34 EVOLVABLE ASSEMBLY SYSTEMS BASIC PRINCIPLES

Mauro Onori
Royal Institute of Technology, onori@iip.kth.se
José Barata
Universidade Nova de Lisboa /UNINOVA, jab@uninova.pt
Regina Frei
Royal Institute of Technology, onori@iip.kth.se

This paper addresses the underlying principles of Evolvable Assembly Systems. This paradigm was recently proposed as an answer to the requirements faced by assembly companies in the current world of business and technological changes. The basis for this new approach lies in a multi-disciplinary study of the needs and requirements, and shifts the technological focus from complex, flexible, multi-purpose systems to simpler, process-oriented, dedicated swarms of machine modules.

1. INTRODUCTION

Notwithstanding recent technological advances, the social and economic situation for assembly-intensive companies in Europe is facing considerable challenges. Recent studies quantify European outsourcing at 21% of total assembly activities, and have forecasted a rise to over 40% by 2007 (Kirsten 2002). The hidden issue behind such facts is that assembly has not been rendered cost-effective yet, and this is particularly true for novel products and markets, such as the micro-assembly domain. In order to counter this lack of adequate automation, and to rise to the challenge, the Evolvable Assembly Systems (EAS) paradigm was launched a few years ago (Alsterman et al. 2004; Barata et al. 2005; Onori 2002). The basis for this new approach lies in a multi-disciplinary study of the needs and requirements, and shifts the technological focus from complex, flexible, multi-purpose systems to simpler, process-oriented, dedicated swarms of machine modules. From a paradigm point of view, the EAS approach rejects machine flexibility (sub-optimal at any specific task) for application agility. This paper will attempt to detail the recent developments within Evolvable Assembly Systems, including ontological, methodological, and application developments.

2. BASIC FOUNDATIONS

Evolvable Assembly Systems (EAS) is commonly understood to be just another interpretation of Reconfigurable Assembly Systems (RAS). Since this is not the

Please use the following format when citing this chapter:

Onori, M., Barata, J., Frei, R., 2006, in IFIP International Federation for Information Processing, Volume 220, Information Technology for Balanced Manufacturing Systems, ed. Shen, W., (Boston: Springer), pp. 317–328.

case, it is of some importance to clarify how EAS differs, and which new aspects it proposes. Re-configurable systems take a system perspective and start with, in most cases, the current product and user requirements. In fact, re-configurability is a further development of flexibility. Flexible systems provided multi-purpose machines that were fairly good at many tasks but optimal in none. And the cost was high. Re-configurability has sub-divided these machines into smaller units and focussed on the system interfaces, but the focus is far too limited in time: current products and company aspects! EAS differs radically, and the following clarifies some aspects:

- Main focus – RAS focuses on re-configurability (geometric setup) of system components; not necessarily automatic. EAS focuses on adaptability of system components through capture of emergent properties; semi or fully automatic. This may raise safety issues.
- Development trigger issue – RAS uses current (existing product features as a development trigger, whereas EAS focuses on the re-engineering needs of the assembly system (product shift).
- Modularity level – RAS applies conventional subdivision of assembly into "transport-handling-assembly-finalisation" blocks, which results in coarse granularity. EAS applies lower-level modularity based on process-level characteristics: the "operational" level becomes subject to modularity, hence resulting in fine granularity.

In truth, the differences are many, and EAS provides, through its tight process-oriented modularity, a much closer link between product design and assembly system development. In order to further clarify how EAS operates, and upon which premises it builds its theoretical and technological foundations, a brief description of the main aspects will follow.

2.1 System Concept

EAS proposes, basically, a radical new way of thinking: in terms of assembly systems, what is required is not a complex solution which tries to accomplish all of the envisaged assembly needs within a closed unit (Flexible Assembly systems) but, rather, a solution which, being based on many simple, re-configurable, task-specific elements (system modules), allows for a continuous evolution of the assembly system. This, in many ways, is an interpretation of swarm strategies. In fact, it is the aggregation of many small, simple entities that will enable a given functionality, and this functionality can disappear by the removal of some of these entities. New functionalities are simply created by forming different formations, or coalitions of entities.

EAS also goes a step further by proposing a totally new way a considering the products and assembly systems: the design cycle of the products will be influenced by which modules are available. In other words, many simple, strictly task-oriented components with standard interfaces are better than few, very flexible but extremely expensive solutions that cannot be integrated within existing systems: in effect, there are no defined systems within EAS, only process-oriented modules. This, however, brings us to a discussion of what a system is. In fact, a system's capability is NOT the sum of the capabilities of its parts. It could be a completely different scenario altogether. The issue being raised here is that EAS will consist of a vast range of

inter-connectable modules. When a system will be created according to the EAS principles, the resulting capability of the sum of the modules will not be so easily predicted. What happens, in fact, when a multitude of small entities is brought together, is that new, unthought-of capabilities may emerge from this coalition. Hence the need to start studying the principles of *emergence*, since the capabilities being brought together may also be viewed as particular functionalities (skills) being offered by each module. A very important point to highlight is the fact that the properties that emerge out of the interaction of the modules that compose the assembly system represent just the complex functionalities (complex skills) of the system we want to create (Barata 2005). The interesting conclusion is that we can generate any functionality we want as long as we provide the control architecture able to accommodate the interaction of modules.

2.2 Control Concept

Starting with the basic assumptions enunciated in the previous section, it becomes clear that traditional control paradigms cannot be used to support EAS. In effect the control solution for EAS must cope with the following aspects:

1) Support the integration of modular components that might include their own controllers with different levels of intelligence,
2) product changes,
3) fluctuations in demand,
4) support the addition and removal of components during normal production,
5) provide support for the operative phase.

Support the Integration of Modular Components

In terms of integration of modular components, then the centralised approach is immediately at a loss since the basic assumption under which EAS is grounded is the idea of independent modules that may be reused and put together at will, according to the system that is being created (coalition). With such an objective the main goal is about plugability or the possibility that coalitions of modules need to be created in a very fast way without any or just minor programming changes. This requirement cannot be achieved with a centralised approach which may be optimal for static systems but copes poorly with dynamic life cycle systems. The hierarchical approach is also very weak because hierarchies deal poorly with structural changes. Even if each module has its own controller, and therefore becomes distributed, it is not convenient that each module controller has functionalities that allow essentially the creation of very static hierarchies.

Hence a different approach should be followed in which each module should have a level of intelligence that enables it to participate in societies of modules (coalitions) following different types of hierarchical structures. This means a completely new approach on how module controllers should be designed because modules do not know á priori (statically) what kind of requests and from whom these requests will come. These new extra functionalities for participating in dynamic coalitions with different types of hierarchical structures requires new challenges on the module controller architecture.

The multiagent paradigm is a good choice for two reasons. The first reason is related to modelling and abstraction. In fact it is quite simple to make the connection

between a module and a modular component. The second reason is related to the need for plugability. In effect if a modular system is being considered they must be plugged or unplugged at will. Therefore a multiagent system seems to be ideal because, by definition, a multiagent system is adequate to plug and unplug agents (in this case component modules). Another aspect connected to a modular system is the fact that a particular assembly or production system is a composition of modules, which can then be also considered as a coalition of modules. This is another connection between production modules and the agent world. A very well known and established work is in fact the work on agent coalitions (Castelfranchi et al. 1992; Klusch and Gerber 2002; Pechoucek et al. 2002; Shehory and Kraus 1995), which can be applied also here (Barata 2005).

It is important to emphasise that modularity does not immediately imply modules with very intelligent controllers. However, if it is considered that modules can be plugged and unplugged in a very simple way (none or minor programming effort) then at least some functionalities related to plugability are needed at module level.

Product changes

Product changes may imply alterations in the structure of the assembly system. The system evolves by changing, rapidly, the modules that compose the coalition. This change must be done with no or minor programming. The aspects of plugability referred to in the previous point are fundamental to ensure a smooth change. However, more than just simple plugability is needed because one must consider an entity that will be responsible for the changes in the coalition or society, and the controller modules must be able to cope with this entity by providing them with relevant information (such as the skills that each one is able to perform) in addition to information about their past experience.

Therefore, an entity that should be called upon whenever a major change is required to the coalition is being implied. This entity (tool) should be able to guide a systems integrator in the process of creating a new coalition fitted for a product family. The most important aspect is that this tool should be able to guide the user by suggesting that the coalition is not created based only on product features but also on the skills made available by the modules.

In otner words, each module should be able to supply information of its own skills or functionalities to be used by the external tool or configurator. In addition to this information, the user should also get information about the quality of the module he/she is choosing. Consequently, each module controller should include functionalities to store relevant information such as the number of faults, number of working days, etc and also to supply them whenever requested by the configurator. This means extra computational capability (or intelligence) for each module controller.

It is relevant to consider that whenever the coalition is created the set of module controllers (agents) start to interact and a new set of behaviours (skills) emerges. This means that whenever a coalition is created it is not necessary to program the interactions between its members. The only thing the configurator has to do is assist in the creation of the coalition.

The relation with the multiagent paradigm for this requirement is that an agent supports various behaviours. In fact, in the illustrated situation, the agent in addition to its normal work of controlling its module should also perform behaviours related

to answering requests about its own skills and monitoring and storing relevant information about its physical entity (module). On top of these aspects the configurator can itself be an agent, in this case a configurator agent that interacts with a user and with the society of modules to get information.

If the change on the product only implies alterations of the flow within the system then the change does not imply any changes to the coalition but only changes in the way the product is processed. If it is considered that the product is an entity that know its process steps (process sequence) and considering that the transportation system is intelligent enough to know what kind of modules and which skills they publish, then whenever there is a product change that can be coped within the system nothing is required to change since it is all about negotiation between the product and the transportation system. It must be clarified that what is being considered here is a situation in which each traveling product (piece) is represented within the control system as an intelligent entity that can interact with the other entities. If, for instance, it is considered that each product is transported by an autonomous entity (kind of AGV) that can move to the locals where the other modules might do some operation on the product, and in addition to that the transporter can make broadcasts to know which modules are available and which are their skills, then it becomes easier to understand why product changes of this type do not require any reprogramming.

Taking into consideration the previous requirements it is quite easy to make the connection between product changes and the need for a controller based on the multiagent paradigm. The autonomy makes the agent responsible for its own acts, which means that each module is individually responsible for the actions over the product. Adding or removing one of these modules does not have a big impact on the overall system structure since each module is a confined individual entity. It must be noted that being confined does not mean any interactions with the other modules. In fact the social ability that characterises the multiagent world is fundamental to ensure that the modules interact among themselves.

Fluctuations in the demand

This aspect is mostly related with the ability to plug or unplug modules, as well as with the capability of the modules to adapt themselves to the new requirements. If the demand increases a very obvious solution is adding new modules to increase throughput or in case of lower demand modules can be removed. But a less obvious solution of system adaptation to the new requisites may take place. For instance lets suppose the system is composed of intelligent modules that always tries to save energy. On a situation of low throughput, each module understands that it can do the work at slower speed, therefore saving energy. It can even be the case that parts of the system are shutdown, which means that certain modules are shutdown; only part of the controller is active, looking for changes in the demand. If a demand increases the modules detects this new situation and the ones already working start to produce faster and the ones that have shutdown themselves start working. The reverse happens in a situation of reduced demand.

This capacity flexibility can only be achieved if each module is intelligent enough to be able to execute these actions. It is very important to consider that the agents needs to interact among themselves in order to be able to sense that a demand

is increasing or decreasing. It would be ideal that the agents could figure out what is going on based on their local knowledge.

Addition or removal of modules during normal operation
This aspect is about minor changes that might be made to a system during normal operation. These minor changes might happen due to minor changes in the product that do not require the intervention of the configuration tool and/or to support maintenance interventions without having to stop the system. For instance, if the system is operating and at certain level a feeder or even a gripper is required to be removed for short period maintenance then the system should be able to still operating, even if at reduced capability. It might be the case that the removal of the gripper inhibits the capability of the cell to do some operations but the other might still be supported.

These requirements are again best suited for the multiagent paradigm since only with intelligent modules that support emergence in the sense that the new capabilities of the system are automatically derived whenever a new module is added or removed, it is possible to solve this problem.

An interesting side effect of this requirement is that the system can be composed manually without the need for a configurator. Because of the mergence that is supported by each module controller the behaviour of the system emerges naturally out of the coalition.

Operative phase
The need for better operational support means all the tasks that are required to support the shop floor while it is operating. This include aspects such as online reconfiguration of production parameters, maintenance support functionalities, advanced diagnostic systems, advanced user interfaces, etc. All these requirements are also better solved if the multiagent paradigm is considered, because among other aspects it is important to model each manufacturing component or module as an entity that includes all relevant information (maintenance parameters, process related variables, etc) that can be used in supporting the aspects referred before. This shows how the connection between modular component and agents is done. Another important aspect at this level is the autonomy that each agent (module) must possess. In fact each agent must support simultaneously different behaviours or actions. For instance an agent representing a certain module participating in a coalition (system) must simultaneously do the following tasks: 1) answer requirements for executing some work (for instance, the request to transport one piece from one point to another received by a gantry robot), 2) keeping monitoring the internal sensors of the module, 3) answering requests from external users for internal parameters values, 4) interacting with other modules for advanced diagnosis functionalities, etc.

2.3 Emergence

The EAS paradigm, considers the optimisation under conditions of change. As a *system*, an assembly system actually consists of many different, small, but cooperating, entities, or if we like, organisms. There will be higher organisms (the system as a whole, or modules) and lesser ones (grippers, fixtures). As this approach is adopted by EAS, with many simple, dedicated process units, special conditions

arise. This brings us to how systems should be interpreted, and a branch of studies called "Systems Thinking".

Systems Thinking enables one to progress beyond simply seeing events to seeing patterns of interaction and the underlying structures which are responsible for the patterns. A system is something that maintains its existence and functions as a whole through the interactions of its parts. It consists of many different parts and organs, each acting separately yet all working together and each affecting the others (analogy with EAS: The modules are the parts and the assembly system is the system).

The essential difference, from a EAS point of view, is that in a true system, when one part is added, the additional functionality is not an obvious addition of the added parts' functionality. This is what is called emergent properties, and is the basis of EAS. *Systems have emergent properties that are not found in their parts. You cannot predict the properties of a complete system by taking it to pieces and analysing its parts.*

Similarly, the smaller the constituents of the system, the easier it is to define, structure and coordinate the skills being brought in and out of it.

Note: the constituents have to be classified in terms of the overall system demands, changing external conditions, and particular tasks to de carried out.

It is suggested, due to the potentially large level of complexity, and liability issues, to subdivide emergence into 3 levels: primitive, communal, and full emergence.

Primitive emergence:
This refers mainly to very simple self-organisation. The main prerequisite is the ability of the system modules to self-recognition. This is to be viewed as a fundamental objective of plugability. There must be an open information exchange between the modules and the organiser of assembly events, whether this be at configurator level or performed autonomously by more "intelligent" (agent-based) modules. This type of emergence has a smaller effect on systems based on coarse modules: i,e.- the finer the granularity, the higher the need for primitive emergence to satisfy the reconfigurability & evolvability demands.

Communal emergence:
This refers mainly to self-organisation with adaptability. In this case the system is not only fully capable to self-program and setup itself, but it also enables some forms of adaptability. This is not only self-calibrations but, primarily, the ability of the system to adapt if changes are brought to the "world perimeter" its comprises, such as the product or fluctuation scenarios described earlier.

At this point, the emergent solution may detect possibilities and inform the user/controller of such emergent properties. An example of such Community-adaptive emergence could be the creation of a Pick&Place capability out of a gripper and one or two linear or rotational movements.

Full/Evolutionary emergence:
This refers to full adaptability and evolvability. This level enters the realm of classical emergence, in which the emergent property itself is often unpredictable and unprecedented, and may represent a new level of the system's evolution. Full

emergence means that EAS learns how to perform "tricky" tasks, similar to the way a human operator gets better when he gains more experience. This means that the systems has to be taught how to use its acquired knowledge to draw further conclusion. Of great importance is the idea that certainty/uncertainty aspects surface. The emergent properties have never before been tested and exploited, are the result of previously unknown logical/social or other patterns, all of which raises liability and safety issues. *It is for this reason alone that it is of utmost importance to understand the value of an emulator:* it will enable the user to verify the emerging skills/properties before one builds the actual system! If this is available, the liability issues would also become more treatable.

2.4 Evolvability

When considering evolvability, or EAS, one often makes the terrible mistake of considering qualitative and quantitative features on the same plane of thought. This has led the EAS developments into quite some confusion, and some preliminary clarification is required.

A qualitative feature describes a given EAS characteristic from a general viewpoint, and cannot easily be assigned any set metric for quantification, unless this metric is clearly defined.

A quantitative feature describes a performance characteristic and can be measured. Hence, when discussing evolvability one should first ask whether such concepts may be measured at all. In an attempt to clarify this, the article proposes a tentative set of definitions.

First of all, the evolvability resides within the system characteristics (we will always assemble, etc.- functionality not evolvable, but the qualitative attributes), which must be assumed to be mechatronic in nature. The assembly system may then be defined as an evolvable system if:

- It is a fully "reconfigurable" mechatronic system platform that exhibits an emergent behaviour.
- The assembly units and modules are mechatronically integratable.
- The evolvable & reconfigurable system is composed of process-oriented components.
- The system can automatically determine its functionality based on the components' skills (when components are plugged together to form it).
- There is no (or minimal) investment in the programming & coding, but, rather, in how to establish and exploit relations.
- Maintenance, documentation and the ability to store information in support of operational stability.

However, evolvability is not attained by a single giant leap forward in technology. It is attained in carefully measured steps, that start with specified modularity. From a process-oriented set of modules, one must then assure plugability, followed by reconfigurability and, finally, evolvability. A full set of definitions and clarifications will be given later.

3. EAS PREREQUISITES

Several partners have, for the past few years, elaborated on the EAS paradigm. Some of the more recent developments, such as the ABAS platform, clearly attempt to fathom out the prerequisites and move forward. The EUPASS Integrated Project has been the largest endeavour to date to follow the EAS principles, and has, together with more pinpointed efforts between KTH, UNINOVA, and EPFL, elaborated a set of foundations for EAS.

3.1 Definitions and metrics

As stated earlier, it became quite clear, at a very early stage, that means to validate the concept were needed. The first step was to define the way certain aspects were being interpreted, and then move forward to try and set up adequate metrics for exploitable validation. The basic definitions are given below, starting with the basic EAS prerequisite, which is a process-oriented set of assembly modules.

Module – Any unit that can perform an operation and integrates a specific interface. Granularity level needs to be defined (the lower the level, the higher the degree of emergence)

Granularity – The lowest level of device being considered within a reference architecture. The lower the level of building block (tool, gripper), the higher the emergent behaviour: if a gripper can "communicate" with a robot, new operational characteristics may emerge (flip product/part in motion, fine positioning...). However, this implies that an overload of definitions and information management may arise, and a minimum level needs to be clearly set.

Plugability – The ability to rearrange and integrate system components within the framework of a given system architecture. The resulting new layout does not preclude efficient performance, one simply and physically plugs together a new arrangement.

Reconfigurability (interoperability) – The ability to reaarrange available system components to perform new, but pre-defined operations (plugability plus characteristics that ensures the efficient performance of the resulting new layout).

Evolvability – It is a fully "reconfigurable" mechatronic system platform that exhibits an emergent behaviour which introduces new or refined levels of functionality. It requires a stringently defined reference architecture to enable the correct application of the relevant characteristics.

These are **qualitative** features. Therefore, an attempt at setting quantitative features to the EAS objectives was carried out, resulting in the table below:

EAS Qualitative Features	EAS Quantitative Features
Evolvability-conformity	Skills repository & Management
Plugability- control specifications	Module description/blueprint
Plugability- user requirements	Application guidelines
Safety conformity	CE & safety certification procedures
Evolvability & Safety	Rules related to emergent behaviour
Plugability-practical implementation	EAS "wrapper" solution: hardware
Evolvability-practical impl.	EAS "wrapper" solution: software

At the highest level the EAS application will require some form of virtual repository that stores the protocols, guidelines, and other support structures for users to be able to comply with its specifications. The Reference Architecture will be available through this level. Plugability is not only the required control structures and approaches applied to exchange and adapt data, but it must also indicate for external users what may be required in order to comply. Emergence, defined earlier, also implies that unwanted characteristics may emerge, and raise safety or liability issues. This will have to be considered carefully, and specific rules will need to be enforced. One of the major keys to a successful EAS application resides in how the modules will interact with one another, which implies that much more than traditional interfaces will be required. These are, for the time being, defined as "wrappers", which also implies that legacy components may be adapted to the EAS format. Finally, but also fundamentally, EAS requires an extremely well defined Reference Architecture. This architecture will be a mirror of the ontology developed for a particular class of products. This Reference Architecture is currently being developed out of the ontology definition that will be given in the next section.

3.2 Ontologies

Ontology is a concept widely used today in Knowledge Engineering, Artificial Intelligence and Computer Science in any application that involves knowledge management and information management. Its importance in other engineering domains such as mechanical engineering and assembly, in particular, is quite natural since this domain requires computer based supporting tools to help in the design and operation of agile assembly systems such as Evolvable Assembly Systems. The full development of the EAS concept cannot be done without design supporting tools and advanced control solutions whose main requirement is the ability to quickly change and adapt. However, this requires the assembly domain to be fully understood and modeled in a way that can be used by the different computerized tools involved within assembly. Only if the main concepts behind the assembly domain and how they interrelate are fully understood and conveniently represented using some kind of formalism it will be possible to create computer programs that support the development and operation of EAS. In addition to the central aspect of creating computer representations that model the concepts and their relationships it is fundamental that these representations are agreed by the assembly community because it must be taken into account that different tools will be used and developed across the domain and therefore it will be of great advantage if the tools could somehow reuse the models that are developed. These computer representations (models) can be defined using ontologies and this is the reason why they are being addressed here.

Ontologies are much more than simple taxonomies since they allow appending semantics to relations among entities. So, we may envision relations that describe dynamics, social interactions, etc. However, much work needs to be done in knowledge modelling and knowledge management, which should not be a surprise since ontologies are about representing knowledge. For knowledge management existent methodologies, like CommonKADS (Schreiber et al. 1994) may be exploited. But the great challenge is about knowledge modelling, since the approach

for ontology creation needs always knowledge modelling both for the situations in which the ontology is created using a top-down or bottom-up approach. One challenge is modelling operational knowledge, which is not clearly stated in products' data sheets. For instance, two modules can be mechanically interconnected but practical experience may indicate that such a connection is usually ineffective as mechanical stress is imposed, which in turn it is application dependent. The problems are increased if the knowledge of several experts needs to be merged, if the knowledge is contradictory, etc. The ontology research work about creating ontologies tries to provide answers to this question (Gómez-Pérez et al. 2004).

Some steps are being done in the direction of structuring the assembly domain in the European project EUPASS, in which some preliminary ontologies about the assembly process are being done using OWL. The most important concepts that are necessary in defining a precision assembly ontology to implement successful EAS must include the following sub-ontologies: (1) ontology of modules, (2) ontology of processes, (3) ontology of skills, and (4) ontology of products.

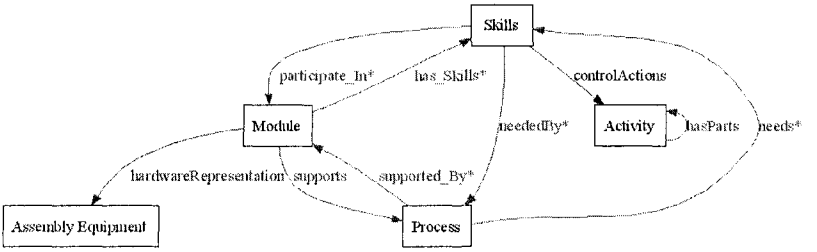

Figure 1 – Most important concepts and how they interrelate

An ontology of modules should identify the modules available to be used in coalitions (assembly systems). These modules can range from simple entities such as a gripper to a very complex entity such as an assembly cell. The ontology of processes should identify all the classes of processes that are used in assembly such as gluing, pick&place, joining, etc. The ontology of skills needs to formalise the skills that may be found in assembly. Not only is it necessary to define the individual skills associated to each module but also the skills that emerge out of the basic ones. The ontology of products should identify families of products that share certain assembly processes. These concepts grouped as sub ontologies are related according to figure 1. In figure 2 it is shown just part of the taxonomy of the ongoing ontology work that describes the assembly process.

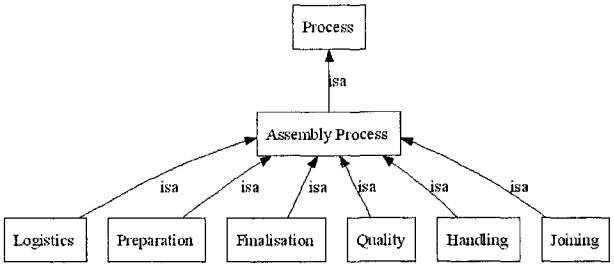

Figure 2 – Part of the taxonomy of the assembly process

4. CONCLUSION

Results are currently being published (Barata et al. 2006), and include methodologies for selecting modules, ontologies, and Reference Architectures. This article has focussed on the underlying principles. This is important since a new paradigm is taking form. As stated earlier, the underlying foundation of EAS is evolvability. Evolvability is, in rough terms, the most advanced form of adaptability. Hence it implies that the systems must adapt to changes. Changes may be either predictable or unpredictable in nature. Changes may occur at internal system level or external gobal level. Unpredictable behaviours that may affect EAS systems are termed as "emergent" and can be either great opportunities or extremely disruptive. A solution that cannot even attempt to tackle emergent events cannot be deemed as Evolvable. Therefore, the work is now focusing on Reference Architectures and validation cases.

5. ACKNOWLEDGEMENTS

This work has been partially done with the support of the European Commission through the Integrated Project EUPASS. The authors would also like to thank all their colleagues in the EUPASS project.

6. REFERENCES

1. Alsterman, H., Barata, J., and Onori, M. (2004). "Evolvable Assembly Systems Platforms: Opportunities and Requirements." Intelligent Manipulation and Grasping, R. Molfino, ed., IMG'2004, Genova, 18-23.
2. Barata, J. (2005). *Coalition Based Approach For ShopFloor Agility*, Edições Orion, Amadora - Lisboa.
3. Barata, J., Camarinha-Matos, L. M., and Onori, M. "A Multiagent Based Control Approach for Evolvable Assembly Systems." *INDIN 05 - 3rd International IEEE Conference on Industrial Informatics*, Perth - Australia.
4. Barata, J., Onori, M., and Frei, R. "Applying Evolvable Assembly Systems." *ISIE'06 - IEEE International Symposium on Industrial Electronics*, Montreal - Canada.
5. Castelfranchi, C., Micelli, M., and Cesta, A. (1992). "Dependence Relations Among Autonomous Agents." Decentralized A.I. 3, E. Werner and Y. Demazeau, eds., Elsevier Science Publishers B. V, Amsterdam, NL, 215-227.
6. Gómez-Pérez, A., López, M. F., and Corcho, O. (2004). *Ontological Engineering*, Springer-Verlag, London.
7. Kirsten, M. (2002). "Present: Trends in the Western European Electronics ", Reed Electronics Research.
8. Klusch, M., and Gerber, A. (2002). "Dynamic Coalition Formation Among Rational Agents." *IEEE Intelligent Systems*, 17(3), 42-47.
9. Onori, M. "Evolvable Assembly Systems - A New Paradigm?" *International Symposium on Robotics*, Stockholm, Sweden.
10. Pechoucek, M., Marik, V., and Bárta, J. (2002). "A Knowledge-based Approach to Coalition Formation." *IEEE Intelligent Systems*, 17(3), 17-25.
11. Schreiber, A. T., Wielinga, B. J., De Hoog, R., Akkermans, J. M., and Van de Velde, W. (1994). "CommonKADS: A Comprehensive Methodology for KBS Development." *Ieee Intelligent Systems & Their Applications*, 9(6), 28-37.
12. Shehory, O., and Kraus, S. (1995). "Coalition Formation among Autonomous Agents: Strategies and Complexity." From Reaction to Cognition, C. Castelfranchi and J. P. Muller, eds., Springer-Verlag, Heidelberg, 57-72.