Stigmergy: a design pattern for product-driven systems

Rémi Pannequin, André Thomas

STIGMERGY: A DESIGN PATTERN FOR PRODUCT-DRIVEN SYSTEMS

Rémi Pannequin, André Thomas

Research Centre for Automatic Control (CRAN)
CNRS (UMR 7029) – Nancy University,
27, rue du merle blanc, F-88000 Epinal, France
{remi.pannequin, andre.thomas}@cran.uhp-nancy.fr

Abstract: This paper proposes a new interpretation of stigmergy, where cooperation between production actors is achieved thanks to attributes (informational pheromones) attached to products. From this interpretation is introduced a new design pattern that can be used to develop product-driven systems. Agent-oriented components which implement it are presented, and then applied on an experimental platform.

Keywords: multi-agent systems; software architecture; manufacturing control; product-driven control.

1. INTRODUCTION

High competition between enterprises and market volatility lead enterprises to be more *agile* (Christopher, 2000). Agility, from the point of view of production control, may be seen as the ability to operate with a high level of coordination and proactivity throughout the supply-chain, and *at the same time* to react efficiently to disturbances on the shop floor while taking into account the increasing process complexity (variabilities, high product variety, reconfiguration issues).

The first point have been targeted by a "centralized" approach of decision-making, using tools and methods mainly based on operation research, while in the second one a "distributed" approach of decision-making, such as anthropocentric or visual management methods (kanban, empowered operators, ...) and more recently holonic and agent-based manufacturing (Babiceanu and Chen, 2006), have been also successful.

To combine these two centralized and distributed approaches of decision making, *hybrid* systems have been proposed. The key point to make this hybridation possible, is to define an interface able to cope with the differences between the two decision making approaches, and to ensure a coherence between the physical flows and their informational representations.

The *product-driven* paradigm (Morel *et al.*, 2007) is based on the assumption that the product is the core object is this system. Indeed, the product is the common object shared by a vast majority of services in the enterprise, for instance centralized and distributed decision systems. Therefore, hybrid systems may be enabled by active products, which integrate every actor in the company, from the central (software) systems to the processes, products and also operators, into the same *ambiant* information system. Using a combination of identification, embedded systems and agent technologies, the physical product itself can become active in it environment (Mcfarlane *et al.*, 2003).

However, developing product-driven systems (PDS) asks a number of questions, ranging from conceptual issues, to logical and technical ones. This paper focus on architectural aspects of PDS. Indeed, if there is a consensus about the concept of an *active product*, the design patterns required to actually develop such a system are not well-defined.

Therefore, this paper propose a architectural pattern based on stigmergy to design product-driven systems.

First, we will state the problems faced while designing product-driven systems, and remind the principle of stigmergy. Then, in section three, the actual proposition will be enunciated. Finally, section four will present an application of our approach on a experimental platform.

2. PROBLEM STATEMENT

2.1 Architectural patterns

One of the main issue to solve in the development phase of a PDS (or in any distributed control system) is what entities to define, and how to structure their interactions to achieve the industrial goal. PROSA (van Brussel *et al.*, 1998) defines four types of entities (products, orders, resources and staff agents), and has been used by many implementations of PDS (with minors variations concerning product vs order agents).

Some patterns have also been proposed to structure entities. The simplest pattern is a master/slave coordination, where agents receive requests, that are broken down into sub-requests sent to slave agents. Likewise, reports are aggregated and sent back to the original request initiator. This pattern define a hierarchical structure (as in (Albus and Barbera, 2005)) that enable to achieve high performance levels but that is often rigid and therefore not able to adapt to changing operation conditions. To solve some of this issues, unconstrained, dynamic or partial hierarchies have been proposed (Brennan and Norrie, 2001), but remains hard to implement.

Another approach is to use negotiation or auction-based patterns. These approaches are based on the emergence of a complex global behavior from simple local entities that interacts. Because only elemental behaviors are defined, such emerging systems are able to adapt easily to change. However, their global behavior is hard to predict and misbehaviors such as famine, deadlocks and livelocks can degrade performance. The design of such system is often inspired by human or animal behavior. For instance the contract-net protocol (Smith, 1980) is one of the most widely used negotiation pattern, and has been industrially applied, for instance at Daimler (Bussmann and Schild, 2001).

2.2 Stigmergy

Stigmergy is a cooperation mechanism that come from the study of animal behavior (ethology), in particular social insects. It has been first observed in the case of nest-constructing termites by (Grassé, 1959), who coined the word stigmergy from the Latin roots *stigma* (sign) and *ergon* (work). He observed that coordination between the numerous individuals implied in nest building was achieved using pheromones. Pheromones are chemical substances released by an insect that causes another individual of the same species to react. Termites release pheromones on the building material (dirt balls), thus influencing the behavior of others termites.

Using stigmergy, a global coordination of the complex process involving many entities is done without any global design, and only through indirect interactions through *pheromones*. So stigmergy can be defined as a coordination mechanism of many workers by the mean of pieces of information deposited on their common work.

One of the most famous application of stigmergy is the ant colony optimization algorithm (ACO) (Dorigo and Gambardella, 1997). This algorithm comes from the modeling of the behavior of foraging ants. To find the shortest path between their anthill and a food source, ants release pheromones to mark their tracks. It has been shown that he pheromone path converge to the shortest path. ACO has been used to solve traveling-salesman kind of optimization problems. It has also been transposed in the domain of manufacturing control (Valckenaers *et al.*, 2006).

In this application, orders generate various kind of ants, who search the better route for their orders in the network of manufacturing resources, and release pheromones on the resources which correspond to their intension of using this resource sometime in the future (thus providing proactivity).

The concept of a common work marked with pheromones that control (or influence) the behavior of workers seems quite close to the concept of an active product, able to interact with its environment. Thus, we propose an pattern based on stigmergy to design product-driven control architectures.

3. PROPOSITION

3.1 Stigmergy-based pattern

In this paper, we apply stigmergy to the manufacturing context by interpreting products as the *common work* and operators, decision systems and processes as the *workers*. Comparatively, approaches based on ant colony optimization identify resources reservation as the common work (the path) and products as workers (ants).

According to this interpretation, we define a *stigmergic product* as a physical object able to carry data (the computer equivalent of pheromones). Likewise, we call *actor* any system that contributes to the elaboration of the product, either directly by transforming its morphology or its position (physical resources), or indirectly by producing and consuming control annotation attached to the products. Noteworthy, the stigmergic product is closely related to the concept of holonic product, in the interpretation that an holonic product

enable to see the physical part and the informational part as a complete object.

Actors can interact with products by reading or writing their attributes. Depending of the kind of actor, three categories of attribute can be defined:

- what the product currently is;
- the requirements that it will have to meet, from the point of view of physics (customer specifications), or of the point of view of control, such as a centralized decisions (e.g. a scheduled processing date);
- the history of what was made: this includes physical transformations (morphological and spacial changes), and control decisions, such as the results of distributed decision processes (e. g. a routing decision).

But beyond simple product-actor interaction, the key point of the proposed stigmergic architecture is to achieve coordination *between* actors only by changing products' attributes, and by reacting to such changes. Two specific cases of this coordination through product's attribute can be highlighted:

- coordination between control actors. For instance, an operator can be informed of centrally-made schedule thanks to a priority attribute (Pannequin *et al.*, 2009).
- coordination between a control actor and the observation system. Physical observation of the product (e. g. an event reporting that product p is arrived in station s) may trigger an attribute change, that can then be perceived by the corresponding control actor.

Finally, the product itself is able to observe its own attributes, and can react to changes of one of its attributes by modifying others. This feature enable the product to manage its own life-cycle, by setting the value of attributes describing its current state and requirements, according to the transformations made.

The stigmergic pattern is summarized by an UML class diagram presented figure 1.

3.2 Interactions with the stigmergic product

According to our interpretation of stigmergy, actors interacts with products by observing and modifying their attributes. The question is to specify the most efficient mechanism to implement these interactions. In social insects behavior, pheromones are deposed directly on the environment (i.e. common work). Thus, an insect (worker) is instantly notified of any relevant information in its current context. This makes appear two essential features of the interaction with pheromones. First, the worker get *only* information that are relevant in a particular context, second, the worker don't have the initiative of the communication.

In the proposed architecture, these features are provided by a subscription/notification mechanism. When a product is created, it broadcast a list of its attributes to other actors. The actors can then respond by subscribing to some attributes. A subscription is a pair (*attributename*, *pattern*). When the attribute's value changes and matches the pattern, the subscriber is notified. The pattern enable to filters out notifications that are not relevant for the actor.

This subscription-based mechanism can be implemented in various ways. For instance, in the object-oriented paradigm, the model-view-controller can provide a way to implement this notification mechanism. In this paper, the multi-agent paradigm have been used. This ensure a clear separation between entities (products and actors are modeled as separate agents) and impose to specify the content language and interaction protocols that are used for agents communication.

A content language has been defined to implement communication among agents. Figure 2 shows two examples of such messages, encoded in the FIPA-SL0 language. Two main concepts have been used: the first one is the concept of 'attribute', seen as the association of a name and a value. Both name and value are strings. The second concept is 'holon', defined by its name (a string) and type (three type of holon have been defined: product, resources and staff).

Several predicates and actions can be build on these concepts. The predicates 'owns', 'has-value' and 'matches' describe respectively that a holon has a sequence of attributes, that a holon's attribute has a particular value, or that a holon attribute value matches a particular pattern (regular expression). The actions change-value and query-value are used to modify or query an attribute value.

```
((action
   (agent-identifier
      :name p1-pla2@tracilog)
   (change
      :attribute (
         attribute
            :name affectation
            :value p0))
))
```

Fig. 2. Example of an attribute change request message in the Fipa-SL0 language.

Several interaction protocols have also been used. The FIPA-subscribe (FIPA, 2005) implements the notification mechanism. First, product agents declare their attribute list, by broadcasting an INFORM message. Agents that want to be notified when an attribute value matches a pattern, initiate a the subscription interaction with the product agent. Moreover, other request-like protocols (such as FIPA-Query and FIPA-Request) are used to query or modify products attributes.

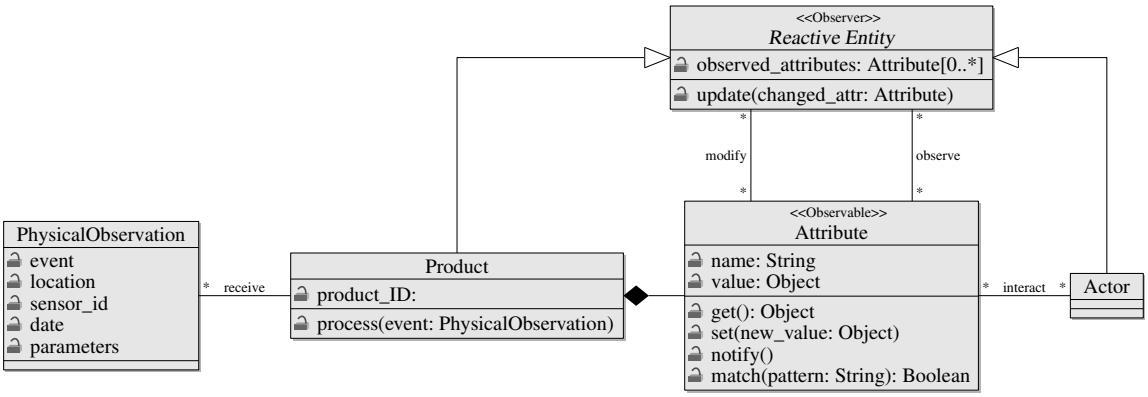

Fig. 1. UML Class Diagram of the proposed stigmergic pattern.

3.3 Structure of an stigmergic product

A stigmergic product is represented as an agent, that is both connected to the physical product, and to others agents representing production *actors*. At the core of this agent is the product's attribute base (figure 3).

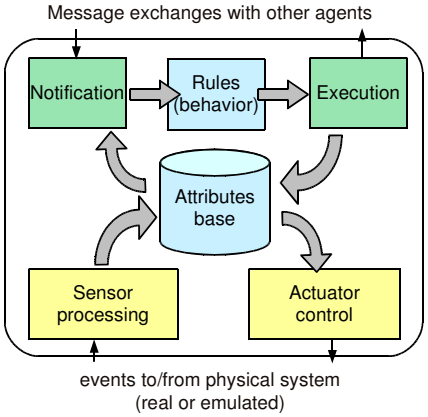

Fig. 3. Internal structure of a stigmergic physical agent

Physical observations received by an agent results in modifications in the attribute base. Several processing approach can be used: on the one hand, an attribute can take the direct value of the event name, or one of the event's parameter (e. g. the epc attribute is assigned after the epc parameter of a RFID-reading event). On the other hand, the attributes value can be indirectly computer from the event. The most typical case is to use a finite-state machine for that task.

Product agents are able to communicate with other actors about attribute values, thanks to two subsystems. The first one notify other agents of attribute changes and respond to queries about attribute value, while the second one receive change requests and implements them in the attribute base.

Finally, the product agent can execute a rule, that represent its behavior. The rule is notified of attribute change and can then modify local attributes.

Many components of this structure are generic and can be re-used. However, when developing a new application of a product-driven system, the rule must be written, and the product attributes defined.

4. APPLICATION

4.1 Test-case presentation

The proposition has been tested on an experimental platform. This platform assembles products composed of:

- a support;
- up to four wooden square pieces, yellow or orange;
- up to four wooden chips.

This product structure generate more than one thousand differents products, providing sufficient variety. Figure 4 shows a some examples of assembled products.

Fig. 4. Examples of products

The experimental platform has four stations linked by a conveyor belt (figure 5) : *M1* is an assembing station, where square pieces are assembled on support; *M2* is a transformation station, this transformation being concretized by putting some chips on the product; *A1* and *A2* are routting stations.

The product components are equipped with identification devices. The supports and square pieces carry a RFID tag. RFID readers are placed in front of each station, and additional readers enable to check products integrity. The color of the square pieces is deduced from reading their tag.

The main issue to solve on this platform is product variety. Indeed, we have witnessed that vendors control systems were not able to deal with mixed flows: the assembly cell could produce only one type of product

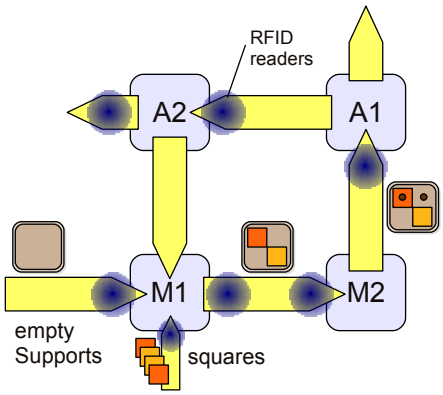

Fig. 5. A overview of the material flow in the test-case

at a time, and must be emptied before setting up a new type. To reduce these setup times to a minimum, the control system must be able to manage a flow where each product is different from the other ones.

Moreover, to test the robustness of the variety management feature, flow disturbances have been introduced: products might be taken from the conveyor belt, and put back elsewhere.

4.2 Intercations between customers, products and processes

The test-case system is controlled using a multi-agent system. Agents have been associated to resources (M1, M2, A1 and A2), and to products (both the support and the square pieces). Staff agents are present in the platform to enable customers to configure and generate production orders (i.e. product agents). Finally, the platform comprises utilities agents, to communicate with the physical system (the programmable logic controller and the RFID subsystems), or to offer common services (agent creation, directory facilitators, ...). The typical number of agents cooperating is between 10 and 50.

A typical interaction scenario correspond to the following sequence of message. First, products are configured from the business point of view and are then introduced on the multi-agent system.

(1) The customer-agent creates a new product-agent, wait agent initialization, and sets the 'reference' attribute of the agent.
(2) The product agent declares its attributes. Actors subscribes to some to them. For instance, M2 subscribes to the attribute 'state', with value 'stationM2'.
(3) The newly created product agent reacts to the change of its reference attribute, initializes its internal bill of operations and set it progM1, progM2,... attributes accordingly.
(4) A physical product is associated with the product agent by scanning its RFID tag. An association between the product ID and the epc code of the tag is created, enabling event brokers to route RFID event to the right product agent.

Then, products agents interacts with the process, until production finishes.

(5) When the physical product arrives in a station (e. g. in M2), a RFID event is received by the product agent, and the 'state' attribute is consequently changed (e. g. 'stationM2')
(6) The station's control agent is notified of the new value of the product's state attribute. It queries the product about its requirements (e. g. attribute progM2), and transforms the physical product accordingly.
(7) When the transformation ends, the station control agent releases the product, and request to change a report attribute (e. g. reportM2). The product agent reacts to this change by updating its internal bill of operation and consequently modify attribute value.

This approach based on stigmergy enables a high product variety (the flow can mix any number of different products), and also a high robustness (products are able to interact with any resource at any time, allowing them to find back their normal route). Comparatively, an existing traditional implementation, developed by an independent engineering company, allowed only to produce one type of product at a time, and was not able to cope with flow disturbances.

The implementation of this system is a relatively simple task, because only the behavior rules and the attribute base have to be implemented. Indeed, most of the components are generics and may be re-used.

The most important feature of the system is the high functionnal independance between defining products requirements, managing products life-cycle, and controlling product transformation resources. Since interactions are based on product attributes, the system can also cope with a completely asynchronous execution of all these tasks.

4.3 Special case of assembly

The assembly between a support an square pieces is a special case, because both types of pieces are controlled by a product-agent. Therefore, three agents must interact here: the product-agent associated with the support (S), the product agents associated with a square piece (Sq), and the resource agent controlling manipulator $M1$. In this situation, the stigmergic pattern can be applied at a smaller scale (figure 6).

Agent S creates agent Sq, get its 'color' attribute, decide to assemble it at some position or to discard it, and set its 'destination' attribute accordingly. Agent $M1$ is notified of the 'destination' attribute of Sq, move the square piece, and finally reports this action.