_V2` creates a virtual link between the required sensors if at least one sensor is not tagged. Finally, all tags are removed using `untag_V`. Notice that `tag_V` and `untag_V` are applied to *all* matches, while `create_V1`/`create_V2` is applied to a particular pair of nodes.

The BRS *Delete Virtual Link* operation is modelled by the simple reaction rule of Fig. 20. The use of sites within the virtual links allows the rule to be applied regardless of the status of the virtual edge, *e.g.* the same rule can remove virtual links that are active or in an error state.

Discussion 6 The lack of negative application conditions to detect duplicate links in BRS is costly, requiring 4 rules to create a virtual link as opposed to 1 in GTS. An alternative approach using 2 rules would be to allow the creation of duplicate links between sensors, with a second rule to garbage collect any duplicate links, however this can leave the model in an inconsistent state should other rules be applied before garbage collection.

In both cases the deletion of a link is simple and requires only a single rule. Even with negative application conditions in GTS, it is often much easier, in both GTS and BRS, to match on the *existence* of an entity/relationship rather than the non-existence.

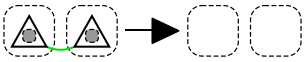


Fig. 20. `delete_V`

5.1.2. Routing Maintenance

Routing maintenance requires interaction between the overlay and underlay networks to ensure the requirements, as specified by virtual links, are met (if the given physical sensor layout allows). Routing maintenance poses a challenge for both GTS and BRS, as a full-fledged search algorithm has to be encoded in transformation/reaction rules. We sketch how to encode path searches the following, in particular how we implement the *Search Active Path* functionality. Activating `a-u` paths, for example, uses a similar search but allowing for `u` links to be included. Complete implementations of all search functionality is found in the reference models [10].

In GTS, we associate an *explored* attribute to each sensor that determines if the sensor has been visited in the current run of the path search algorithm. This is similar to the tagging approach seen in the BRS underlay model, however here we use *attributes* instead of defining tagged variants of entities.

We sketch the search algorithm here, using the rules shown in Fig. 21. In practice, the GTS model implements the Fig. 21e control unit as a combination of 4 control units. The search algorithm is parameterised by the start and end sensors, *i.e.* those participating in the virtual link we wish to check. This requires each sensor in the system to be assigned a unique `id`, in this case an entity attribute that is set when the initial topology is created.

The search is guided using a control unit like the one shown in Fig. 21e. Search begins by setting every sensor in the system to *unexplored* using rule UNSETEXPLORED (Fig. 21a). Then, starting from the start sensor, we set the sensor as *explored* using SETEXPLORED then search, in a depth-first fashion, every *unexplored* node reachable through an `a`-path. Reachable nodes are determined by rule GETNEXT (Fig. 21c) that returns the `id` of an unexplored neighbour. This `id` is then passed to SETEXPLORED and the algorithm loops. If GETNEXT fails to match, *i.e.* when there are no unexplored neighbours remaining, the search backtracks and the next unexplored neighbour is explored. Once all reachable sensors have been tagged, we check, using CHECKEXPLORED (Fig. 21d), if the end sensor has been explored. If so, we know there is a valid `a`-path from start to end.

Like GTS, the BRS approach makes use of tagging to perform path search as shown by the 5 reaction rules in Fig. 22. Initially, rule `source` chooses one end of a virtual link ($\mathsf{V}_x.\mathsf{A}'$) and tags it as the source of the path (F). Afterwards, `path0` tags all sensors reachable by following active underlay links ($\mathsf{E}_x.\mathsf{A}$) from the source. At this point, *if* `active_V` can be applied, the tentative virtual link is promoted to active. Otherwise, `path` is applied again to explore additional `a`-paths. Finally, when `active_V` and `path` are no longer applicable, *i.e.* there are no paths remaining from the source, the tentative link is tagged as $\mathsf{V}_x.\mathsf{Err}$ to indicate the virtual link cannot be established given the current network configuration. Rule priorities are specified as

$$\texttt{source} < \texttt{error} < \texttt{path} < \texttt{active\_V} < \texttt{path0}$$

Placing `source` at the lowest priority ensures that the entire search algorithm runs to completion for a single source node before the another source node is marked.

In both cases, the *Activate* `a-u` *Path* requirement can be modelled by performing a similar search that allows `a-u` paths, and performing an additional step to *activate* any unclassified nodes that are discovered. *Mark Inactive Path* is specified by an analogous extension in which inactive links are identified and iteratively unclassified by the *Unclassify* operation.

Discussion 7 Both GTS and BRS make use of a similar tag based approach to performing search. Such a method is essential to ensure sensors are not visited more than once, and similar approaches are used in standard graph search algorithms.

GTS requires heavy use of control units to implement search (*e.g.* 4 for SEARCHACTIVEPATH). As control units are not part of the core GTS theory, they may prove problematic when expressing complex algorithms. However, rule-based rewriting is often not adequate for such tasks, *e.g.* mimicking a recursive behaviour is

[10] GTS: `https://github.com/timofr/sosywsn/tree/master/modeling/henshin/WSN_Henshin`; BRS: `http://www.dcs.gla.ac.uk/~michele/wsn.big`

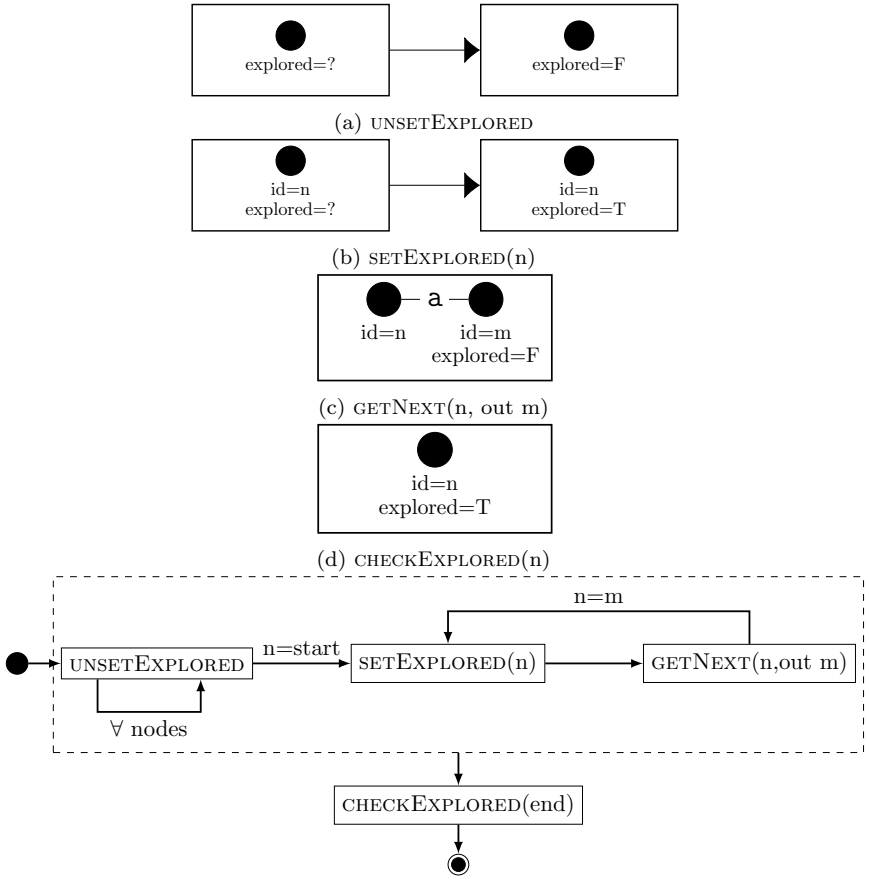


Fig. 21. GTS rules and control unit for searching an active path

cumbersome and error-prone using GTS. While the BRS approach of rule priorities removes the need for imperative control units, it relies on adding algorithmic entities, *e.g.* tags, to the model resulting in larger, and possibly more complex, models.

6. Analysis of WSN models

We give a flavour of model analysis for GTS and BRS here, focusing on verifying properties of both the underlay and overlay networks. We conclude this section with a detailed discussion of the concurrent behaviour of the system, *e.g.* the *interactions* between the underlay and overlay network.

6.1. Instance Correctness

Checking reachability and correctness properties rely on generating a transition system from a given starting topology. This requires the generation of *correct* starting instances.

Correctness for GTS requires all graphs to be an instantiation of the meta-model (type-graph), that is, to be well typed. Automated checking that a graph conforms to a given meta-model is available in the major tools for GTS. While it is possible to check an whether instance is well-typed, methods for automatically generating well-typed instances, perhaps with additional properties such as bounds on the number of nodes and connectivity, remains an active research area [Tae12, SBL$^+$20]. It is further complicated if, for example, we also need to generate valid and meaningful entity attributes as well as structure.

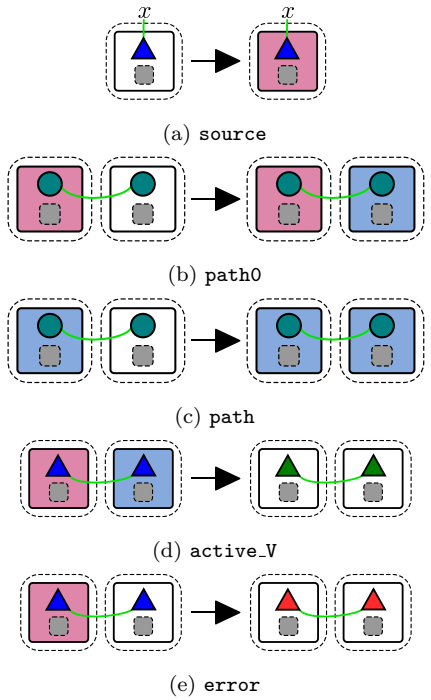


Fig. 22. Diagrams for reaction rules `source` (a), `path0` (b), `path` (c), `active_V` (d), and `error` (e)

Correctness of BRS models may be checked by providing a *sorting scheme* [Mil09, Chapter 6]. Intuitively a sorting scheme assigns a *sort* to each entity type, and a set of constraints on these sorts. For example, we may specify that a sensor node *cannot* contain another sensor node. Generating correct test instances then corresponds to generating a well sorted bigraph. Currently no tool provides support for working with sorting schemes, including checking a bigraph matches a given scheme, and instances must be verified by hand. Creating such a tool is a fruitful area for future work. One practical approach is to construct instances from a set of reaction rules, *e.g.* implementing `newNode` functionality. We then prove, again currently by hand, that the reaction rules only produce well sorted bigraphs. Then, by constructing test instances through the reaction rules we are guaranteed to obtain a well-sorted bigraph. This approach is based on the idea that is it usually simpler to show a rule is well sorted as opposed to an arbitrary bigraph.

6.2. Inductive Reasoning

By utilising a formalism with a strong mathematical foundation, we gain the ability to reason about resulting models using mathematical proof techniques. In particular, we may make use of *inductive reasoning* to prove that properties hold in all cases, *e.g.* regardless of the number of nodes and starting topology.

The idea behind inductive reasoning is to prove that there is no sequence of transformation/reaction rules that cause the system to enter an invalid state. As with mathematical induction, we first reason about the system in some base state, by checking that transformation/reaction rules preserve some required property. We then lift this to arbitrary models by an inductive step (that utilises the base case).

The induction need not be over all rules, for example, for the *Connectedness* property below we assume the rule to delete a link is omitted.

Rule ordering is important in inductive reasoning and we must ensure, for example, that the application of a rule does not enable another rule, that was not previously applicable, to put the system in an invalid state.

Inductive reasoning is complicated, but not impossible (*e.g.* [DG15, Pen09]), for GTS due to the use of control units that allow complex rule application orders. For BRS, rule priorities are simpler to reason over, and inductive reasoning has been previously applied [BCRS16]. None of major tools for GTS or BRS support

inductive reasoning automatically. As with tooling for automated sort checking, we believe automating inductive reasoning to be a key area for future work.

6.3. State Space Analysis

Inductive reasoning requires careful analysis of the models and is not suitable for non-expert users. Instead both GTS and BRS models can be analysed through state space exploration. Here, starting with an initial (bi)graph, rewrites are repeatedly applied until a labelled transition system (LTS) is created. In this case we have predicate labels on states and rule names on transitions. In BRS theory there are techniques, from the process algebra community, to allow *action* labels on transitions [Mil09, Chapter 6.]. Model checking techniques, often combined with temporal logic, then specify and check properties we expect to hold on the transition system. Full state space analysis is not possible for all systems, *e.g.* they may be infinite, but is possible for many practical cases [BDK+12].

Regarding tool support, state space generation is a built-in feature of Henshin, while Groove natively supports model checking of temporal logic formulas. BigraphER provides tools to export a transition system for use external model checking tools, *e.g.* PRISM [KNP11]. Separating model generation (through rewrite rules) from model checking may lead to some inefficiencies in analysis, *e.g.* it is not possible to reduce the search space through symbolic analysis on the original model, however rewrite based approaches have several benefits over component-based (such as those found in PRISM) including "changeability", "expressiveness", and "simplicity" [KG12].

6.4. Underlay Properties

As described in Section 2.4, we consider three properties of the underlay network: *Connectedness*, *Redundancy Reduction*, and *Liveness Resolution*.

Connectedness Connectedness is the property that, given a connected start state, *i.e.* one where there is at least one `a-u` *path* between each pair of nodes, on any execution where no underlay link is deleted, all reachable states remain connected.

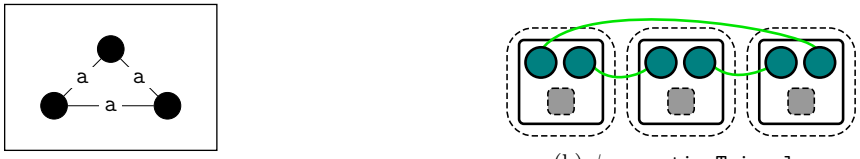
Connectedness is difficult to prove for both GTS and BRS models (and models in general) as it requires universal quantification over the whole model, *i.e. forall* pairs of nodes rather than requiring a single match. Using model checking here is difficult, often requiring external algorithms to perform a search of the whole topology at each new state.

Inductive reasoning allows universal properties to be checked. A sketch of this inductive reasoning for BRS is as follows. The base case is a topology consisting of two nodes. As the start state is connected, there must be either an `a` or `u` link between these two nodes. As we assume no underlay link is deleted, the only rule that can change the link status to `i` is `untag` (Fig. 17c). Links may only be tagged using the `tag` rule (Fig. 17a) which requires three nodes, hence the two node topology is always connected. For an n node topology we may have triangles of nodes and *at most one* link in the triangle might be inactive if `tag`/`untag` have been applied. The other two links must be `a-u` links and, given the base case, these are always connected. Therefore the whole topology remains connected. Inductive reasoning for GTS takes a similar form.

Redundancy Reduction Redundancy Reduction is the property that the system reduces the number of active links to the minimum required by removing active triangles. That is, in a topology with no unclassified links there will be no active triangles.

For GTS we can formulate the Redundancy Reduction property over a single forbidden structure *i.e.* a triangle of active links. The non-existence of this structure is then verified using a rule with a negative application condition. Alternatively, this structure can be used in an identity transformation rule that matches the active triangle structure but performs no changes to the graph. This causes the rule to be labelled in the resulting transition system allowing it to be checked by a model checker. We use the second form here as it is closer to how the BRS detects the potential for redundancy reduction.

Similarly to GTS, in BRS we may define predicates as bigraphs [BCRS16] that are checked, via matching, at each step of generating the transition system, and appear as state labels allowing for verification via a model checker.



(a) ϕ_1 = ACTIVETRIANGLE (b) ϕ_1 = `activeTriangle`

Fig. 23. Redundancy reduction predicates to detect active triangles

(a) ϕ_2 = UNCLASSIFIED (b) ϕ_2 = `unclassified`

Fig. 24. Liveness predicates to detect unclassified links

The predicates, ϕ_1 for both GTS and BRS are given in Fig. 23. Both predicates are simple and detect when an active triangle exists between three nodes in the system. If this is the case, then there is the potential for further redundancy reduction that has not taken place.

We may then specify, as an LTL formula that, assuming the network does not change, *i.e.* no new undefined links are created, we eventually reach a state with no active triangles: $\mathbf{F}(\neg\phi_1)$.

Liveness Resolution Liveness Resolution expresses that the system, given no operations that add unclassified links, reaches a state containing no unclassified links *i.e.* links are either assigned active or inactive.

As with Redundancy Reduction we can specify a predicate, ϕ_2, shown in Fig. 24, that match when an unclassified link exists in the system. We can then check that eventually all unclassified links are assigned a state using the LTL formula $\mathbf{F}(\neg\phi_2)$.

6.5. Overlay Properties

We are interested in a single overlay network property: Virtual Link Fulfilment. This property specifies that, for virtual links with no `a-u` path, a `a-u` path will eventually be created, *i.e.* topology control works correctly. Note that it is not always possible to create such a link, for example, when we have a virtual link to an isolated node with no underlay links.

The topology search algorithms have features to determine if a virtual link is unfulfilled, *i.e.* Err tagged links. To check the Virtual Link Fulfilment property, we can check simple predicates based around this existing functionality as shown in Fig. 25. As the GTS model requires parameters to identify the particular nodes in the predicate there is essentially one predicate for each start/end pair that have a virtual link.

Importantly, the virtual link fulfilment predicates are only valid after the current round of the search algorithm has run, *i.e.* when the models have not reset the tags. Therefore, we do not know the state of the predicate *between* search runs. As we only wish to check that *eventually* all links are fulfilled this is not an issue for verification.

Again, the LTL formula states that eventually we should not have any unfulfilled links if topology control works correctly: $\mathbf{F}(\neg\phi_3)$

(a) ϕ_3 = UNFULFILLED(start,end) (b) ϕ_3 = `unfulfilled`

Fig. 25. Virtual Link Fulfillment Predicates

6.6. Complex and Quantitative Properties

The model checking approach is not limited to simple reachability properties such as those given above. For example, we may specify that topology control never introduces new active triangles as it runs:

$$\phi_2 \rightarrow (\neg\phi_1 \mathbf{U} \neg\phi_2)$$

This states that whenever topology control is needed, *i.e.* we have an unclassified link, we should never obtain an active triangle until topology control is complete, *i.e.* all links are classified.

In general, we are not only interested in qualitative properties, *e.g.* that there are no active triangles, but may also be interested in quantitative properties such as the time it takes topology control to complete or how the system operates under uncertainty, *e.g.* the probability of a node failure. GTS were be extended with probabilistic and real-time support [MGK18], while stochastic and probabilistic models (but not real-time models) are supported for BRS [KMT08] in BigraphER. From a practical modelling perspective, adding rates/probabilities/timing amounts to adding conditions to the rewrite rules, *e.g.* $l \rightarrow r$ with probability p. Such probabilistic and timed models have been previously used in wireless sensor protocol design [KMH08].

6.7. Concurrency Analysis

The co-existence of different events, such as an underlay link being deleted due to node movement while topology control is running, naturally results in concurrent behaviour.

Conflicts occur when the execution of an rule obstructs another rule. For example, in our model, an operation deleting an unclassified link conflicts with any action changing the status of the same link. Such a conflict is asymmetric as link deletion can still be performed after a status change but not the other way around.

The three underlay components (node behaviour, link behaviour, topology control) are naturally concurrent. In GTS, the arising conflicts can be avoided using control units and negative application conditions. To statically detect potential conflicts, *critical pair analysis* (CPA) theory has been proposed for GTS [EEPT06], with ongoing theoretical research [BLST17] and tool support as in Henshin and AGG. However, the CPA techniques are based solely on the transformation rules and do not account for the use of control units. Given the common use of control units in GTS, CPA often results in false positives, reducing the benefit to the modeller. Recent work proposes the first notions for an operational, thus, control-aware interpretation of independence and conflicts [KCL18].

In BRS, tagging schemes and rule priorities are used to avoid conflicts between rules. To aid in concurrency analysis, BigraphER is able to detect undesirable states at run-time, whose analysis might help in conflict identification and subsequent model refinement. Likewise, BigraphER supports filtering of intermediary states that are used only for control allowing, for example, the recursive topology control algorithm to be treated as a single step action.

Explicit Synchronisation, allows two rules to run at the same time, *i.e.* in a single transition step. In GTS there are no established techniques to compose independent rule applications into synchronised actions. There have been proposed various rule composition techniques for GTS, most notably *amalgamation* [RK09], however these techniques do not explicitly address a *concurrent* scenario.

Explicit synchronisation is not specified in BRS. In general, when both $R \longrightarrow\!\!\blacktriangleright R'$ and $P \longrightarrow\!\!\blacktriangleright P'$ are applicable, only one rule application can take place at one evaluation step. Explicit synchronisation can be expressed by adding reaction rule $R \parallel P \longrightarrow\!\!\blacktriangleright R' \parallel P'$ to the BRS; assuming R and P do not overlap.

Concurrency analysis in both GTS and BRS is a promising area for future research. The different philosophy and approaches to control/concurrency is a crucial discrepancies between GTS and BRS. This is due to their theoretical origins: GTS generalises term rewriting [Roz97] and originates in classical grammars where operational and concurrency issues are out-of-scope, while BRS are a generalisation of process calculi, having operational semantics and concurrency at their core.

7. Comparison of GTS and BRS

To guide future application modelling, to aid in selecting between GTS and BRS, and to highlight areas we think fruitful for future investigation, we have discussed the similarities and differences of GTS and BRS

Table 3. Summary comparison of features of GTS and BRS

Feature	**GTS**	**BRS**
Categorical Object	Graphs	Interfaces
Categorical Arrows	(Partial) Graph Morphisms	Bigraphs
Matching Semantics	3 Step gluing	Sub-graph rewriting
Locality	×	✓
Hierarchical Entities	×	✓
Hyperedges	×	✓
Entity Attributes	✓	Via nesting
Explicit Abstraction	×	✓
Inheritance	✓	×
Instantiation maps	×	✓
Negative Application Conditions	✓	×
Control Flow	Imperative Control Units	Priorities/Tagging
State Space Analysis	✓	✓
Inductive Reasoning	✓	✓
Tool Support	Henshin, Groove, AGG, eMoflon	BigraphER, BigRed
Meta-Models	Type Graphs	Sorting

throughout. This section brings together these discussions and highlights areas of future work. As BRS is a newer formalism than GTS we expect most ideas to flow GTS to BRS. A summary of the different features is in Table 3.

While both based on graph structures the formalisms are theoretically quite different. GTS tends to use traditional DAG structures with nodes representing entities and *binary* links representing relationships. On the other hand, BRS have two overlaid graph structures: a DAG[11] with binary links representing containment and a hypergraph allowing multi-arity links. As we have seen (Section 4) fixed arity constraints on bigraph entities are often restrictive when modelling, with additional place graph nodes operating as link ends often used to overcome this. An interesting research question is whether bigraphs themselves could be redefined, and what you gain/lose, by disallowing hyperedges in the link graph and instead requiring explicit entities in *all* cases – bringing it closer to a GTS-style setup with essentially (ignoring interfaces) *coloured* links to represent place vs linking dimensions.

GTS provides additional features that augment the graph structures. For example, they allow *Entity Attributes*, *e.g.* a entity contains an integer x, with languages that allow querying and updating. Likewise *Imperative Control Units* augment the rewriting system to provide fine-grained, and possibly conditional, control. BRS have less features in this regard: attributes can be described by nesting additional controls, however as features to query and update the attributes must be provided directly through reaction rules it is difficult to perform calculations without encoding a programming language within the bigraph itself[12]. Likewise, control flow is limited to priorities only, with all other control flow being pushed into the model, *e.g.* through tagging. We believe BRS can benefit from making it easier to specify attributed and control flow and could borrow many of these aspects from the GTS theory. One promising area is to allow a *strategy language*, like those found in term rewriting [MMV04], to specify imperative control flow while sticking to the core rewriting theory of bigraphs.

One of the key features lacking from BRS are (negative) application conditions that significantly increase the expressiveness of GTS. Adding this feature to BRS is challenging due to the differences in how the systems are constructed *i.e.* bigraphs are formed compositionally requiring careful management of symmetries/interface renaming etc., while GTS treats graphs as objects with (partial) maps between graphs. In order to reuse techniques, a promising direction is to learn from pushing BRS closer to GTS *e.g.* [Ehr02, SS04], while also pushing GTS closer to compositionally defined structures, *e.g.* [DJ19]. While we expect interesting results from such work, we are not advocating that GTS should *become* BRS (or vice-versa). As we have shown throughout this paper each to excels in different aspects of modelling.

Although not explored much in this paper, the compositional nature of bigraphs allows them to be constructed algebraically as well as diagrammatically. The algebra forms the basis of the BigraphER programming language allowing users to specify large models (that are difficult to draw) and to benefit from

[11] Assuming sharing.

[12] The BigraphER implementation makes this easier by allowing rules to be generated for specific sets of inputs, *i.e.* to generate all rules that match (entities) x and y and rewrite to (entity) $x + y$.

reuse of expressions. Likewise we envisage the same algebraic language being usable by multiple independent tools. GTS on the other hand tend to be largely described diagrammatically.

GTS and BRS support similar features for model analysis, in particular, both make heavy use of reachability analysis, *e.g.* to allow model checking. Concurrency analysis, *e.g.* confluence/critical pairs, is a large research topic in GTS that is under-explored for BRS with no general theory or tool support available. This is perhaps surprising given the background of BRS is the process algebra/concurrency communities. Exploring this further fruitful area for future research and we expect BRS can learn significantly from the GTS, term rewriting, and concurrency communities.

Finally, while we focused on GTS and BRS there remains interesting future work to compare a wider range of modelling formalisms, *e.g.* process algebra, from a practical modelling standpoint: to highlight interesting cases and allow modelling formalisms to further learn from and guide each other.

8. Conclusions

By modelling topology control in wireless sensor networks using both graph transformation systems (GTS) and bigraphical reactive systems (BRS), we have gained a deeper understanding of the practical modelling abilities of both formalisms.

To show the usefulness of formal models, we described how analysis can be performed in both formalisms. State space analysis, where the model is analysed as a labelled transition system – with states are represented as (bi-)graphs, and transitions as (bi-)graph transforms/rewrites – is common for both GTS and BRS. Labelled transition systems work well with existing model checking tools that allow temporal properties to be checked. While this is appropriate for simple predicates, *e.g.* such as checking the system eventually converges to a particular state, it cannot handle properties with universal quantification over the whole model, *e.g.* a property holds for all pairs of sensor nodes. Approaches, for both GTS and BRS, to handle universally quantified predicates is an area of active research. A promising approach is inductive reasoning, based on the rewrite rules, to prove no sequence can produce a failure state. Currently no tools support automated inductive reasoning. Concurrency analysis is currently stronger for BRS due to the process calculi background, yet many techniques are being considered for GTS. This remains an area of future research.

While we have only shown GTS and BRS applied to a single application area, as systems continue to be larger, more complex, and operate in uncertain environments, the need for intuitive modelling tools becomes greater. GTS and BRS both have the ability to express complex relationships between entities without sacrificing on the readability of the models, and this paper motivates their future use in WSNs, networking, and other domains.

Acknowledgements

This work is supported by the Engineering and Physical Sciences Research Council, under grants EP/N007565/1 (S4: Science of Sensor Systems Software); PETRAS SRF grant MAGIC (EP/S035362/1), and by the German Research Foundation (DFG) as part of project A1 within CRC 1053–MAKI.

References

[ABJ+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced concepts and tools for in-place EMF model transformations. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2010.

[ACS20] Blair Archibald, Muffy Calder, and Michele Sevegnani. Conditional bigraphs. In Fabio Gadducci and Timo Kehrer, editors, *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*, volume 12150 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2020.

[APN19] Faeq Alrimawi, Liliana Pasquale, and Bashar Nuseibeh. On the automated management of security incidents in smart spaces. *IEEE Access*, 7:111513–111527, 2019.

[ASH+20] Blair Archibald, Min-Zheng Shieh, Yu-Hsuan Hu, Michele Sevegnani, and Yi-Bing Lin. Bigraphtalk: Verified design of iot applications. *IEEE Internet Things J.*, 7(4):2955–2967, 2020.

[BBKW89] Jos C. M. Baeten, Jan A. Bergstra, Jan Willem Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *Theor. Comput. Sci.*, 67(2&3):283–301, 1989.

[BCRS16] Steve Benford, Muffy Calder, Tom Rodden, and Michele Sevegnani. On lions, impala, and bigraphs: Modelling interactions in physical/virtual spaces. *ACM Trans. Comput.-Hum. Interact.*, 23(2):9:1–9:56, 2016.

[BDK+12] Nathalie Bertrand, Giorgio Delzanno, Barbara König, Arnaud Sangnier, and Jan Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPIcs*, pages 101–116. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[BLST17] Kristopher Born, Leen Lambers, Daniel Strüber, and Gabriele Taentzer. Granularity of conflicts and dependencies in graph transformation systems. In *Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*, pages 125–141, 2017.

[BTM18] Michael J. Breza, Ivana Tomic, and Julie A. McCann. Failures from the environment, a report on the first FAILSAFE workshop. *Computer Communication Review*, 48(2):40–45, 2018.

[CCC+18] Muffy Calder, Claire Craig, Dave Culley, Richard de Cani, Christl A. Donnelly, Rowan Douglas, Bruce Edmonds, Jonathon Gascoigne, Nigel Gilbert, Caroline Hargrove, Derwen Hinds, David C. Lane, David Mitchell, Giles Pavey, David Robertson, Bridget Rosewell, Spencer J. Sherwin, Mark J. Walport, and A. Wilson. Computational modelling for decision-making: where, why, what, who and how. In *Royal Society open science*, 2018.

[CKSS14] Muffy Calder, Alexandros Koliousis, Michele Sevegnani, and Joseph S. Sventek. Real-time verification of wireless home networks using bigraphs with sharing. *Sci. Comput. Program.*, 80:288–310, 2014.

[CMR96] Andrea Corradini, Ugo Montanari, and Francesca Rossi. Graph processes. *Fundam. Inform.*, 26(3/4):241–265, 1996.

[CS14] Muffy Calder and Michele Sevegnani. Modelling IEEE 802.11 CSMA/CA RTS/CTS with stochastic bigraphs with sharing. *Formal Asp. Comput.*, 26(3):537–561, 2014.

[DG15] Johannes Dyck and Holger Giese. Inductive invariant checking with partial negative application conditions. In Francesco Parisi-Presicce and Bernhard Westfechtel, editors, *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*, volume 9151 of *Lecture Notes in Computer Science*, pages 237–253. Springer, 2015.

[DJ19] Nachum Dershowitz and Jean-Pierre Jouannaud. Drags: A compositional algebraic framework for graph rewriting. *Theor. Comput. Sci.*, 777:204–231, 2019.

[DRM14] Antônio Vicente Lourenço Dâmaso, Nelson Souto Rosa, and Paulo Romero Martins Maciel. Using coloured petri nets for evaluating the power consumption of wireless sensor networks. *Int. J. Distributed Sens. Networks*, 10, 2014.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[EHK+97] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. Algebraic approaches to graph transformation - part II: single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 247–312. 1997.

[Ehr02] Hartmut Ehrig. Bigraphs meet double pushouts. *Bulletin of the EATCS*, 78:72–85, 2002.

[FvGH+12] Ansgar Fehnker, Rob J. van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. A process algebra for wireless mesh networks. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012,*

Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings, volume 7211 of *Lecture Notes in Computer Science*, pages 295–315. Springer, 2012.

[GdLW+13] Esther Guerra, Juan de Lara, Manuel Wimmer, Gerti Kappel, Angelika Kusel, Werner Retschitzegger, Johannes Schönböck, and Wieland Schwinger. Automated verification of model transformations based on visual contracts. *Autom. Softw. Eng.*, 20(1):5–46, 2013.

[GdMR+12] Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using GROOVE. *STTT*, 14(1):15–40, 2012.

[GRJD19] Amal Gassara, Ismael Bouassida Rodriguez, Mohamed Jmaiel, and Khalil Drira. Executing bigraphical reactive systems. *Discrete Applied Mathematics*, 253:73–92, 2019.

[Hec06] Reiko Heckel. Graph transformation in a nutshell. *Electron. Notes Theor. Comput. Sci.*, 148(1):187–198, 2006.

[KCL18] Géza Kulcsár, Andrea Corradini, and Malte Lochau. Equivalence and independence in controlled graph-rewriting processes. In *Graph Transformation - 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25-26, 2018, Proceedings*, pages 134–151, 2018.

[KG12] Christian Krause and Holger Giese. Probabilistic graph transformation systems. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph Transformations - 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29, 2012. Proceedings*, volume 7562 of *Lecture Notes in Computer Science*, pages 311–325. Springer, 2012.

[KLS18] Géza Kulcsár, Malte Lochau, and Andy Schürr. Graph-rewriting petri nets. In *Graph Transformation - 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25-26, 2018, Proceedings*, pages 79–96, 2018.

[KMH08] Michael Katelman, José Meseguer, and Jennifer C. Hou. Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In *Formal Methods for Open Object-Based Distributed Systems, 10th IFIP WG 6.1 International Conference, FMOODS 2008, Oslo, Norway, June 4-6, 2008, Proceedings*, pages 150–169, 2008.

[KMT08] Jean Krivine, Robin Milner, and Angelo Troina. Stochastic bigraphs. *Electr. Notes Theor. Comput. Sci.*, 218:73–96, 2008.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 585–591, 2011.

[KNPR18] Barbara König, Dennis Nolte, Julia Padberg, and Arend Rensink. A tutorial on graph transformation. In Reiko Heckel and Gabriele Taentzer, editors, *Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig*, volume 10800 of *Lecture Notes in Computer Science*, pages 83–104. Springer, 2018.

[KR18] Hans-Jörg Kreowski and Grzegorz Rozenberg. Graph surfing by reaction systems. In *Graph Transformation - 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25-26, 2018, Proceedings*, pages 45–62, 2018.

[KSS+14] Géza Kulcsár, Michael Stein, Immanuel Schweizer, Gergely Varró, Max Mühlhäuser, and Andy Schürr. Rapid prototyping of topology control algorithms by graph transformation. *ECEASST*, 68, 2014.

[KSV+18] Roland Kluge, Michael Stein, Gergely Varró, Andy Schürr, Matthias Hollick, and Max Mühlhäuser. A systematic approach to constructing families of incremental topology control algorithms using graph transformation. *Software Engineering und Software Management 2018, Fachtagung des GI-Fachbereichs Softwaretechnik, SE 2018, 5.-9. März 2018, Ulm, Germany.*, pages 109–110, 2018.

[KVS15] Roland Kluge, Gergely Varró, and Andy Schürr. A methodology for designing dynamic topology control algorithms via graph transformation. In *Theory and Practice of Model Transformations - 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015. Proceedings*, pages 199–213, 2015.

[LAS14] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Developing emoflon with emoflon. In

Theory and Practice of Model Transformations - 7th International Conference, ICMT 2014, Held as Part of STAF 2014, York, UK, July 21-22, 2014. Proceedings, pages 138–145, 2014.

[LML08] Xiaofang Li, Yingchi Mao, and Yi Liang. A survey on topology control in wireless sensor networks. *10th International Conference on Control, Automation, Robotics and Vision, ICARCV 2008, Hanoi, Vietnam, 17-20 December 2008, Proceedings*, pages 251–255, 2008.

[LS10] Ivan Lanese and Davide Sangiorgi. An operational semantics for a calculus for wireless systems. *Theor. Comput. Sci.*, 411(19):1928–1948, 2010.

[LYW+19] Yahui Li, Xia Yin, Zhiliang Wang, Jiangyuan Yao, Xingang Shi, Jianping Wu, Han Zhang, and Qing Wang. A survey on network verification and testing with formal methods: Approaches and challenges. *IEEE Communications Surveys and Tutorials*, 21(1):940–969, 2019.

[MGK18] Maria Maximova, Holger Giese, and Christian Krause. Probabilistic timed graph transformation systems. *J. Log. Algebr. Meth. Program.*, 101:110–131, 2018.

[Mil09] Robin Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[MMV04] Narciso Martí-Oliet, José Meseguer, and Alberto Verdejo. Towards a strategy language for maude. In Narciso Martí-Oliet, editor, *Proceedings of the Fifth International Workshop on Rewriting Logic and Its Applications, WRLA 2004, Barcelona, Spain, March 27-28, 2004*, volume 117 of *Electronic Notes in Theoretical Computer Science*, pages 417–441. Elsevier, 2004.

[PDH13] Gian Perrone, Søren Debois, and Thomas T. Hildebrandt. A verification environment for bigraphs. *ISSE*, 9(2):95–104, 2013.

[Pen09] Karl-Heinz Pennemann. *Development of correct graph transformation systems*. PhD thesis, University of Oldenburg, Germany, 2009.

[PKLS16] Sven Peldszus, Géza Kulcsár, Malte Lochau, and Sandro Schulze. Continuous detection of design flaws in evolving object-oriented programs using incremental multi-pattern matching. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, pages 578–589, 2016.

[RK09] Arend Rensink and Jan-Hendrik Kuperus. Repotting the geraniums: On nested graph transformation rules. volume 18, 2009.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[San05] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, 2005.

[SBG+17] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaela Groner, Timo Kehrer, Manuel Ohrndorf, and Matthias Tichy. Henshin: A usability-focused framework for EMF model transformation development. In *Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*, pages 196–208, 2017.

[SBL+20] Oszkár Semeráth, Aren A. Babikian, Anqi Li, Kristóf Marussy, and Dániel Varró. Automated generation of consistent models with structural and attribute constraints. In Eugene Syriani, Houari A. Sahraoui, Juan de Lara, and Silvia Abrahão, editors, *MoDELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020*, pages 187–199. ACM, 2020.

[SC15] Michele Sevegnani and Muffy Calder. Bigraphs with sharing. *Theor. Comput. Sci.*, 577:43–73, 2015.

[SC16] Michele Sevegnani and Muffy Calder. BigraphER: Rewriting and analysis engine for bigraphs. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 494–501, 2016.

[SKCM18] Michele Sevegnani, Milan Kabác, Muffy Calder, and Julie A. McCann. Modelling and verification of large-scale sensor network infrastructures. In *23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018, Melbourne, Australia, December 12-14, 2018*, pages 71–81, 2018.

[SS04] Vladimiro Sassone and Pawel Sobocinski. Congruences for contextual graph-rewriting. *BRICS Report Series*, 11(11), 2004.

[Tae03] Gabriele Taentzer. AGG: A graph transformation environment for modeling and validation of software. In *Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, Charlottesville, VA, USA, September 27 - October 1, 2003, Revised Selected and Invited Papers*, pages 446–453, 2003.

[Tae12] Gabriele Taentzer. Instance generation from type graphs with arbitrary multiplicities. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 47, 2012.

[TKG17] Christos Tsigkanos, Timo Kehrer, and Carlo Ghezzi. Modeling and verification of evolving cyber-physical spaces. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 38–48, 2017.

[TPGN18] Christos Tsigkanos, Liliana Pasquale, Carlo Ghezzi, and Bashar Nuseibeh. On the interplay between cyber and physical spaces for adaptive security. *IEEE Trans. Dependable Sec. Comput.*, 15(3):466–480, 2018.

[WBD+18] Matt Webster, Michael Breza, Clare Dixon, Michael Fisher, and Julie A. McCann. Formal verification of synchronisation, gossip and environmental effects for wireless sensor networks. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 76, 2018.