Designing Self-Organising Emergent Systems based on Information Flows and Feedback-loops

Tom De Wolf and Tom Holvoet, {Tom.DeWolf,Tom.Holvoet}@cs.kuleuven.be
Dep. of Computer Science, KU Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium

Abstract

For self-organising emergent solutions, there is no support to explicitly design system-wide information flows and feedback-loops in those flows. This is problematic because the problem-solving power mainly resides in coordination between agents which needs information exchange. The flows of inherently decentralised information are essential for emergence. Feedback loops are a prerequisite for self-organisation. The design should explicitly focus on information flows. Otherwise, engineers are overwhelmed by details of decentralised coordination mechanisms such as gradient fields without first looking at the bigger coordination picture. We introduce “Information Flow” as abstraction to design a solution independently of the coordination mechanism details by explicitly identifying information flows and designing feedback-loops. UML 2 activity diagrams are the notation. An automated guided vehicle transportation system shows the usefulness and feasibility.

Introduction. Self-organising emergent solutions are a promising approach for inherently decentralised and highly dynamic problem domains. A system exhibits emergence when there are coherent macroscopic or system-wide properties, behaviours, structures, etc, that arise solely from the microscopic or local interactions between agents which know nothing of the desired macroscopic property [2]. Self-organisation is a dynamical and adaptive process where systems acquire and maintain structure without external control [2]. A combination of both promises a decentralised, flexible, and robust solution. Typically, decentralised coordination mechanisms such as gradient fields are used [3]. Designers are overwhelmed by the details of these mechanisms (e.g. large set of parameters, which mechanism?, how to use it?, etc.). The bigger coordination picture needs to be considered beforehand.

The main problem-solving power of emergence resides in coordination between agents instead of in intelligent reasoning of individual agents. Agents need local access to information located elsewhere in the system to coordinate their next actions. Emergence requires information flows towards agents. Explicitly modelling macroscopic information flows and relating them to microscopic actions allows to design at an abstraction level closer to the macro-level and independent of microscopic details of decentralised coordination mechanisms that enable information flows. Feedback-loops in information flows are a prerequisite for self-organisation [8]: "Think Flows rather than Transitions. Our training [...] conceives [...] processes in terms of discrete state transitions, but the role of autocatalysis in [...] self-organization urges us to pay attention to the flows of information among them, and to ensure that these flows include closed loops." Autocatalysis comes from chemistry and indicates that a product of a reaction serves as a catalyst for that same reaction. Agent actions need to stimulate other actions and eventually re-stimulate the first actions. The interplay between information flows and actions of agents has to form closed feedback loops. Modelling information flows allows to design loops explicitly and focus the creative effort on the enabler of self-organisation.

We introduced “Information Flow” as design abstraction in [4] to explicitly identify information flows. This paper additionally identifies the need for feedback-loops, and applies the approach to a new and complex case study on automated guided vehicles. UML 2 activity diagrams [1] are the notation because they serve the need to indicate which actions need which information flow(s). An extended version of this paper in which UML 2 diagrams are motivated extensively, in which a second case is worked out in detail, and with more related work can be found in [5].

Case Study: Automated Guided Vehicles. An Automated Guided Vehicle (AGV) warehouse transportation system uses multiple computer guided vehicles, called AGVs. AGVs move loads (e.g. packets, materials) in a warehouse. Each AGV can only do a limited set of local actions such as move, pick load, and drop load. The goal is to efficiently transport incoming loads to their destination. Figure 1 shows a simple factory lay-out: locations where loads must be picked up at the left, destinations at the right, and a network in-between of stations and segments on which AGVs move. The AGV problem is dynamic: many lay-outs, loads arrive at any moment, AGVs
move constantly and fail, obstacles and congestion appear, etc. Information on this is scattered through the system, but needed by AGVs to decide on their next move. AGV movement should result in feedback to each other.

**Information Flow Design.** To take information into account it must be available to the agent locally. An AGV needs to know the available loads to pick up, but this information is located with the loads elsewhere in the system. To make the notion of ‘local’ more concrete we define ‘Locality’. ‘Locality’ w.r.t. a system entity (e.g. an agent, loads, stations, etc.) is defined as that limited part of the system for which the information located there is directly accessible to the entity, i.e. no need to communicate with other entities. A locality is designed for each system entity, and differs in each application. A locality can be limited in distance, time, number of network hops, etc. To achieve global coordination in a decentralised way, information scattered through the system flows or propagates between localities of entities needing and/or providing the information:

> An information flow is a stream of information from source localities towards destination localities and this stream is maintained and regularly updated to reflect changes in the system. Between sources and destinations, a flow can pass other localities where new information can be aggregated and combined into the information flow.

Experience in building self-organising emergent solutions indicates that it is impossible to analytically derive the microscopic actions and information flows for achieving a given macroscopic behaviour. This remains a creative step. Information flows at least explicitly support designers to first focus on the system-wide flows, and less on overwhelming details of coordination mechanisms [3]. What follows outlines the information flows and feedback loops for the AGV case. There is a small notational difference in this paper compared with UML 2 [1]. Data flow edges are dashed lines instead of solid lines to emphasise them as information flows. Tick gray arrows annotate feedback-loops.

**Identifying Information Flows.** The steps to identify information flows shown below are not necessarily performed sequentially. An iterative process typically provides feedback to revise decisions if needed.

**Step 1: architecture and locality.** Before the design of information flows, the main structure or architecture is designed. This includes identification of agents, other entities, and the “locality” of each entity. There is no step-by-step procedure for this and needs to be done creatively. Each AGV is controlled by an agent. The locality of an AGV is its view range, i.e. the current station on which it is located and its outgoing segments. For a load this is its current location (AGV or station), and a station is its own locality.

**Step 2: microscopic actions.** A macroscopic requirement is achieved by microscopic actions that contribute to that requirement. Decomposing macroscopic behaviour in a set of actions is an important step. There are two main macroscopic requirements. First, incoming loads have to be dispatched to AGVs. Second, AGVs have to be routed in a factory. For dispatching the action is assign load. Routing AGVs implies move actions towards goal stations (i.e. pick-up or drop-off station). Also pick-up and drop-off actions are needed.

**Step 3: information needed.** The information needed in the localities of agents deciding on the actions is identified. AGVs can only locally decide where to move if they know which directions lead to the goal and how fast (e.g. distance). Also information on which directions lead to congestions and/or obstacles is important. This information is located with the destination and intermediate stations. To assign a load information is needed on the available loads which is located with the loads.

**Step 4: information flows.** Once the designer knows which information is needed and where it originates, the information flows can be established. The diagram in figure 2 is constructed as follows. Each locality (e.g. AGV view range, load location, station) has its own partition in which the actions are executed by the entity of the locality (e.g. AGV x). Information flows are shown as data flow edges with the type of information as the data type (e.g. Load Info). Some localities have parameters x and i indicating that it exists for multiple AGVs and loads. Information flows are established for all AGVs or loads, unless indicated by a condition on the flow (e.g. \(\text{AGV within distance Y}\)). We discuss the different information flows for dispatching and routing. These flows should form feedback loops (see later).

**Dispatching.** We only assign a load permanently to an AGV when the load is picked-up. Until then, other AGVs that are better suited can take over (upper-left of figure 2). An AGV chooses the best load \(d\) (Choose Best Load \(d\)) and intends to pick it up by Moving Towards Load \(d\).

![Figure 1. AGV case: example factory lay-out.](image)
This is repeated until Pick-Up is executed. Then Assign Load $i$ To AGV $x$ is executed. We use these flows:

- **FirstInQueue and Load Info:** To choose a load an AGV needs information on available loads, i.e. Load Info flow from the load locality. This flow only starts for the first load in the pick-up station queues. A FirstInQueue flow from the pick-up station locality towards the load locality triggers this.
- **Pick-Up Info:** After a pick-up, the load is assigned to the AGV (Assign Load $i$ To AGV $x$). Information is needed on when the load is picked-up, i.e. Pick-Up Info flow to load $d$ only ($d = i$).
- **Delivered Load Info:** This is feedback on a drop-off.
- **Possible Routes Info:** To move to load $d$, information is needed on the possible routes (see below).

**Routing.** The following information flows are needed for Move Towards actions (bottom-right of figure 2):

- **Possible Routes Info:** AGVs move towards pick-up stations (Move Towards Load) to Pick-Up loads, then move towards drop-off stations (Move Towards Destination for Load) to Drop-Off loads. Each AGV needs information on Possible Routes towards a destination, taking into account distance, obstacles, and congestions. A more complex information flow is required that aggregates and calculates the information while flowing through the system. Information on pick-up and drop-off stations (Destination Info) flows towards AGVs. The information passes through stations and segments on the route between the AGV and the destination (i.e. Intermediate Station). Information on obstacles and congestion is incorporated in the flow when it passes (Obstacle Info, Congestion Info). The flow can pass
multiple Intermediate Stations before reaching the AGV locality. A transformation represents the combination of obstacle/congestion info collected at multiple stations with info arriving from the pick-up or drop-off stations into Possible Routes Info. The transformation also calculates the distances. Eventually this flow arrives with AGVs looking for or holding the load ([for load] or conditions).

- **Position Info:** The Possible Routes Info flow needs info on AGV locations to generate congestion and/or obstacle info. Position Info flows from AGV localities towards intermediate station localities when AGVs move.

**Step 5:** Coordination mechanisms. Information flows still need to be achieved by decentralised coordination mechanisms. Examples are digital pheromones, gradient fields, market-based coordination, etc. [3]. Each flow imposes requirements on the mechanism, and each mechanism has characteristics to consider when deciding on its suitability. The Possible Routes Info flow requires a mechanism that enables aggregation and calculation (i.e. distances) of extra information while flowing through intermediate stations. Gradient fields allow this and are a good choice [3]. Information flows are data flows in activity diagrams which allow to design a solution independently from the details of decentralised coordination mechanisms (e.g. direct messages or gradient fields).

**Feedback-loops.** The feedback-loops in the AGV case:

AGVs - Dispatching - Loop 1. A first loop is established for dispatching loads (tick gray arrows and number 1 in figure 2). From the pick-up station the FirstInQueue flow triggers the Load Info flow in the load locality. This triggers the Choose Best Load, Move Towards Load, and Pick-Up actions. At that moment the AGV locality equals the pick-up station locality and a new load becomes first in the queue. This restarts the loop (dashed tick arrow).

AGVs - Routing - Loop 2. A second loop is established for routing AGVs (tick gray arrows and number 2 in figure 2). AGVs move based on Possible Routes Info. This movement generates the Position Info flow towards where Obstacle Info and Congestion Info are added to the Possible Routes Info flow. As such AGVs give feedback to each other while routing.

**Reflection.** To design self-organising feedback loops, the interplay between information flows and actions on control flows needs to be considered. Thus diagrams only showing information flows such as UML 2 information flow diagrams [7] serve as a clear but limited view. Activity diagrams are more expressive to establish a good solution.

**Discussion.** The information flow diagram in figure 2 is rather complex. It includes both control and information flows. Separating these concerns could simplify the diagrams but makes it harder to get an overview on how control and information flows interact. This makes designing feedback loops harder because loops result from the interplay between control and information flows. It is a trade-off between separation of concerns, complexity, and having a complete picture. Data Flow Diagrams (DFDs) [6] are a similar notation which only shows data flows. This makes it impossible to discover all self-organising loops. DFDs have no formal specification like UML 2 activity diagrams. DFDs are less expressive: no control nodes, partitions, or multiplicities. DFDs are well-known but exist next to the more widely used UML notation which is therefore preferred. Also our approach needs a more rigorous formal specification, for example, to automatically check for loops.

**Conclusion.** The information flow based modelling approach allows to design a solution independently of coordination mechanism details. The approach addresses emergence by explicitly identifying flows of inherently decentralised information through the system. Self-organisation is addressed by explicitly designing feedback-loops. An engineer designs at an abstraction level relating system-wide information flows with individual agent actions, which is where the creative effort is needed for self-organising emergent solutions.

**References**


