Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Towards a Fault Tolerant Manufacturing Control System

Sven-Arne Andréasson

Department of Computer Science and Engineering
Chalmers University of Technology
S-412 96 Göteborg, SWEDEN
e-mail: andreasson@cs.chalmers.se

Abstract: The CHAMP (Chalmers Architecture and Methodology for Flexible Production) system is a general control system for manufacturing that can be configured for arbitrary production. The system consists of producers that can perform operations on products and movers that can move products between producers. Each product is described by a number of operations that are mapped to global operations in the database. A producer can perform any global operation for which it has a program. The mappings of global operations, programs and producers are also present in the database. Since a global operation can be performed by more than one producer there is an inherent possibility for fault tolerance. In the product description variations can be described which also increases the fault tolerance. The system can be run offline where the production is simulated. Thus the control system that is tested by simulation is the real control system.

1. INTRODUCTION

To achieve a dependable manufacturing control system it must be fault tolerant. This due to that mechanical problems must be cooped with since they are unavoidable. The machinery will be ageing and eventually break down. Thus the control system must be flexible enough to handle different configurations and avoid operations that can not be performed. It is also desirable that the system can automatically back out of a blind alley due to the emergence of some mechanical or software problem. Anyhow the control system must be helpful to ease manual recovery.

The CHAMP (Chalmers Architecture and Methodology for Flexible Production) system is developed to fulfill the demands for fault tolerant manufacturing. At present it is a model for controlling one entity, e.g. a manufacturing cell, but is now being developed further to encompass an entire production. The model is seen as an "operating system" for production which means that the products are described in recipes with operations which will be scheduled to the different producers. Thus introducing new products and/or machinery will not need any change in the control system. Only new mappings in the database and corresponding basic producer programs, e.g. PLC programs, have to be developed. This can be added to the control system even when it is running. The scheduling, dispatching and handshake protocols are given and implemented once for all. Different scheduling algorithms can be introduced separately, but they only change the suggestions for which operation to choose from all eligible. The algorithm that decide what operations are eligible is hardcoded into the system.

The present model has been implemented in Java using a relational database.

2. THE CHAMP ARCHITECTURE

The present version of CHAMP is focused on the cell level and admits manual, semi-automatic as well as fully automatic control of discrete-event manufacturing processes (Adlemo et. al. 1995). It consists of a Dispatcher that sends orders to the Resources after getting production suggestions from a Scheduler, see Fig. 1. They all share information in a common database.

2.1 The CHAMP Data Design

The production is described as recipes for products stored in the database. The recipe gives the different operations that should be performed to make the product and in which order they must be done if necessary. The recipe might give different choices of operations or groups of operations. It can also be controlled by two different type of flags, one that give different choices when a product recipe is compiled (when an individual product is entering the system) another that is valid and can be changed until a choice is actually made at runtime. In the recipe it is also possible to describe what can be executed in parallel or what can be executed in any order.

The production units, Resources, are also described in the database. A Resource can be a Producer, a Mover, an Inbuffer or an Outbuffer. All Resources though can act as a Producer. For the resource to be able to perform an operation it must have a corresponding program to execute. To be able to find a resource for an operation these programs are mapped to the resource in the database. A program for an operation is unique

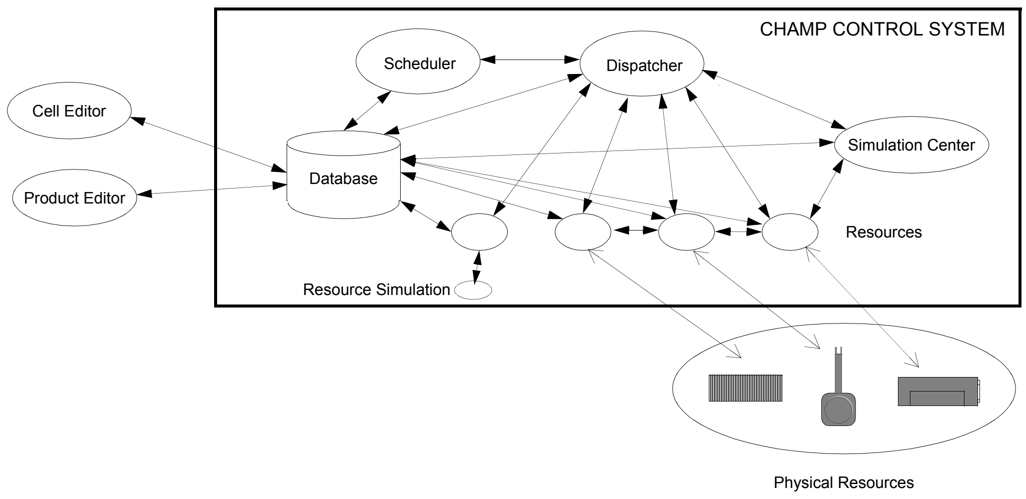

Fig. 1. The CHAMP control system architecture.

for an individual resource since many of these programs must be continually trimmed to its resource as the resource is wearing out. There are also programs for the different movers to move the products between the resources. For each operation there is also registered a variable number of handshake steps for how to load and unload the product. These are also mapped to programs for corresponding resources and movers. For each program there is also registered which tools it might need and for each resource which tools are available. A conceptual model of the database is given in Fig. 2. When a product arrives an individual record is created for it from the corresponding template giving the operations. Then for each operation a resources that has a program for it can perform it. The Scheduler algorithm decides which of these that might be performed at each moment from the template recipe. The recipe gives the

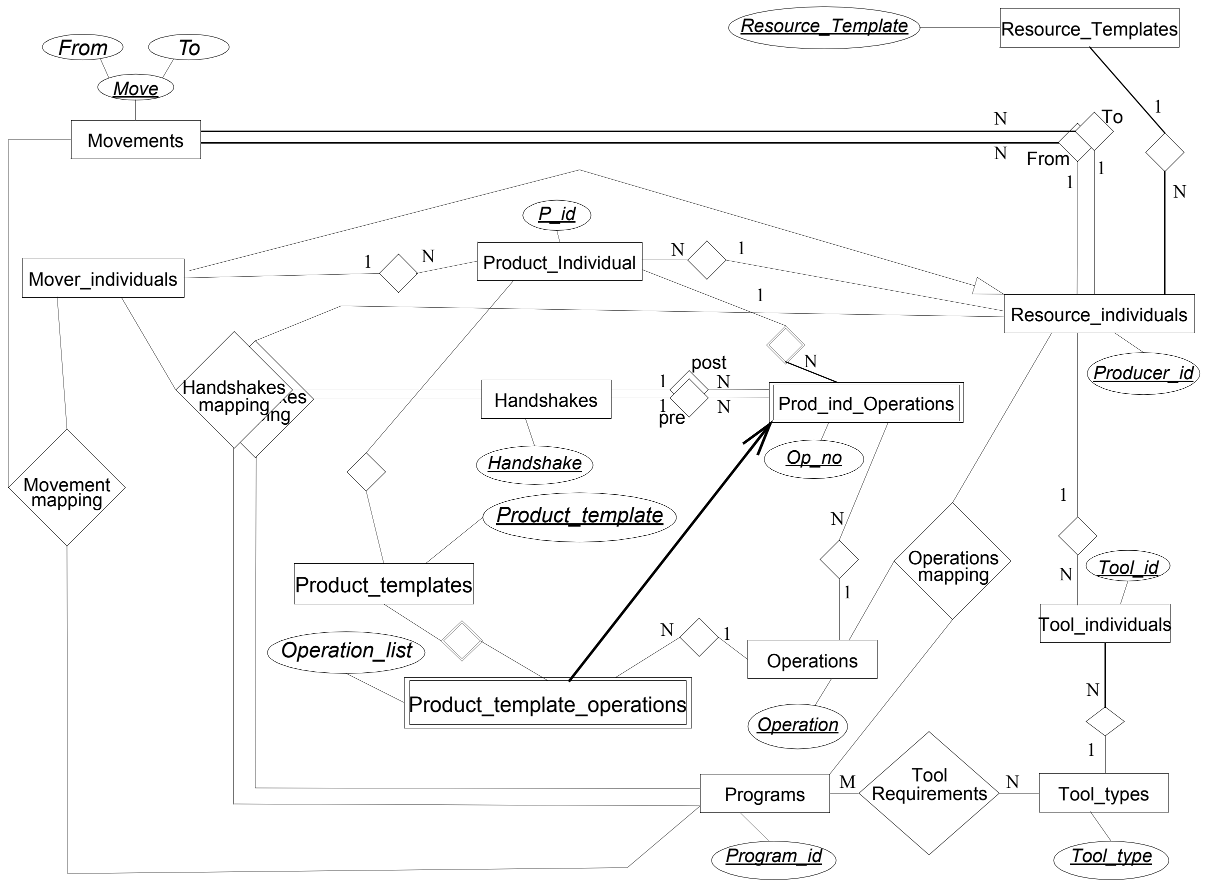

Fig. 2. Main part of the Entity-Relationship (ER) schema of the conceptual design of the database. Only entities, relationships and their key attributes are given.

necessary order of operations and/or possible parallel execution.

2.2 *The CHAMP Control System*

The implementation of the CHAMP model consists of different programs that all use the database (Andréasson 2004). The resources are represented by Resource programs which are of four types, Producer, Mover, Inbuffer and Outbuffer. The last three types can also act as a Producer. The cell control is performed by a Dispatcher program that sends commands to the different Resource programs. When performing the handshakes between movers and resources it is performed by using local commands between the Mover and Resource programs following protocols that can be different for each type of delivery. A Scheduler program sends information about possible operations to the Dispatcher program according to product recipes and possible operation mappings. It is possible to load separate scheduling algorithms into the Scheduler program to give different priorities to the suggestions in order to optimize production. From the list it gets from the Scheduler the Dispatcher program normally chooses the first operation that can be performed by an unoccupied Resource and for which there is a Mover free that is able to transport the corresponding product to that Resource. There is also manual mode where an operator chooses from the given list.

In order to perform simulation there is also a Simulation Client that can simulate an external system sending products to the manufacturing cell. This can be done in two ways, by unannounced deliveries to an Inbuffer or by deliveries to any Resource after announcing the product to the Dispatcher which appoints the receiving Resource after consulting the Scheduler. In addition to these programs there is also a Product Editor which can be used for defining products in the database and a Cell Editor that is used for entering cell data into the database. A program for diagnosing and fixing errors is also underway.

3. FAULT TOLERANCE STRATEGIES

3.1 *Configuring a Manufacturing System*

For a manufacturing system to be fault tolerant it must be able to perform a reconfiguration in case of a failure within the system. To perform such a reconfiguration, the system must be able to react to spontaneous events such as resource failure, tool breakdown, or product jam.

We have identified three different types of configuration; the *Hardware Configuration*, the *Mission Configuration* and the *Work Configuration* (Adlemo and Andréasson 1992, Adlemo et. al. 1993):

- *Hardware Configuration* can be represented as a tree showing how the Resources are grouped, e.g. in production cells. *Hardware Reconfiguration* means to change this tree, e.g. moving a Resource from a cell to another cell.
- *Mission Configuration* can be represented as a tree representation of a product recipe. This shows how operations can be divided into sub recipes as well as which order they must follow. A *Mission Reconfiguration* is to use alternatives within the recipe to perform the production.
- *Work Configuration* is the way the control system has decided to perform the operations. The operations are mapped to corresponding resources that should perform the execution. *Work Reconfiguration* is to reallocate the mapping for one or more operations.

3.2 *Fault Tolerance in the existing system*

Work Configuration: Since the existing system allocates which resource to perform a given operation just before it is to be performed there will not be any allocation to a faulty resource. Thus there will be fault tolerance if there are another resource that can perform the operation, i.e. it has a program for the execution of the operation. If the error occurs during the execution of the operation, then the product must be unloaded in order to be rescheduled. For this we have introduced *Exceptions* in the recipes. These can be used for describing the unloading handshake. This handshake might be different than the handshake after a successful execution. The exception might include a measuring or check operation that decides if or how to continue. The product is then rescheduled the normal way and if there is another Resource that can perform the operation the production can go on. Otherwise the product must be taken aside until the operation can be performed.

Mission Configuration: The possibility to describe variants within a recipe can be seen as having mission reconfiguration. Also here the actual choice can be postponed until just before the specific operation execution. Thus faulty resources can be avoided. Exceptions can also be used here to start a variant of the recipe.

Hardware Configuration: Since we are controlling only one entity with the present system there is no need for automatic hardware reconfiguration. A way to do hardware reconfiguration though is to introduce new resources. However, this must be done manually by registering them in the database with their operation mappings. For a new resource there must also be installed a driver program to link it to the control system.

4. HIERARCHICAL ARCHITECTURE

4.1 *Reasons for a hierarchical architecture*

Since the production units will be clustered at least on different factories or sub companies we believe that it is wise to structure the resources hierarchically. Also the products are naturally structured as trees since we have operations that contains sub operations. And the production units and product operations have to be mapped to one another.

We are aware that hierarchical systems are not always popular. Recent systems tends to be modeled according to heterarchical

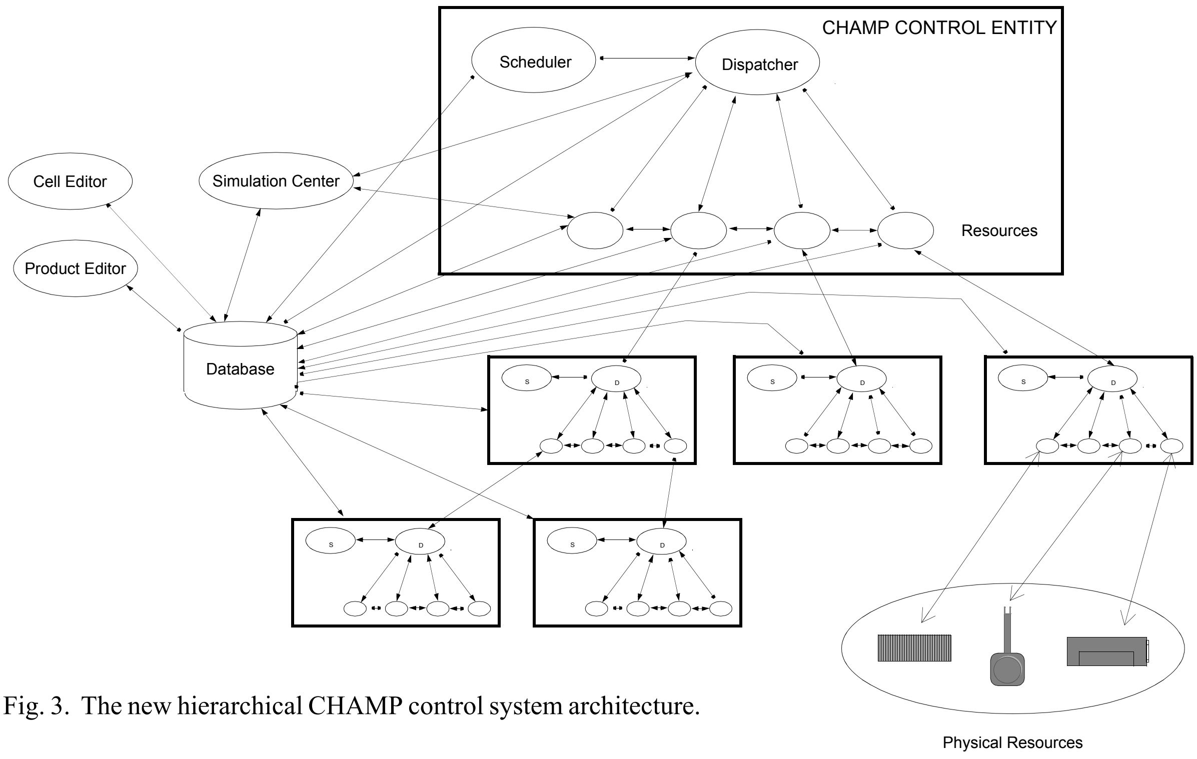

Fig. 3. The new hierarchical CHAMP control system architecture.

principles as the *Holonic* approach (Morel et.al. 2007, Wullink et. al. 2002, Valckenaers and van Brussel 2005). Having a hierarchical model, though, will give a clear responsibility chain for the production and thus the time out alarms when an operation is not ready when expected. Also there might be a choice between a local Resource or on another continent. Such a choice should be treated differently (e.g. with a corporate level scheduling algorithm) than a choice between local Resources (e.g. with a manufacturing cell level scheduling algorithm).

However, although the main structure is hierarchic, the emphasis of the architecture is to keep interactions as well as information at lowest possible level. Thus the upper level of our existing system is not involved at all in handshakes and their states and messages. The handshakes are initiated by the movers directly to corresponding producers. For the proposed architecture this will still be case. Upper levels only give order about a complete operation and then is informed when it is ready or if it has problems. Upper levels also have to keep watch that lower level operations terminates in due time.

4.2 Proposed system architecture

The main design principle for the hierarchical architecture is to use the same architecture for each level. For a manufacturing system this means that the Cell Control System is the same as the Area Control System (controlling products between cells) and the Factory Control System (controlling products between areas). Also the same control system should be used for each entity (node) within the same level. Hence we will call this control system the Entity Control System. Obviously there is a different need for different type of scheduling for different levels and/or different entities at the same level but this is achieved by choosing different scheduling algorithms in the Scheduler. The present implementation actually allows new scheduling algorithms to be written and plugged into and used by the Scheduler even during runtime.

The resources are added to the system by introducing a driver program for it in the system or instead an adapter in the resource. Thus the system can also be introduced partially in an existing plant by writing adapters to existing control systems.

The hierarchy will be achieved by adding a drive routine in the Resource program that allows the Dispatcher program of the level below to act as a Physical Resource. The Dispatcher part of this connection already exists and is used by the Simulation Client. The Scheduler program doesn't have to be changed.

Now the easy way of achieving the hierarchical system would be to implement the drive routine for the dispatcher and then build it from a number of the existing system using separate databases (i.e. separate database accounts or catalogs) where the data for each entity is given in the corresponding database. However, there are drawbacks to this approach. Since movers will have to cross entity boundaries to deliver products the physical position for the deliveries must be known globally. When using different databases many physical positions then have to be registered into more than one database and this will be a source of errors. Another drawback with this approach is that the movers must have access to different databases in order to find the Resources login ports in order to be able to open connection for handshakes. This will lead to problems with database login names and passwords, since the movers must be able to login to many database accounts. A database that offers proper catalogs as MySql could deal with this problem but oth-

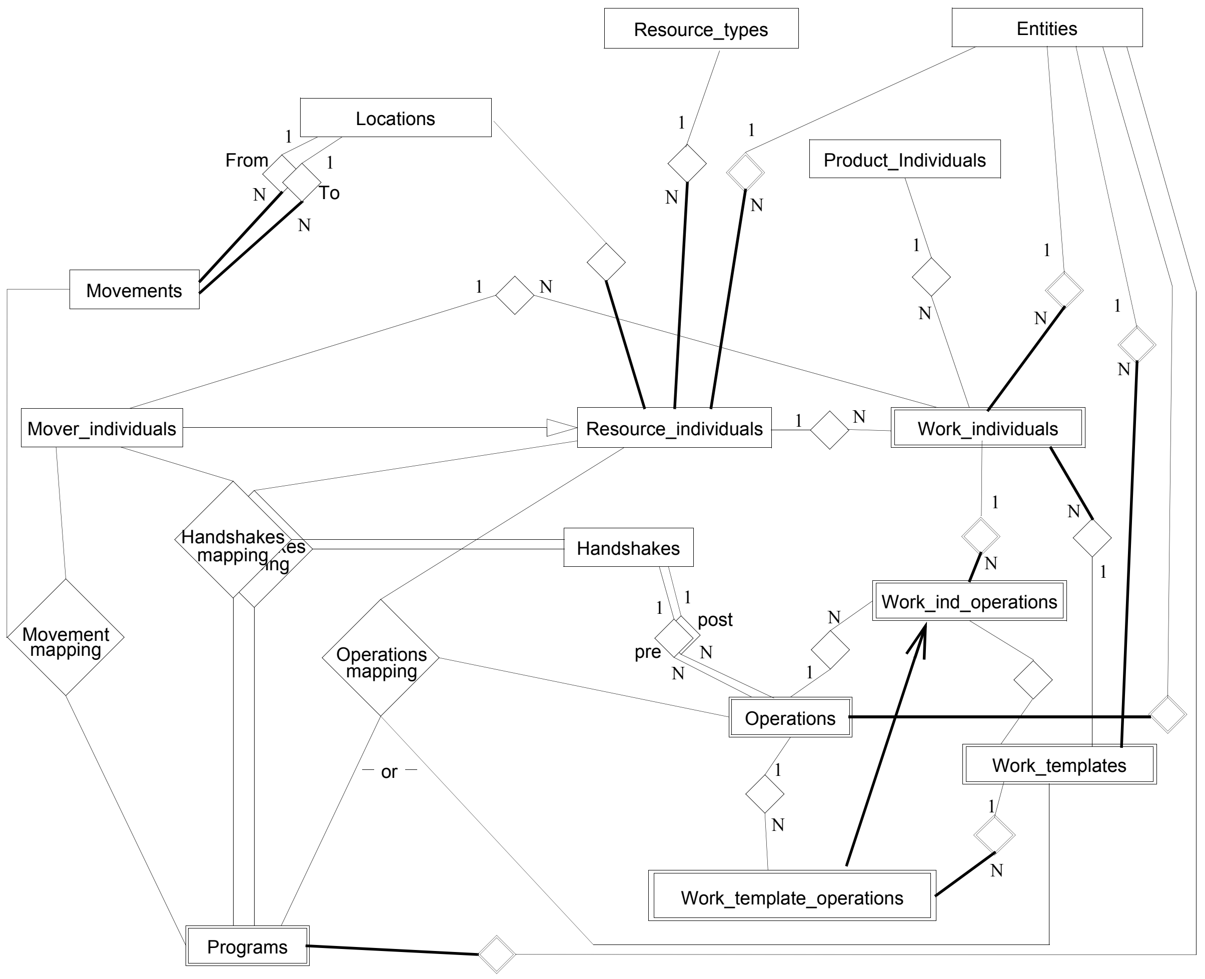

Fig. 4. Main part of the Entity-Relationship (ER) schema of the proposed conceptual design of the database. Only entities and relationships are given.

er databases such as Oracle can not handle this. Our aim is to use a strategy that can be implemented by using most of the existing commercial databases.

The chosen strategy then instead is to use the same database (account) for the entire system. The overall architecture then will be like in Fig. 3. This implies a redesign of the database that will inflict all programs in the system. However, the advantage of this approach is big enough to justify this. Especially since the system already is designed with a common library that implements proxies for the database tables.

The question now is to decide which data that is to be treated as local for an entity and what is to be treated as globally known. As already stated the physical positions have to be globally known. Since the delivering of the products will be done by leaf entities the handshake also have to be found globally since there has to be agreement on handshakes between different entities.
To be able to perform reconfiguration for fault tolerance it is also important that the operations are treated globally.

In the present system a Product Template represents a product type and is used for defining what to do with each product individual. In the hierarchical system a Work Template in an entity will represent an operation in the entity above. This applies for a change in the mapping tables to be able to map an operation to a Work Template in a lower entity rather than a specific program. Then the name of the database tables should be changed to more intuitive names. Whether a global identity for the product should be recorded at each level can be disputed but it should however not be made compulsory since we might want to produce parts that are not dedicated for a specific individual from the beginning.

The proposed conceptual design for the hierarchical system is given in Fig. 4. The local entities are marked as weak entities, i.e. each record must be coupled to an Entity record to be unique. In this way the production planning for each entity can be performed locally without having to consider identities in other system Entities. The global known data is recorded in the entities: Locations, Entities, Operations, Movements, Handshakes and Product Individuals. For these entities the identification must be globally unique. The entities for tools are left out for diagram readability reasons. They can be easily added according to the present system but will subsequently be weak entities.

Grouping the Resources with different Locations into Entities gives possibility to fully utilize the parallelism given in the recepis. A product might be operated on by several robots at the same time while located in an Entity, e.g. robots performing soldering.

There are also data for product flags and operation parameters that are left out in the diagram.

5. FAULT TOLERANCE IN THE PROPOSED SYSTEM

For the proposed architecture there will be a considerably potential for fault tolerance. The most important features to achieve this is the use of global Operations and global Locations. When an operation can not be performed at one entity there might exist another entity that can execute it. This might be solved automatically due to the global operation identification. Also when finding other entities and sub entities the global location identification is useful.

- *Hardware Reconfiguration* will be applicable in the proposed system. As an example a producer might logically be moved from one production cell to another in order to replace a faulty unit or give extra execution power to avoid a bottleneck. However, this demands that the production cells are reasonable close to each other.
  To move a sub entity logically from one super entity to another means that it will obey the latter Dispatcher program instead of the first. There will be no need for changing Location or Operation data in the database, only the sub entity super entity relation.
- *Mission Reconfiguration* can be performed among the different entities at different levels. This will demand scheduling algorithms that use information from levels below in the hierarchy. The scheduling algorithms, especially on the upper levels, must take into account the cost of moving the product. A move might be across continents! The upper level algorithms might include manual acknowledgement before taking costly actions.
- *Work Reconfiguration* can also be done at different levels. This will also require scheduling algorithms that use information from deeper levels. Actually these will be required anyway, since the scheduling algorithms must look at deeper levels for performance reasons. This has not been a big issue in the present system, the most advanced have been to schedule randomly among possible choices.
  As for the Mission Reconfiguration, the cost of product movement must be taken into account before taking action.

The delegation of control down in the hierarchy also provides for fault tolerance. The upper level gives order about the execution of an operation as well as the product transportation and then only waits for a completion message or if not, an error message. If nothing is received within a time limit an exception process will start to try reconfiguration. This is also performed when an error is reported from the level below.
But mostly problems are avoided at first configuration of an operation since all faulty units will be excluded from the beginning.

6. CONCLUSION

We have described the new proposed hierarchical design of the Champ system. It is shown that it will offer a high potential for fault tolerance by using the concepts of Hardware Reconfiguration, Mission Reconfiguration and Work Reconfiguration. This will be practically possible by the use of global operation identities and global location identities.

7. REFERENCES

Adlemo, A., Andréasson, S.-A. (1992). Models for Fault Tolerance in Manufacturing Systems, Journal of Intelligent Manufacturing. vol. 3, no. 1, February 1992, pp. 1-10.

Adlemo, A., Andréasson, S.-A., Johansson, M. I. (1993). Fault Tolerance Strategies in an Existing FMS Installation. Control Engineering Practice, vol. 1, no. 1, February 1993, pp. 127-134.

Adlemo, A., Andréasson, S.-A., Fabian, M., Gullander, P., Lennartsson, B. (1995). Towards a Truly Flexible Manufacturing System. Control Engineering Practice, 3, pp. 545-554.

Andréasson, S.-A. (2004). Implementation of the CHAMP System. Real-Time Programming 2004, pp. 33 - 138. ISBN/ISSN: ISBN: 0-08-044582-9 CPL 12478.

Morel, G., Valckenaers, P., Faure, J.-M., Pereira, C. E., Diedrich, C. (2007). Manufacturing plant control challenges and issues. Control Engineering Practice (2007), doi:10.1016/j.conengprac.2007.05.005.

Wullink, G., Giebels M.M.T., Kals, H.J.J.(2002). A system architecture for holonic manufacturing planning and control(EtoPlan). Robotics and Computer Integrated Manufacturing 18 (2002) 313–318.

Valckenaers, P., van Brussel, H. (2005). Holonic manufacturing execution systems. 55th General Assembly of CIRP, Antalya, Turkey, August 21-27, 2005.