A Software Architecture for Industrial Automation

Technical Report IC/2003/28

Rodrigo García García
Software Engineering Laboratory
Swiss Federal Institute of
Technology Lausanne (EPFL)
Ecublens
CH-1015 Lausanne, Switzerland
E-mail: rodrigo.garcia@epfl.ch

Esther Gelle
Information Technologies Dept.
ABB Switzerland Ltd
Corporate Research
Segelhof 1
CH-5405 Baden-Dättwil
E-mail: esther.gelle@ch.abb.com

Alfred Strohmeier
Software Engineering Laboratory
Swiss Federal Institute of
Technology Lausanne (EPFL)
Ecublens
CH-1015 Lausanne, Switzerland
E-mail: alfred.strohmeier@epfl.ch

Abstract

The Aspect Integrator Platform (AIP) from ABB was designed to build the next generation of industrial automation applications. This platform is part of a set of products that provide the means to model, control and supervise continuous or discrete processes in various market domains, ranging from chemical and metal to paper and consumer industries. Each product works at a different level in the manufacture process, having distinct safety and real time requirements, but all of them rely on a common architecture for interoperability. The architecture proposes a set of components that can be reused in the different products. The current implementation of the architecture provides considerable flexibility in terms of modelling domain information and dynamically modifying it at run-time. On the one hand, this is a feature required by applications that must run 24 hours a day. On the other hand, this flexibility adds complexity to the maintenance of the installed application because dependencies among its components change dynamically. In this paper, we study the different kind of dependencies that can arise between components and show them in the context of an example from automotive industry. We then show how dependency tracking and consistency among components can be improved by representing them in XML, thanks to the structuring and validation properties of XML Schemas. Finally, we also outline the advantages that the use of XML would provide to future developments of the platform in the areas of data manipulation, transmission and storage.

A Software Architecture for Industrial Automation

Technical Report IC/2003/28

I. Introduction

The ABB automation platform, named Aspect Integrator Platform (AIP), is designed to be the future platform for industrial automation applications. This platform is part of a set of products that provide a process control system which is used to control continuous or discrete processes in various market domains, ranging from industrial systems for power plants, utilities, pulp and paper, metals and minerals, chemicals and consumer industries. The main characteristics of these products and applications are reliability, compatibility and stability, in some cases safety and time critical functionality. The latter is extremely important since industrial processes are running 24 hours a day and it is often not possible to interrupt a process just to upgrade a software [1], [2]. Any changes on the software must be achieved without impact on the existing technical process. In general a technical process consists of several layers (Fig. 1), which are process, field, group control, process control, production, and enterprise management level [3]. The lowest level is the process level. It contains the devices that need to be controlled, e.g. a high voltage switch in a transmission network or a robot in a production cell. On the next level, the field level, sensors, actuators, and drives provide the interface between the physical process and the control system. On the group control level, controller devices and applications control a group of devices. These process controllers obey hard real-time constraints. On the process control level, operator stations and applications supervise and control the entire process. They gather and analyze process information. The production level comprises applications for production planning, for example manufacturing execution systems (MES), which analyze and optimize entire processes with respect to production. The enterprise management level finally comprises applications with transactional character to deal with enterprise resource planning (ERP), e.g. human resource planning, supply chain, order processing, and finance.

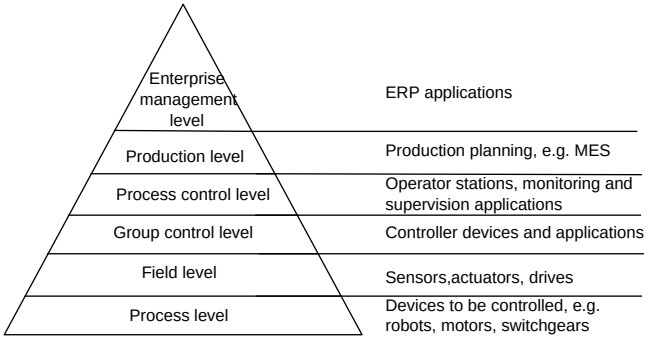

Fig. 1. A technical process with its layers process, field, group control, process control, production, and enterprise management.

The Aspect Integrator Platform, which we present in this paper, is concentrating on the process control level, implementing the functions of process control and supervision. The basic functionality of such an automation platform consists of process supervision, i.e. acquisition and presentation of information on the process status, local and remote control of the process, alarm and event handling, and history management, i.e. acquisition and presentation of information on the process history. It provides a graphical user interface to present the process status and history as well as control interface to connect to the devices. Since this basic functionality does not vary much across different market domains, the main motivation to build a software platform for a distributed automation system is to foster the idea of reuse by avoiding parallel developments in different businesses, harmonizing interfaces to third party applications, and focusing on product lines for different market segments [3].

The agreement on such a product structure that serves a family of products together with a basic set of components and the interaction that holds between them is commonly called a software architecture [4]. Such an architecture proposes a set of components that are reused in the different products. In addition, the current architecture provides considerable flexibility in terms of modelling domain information and dynamically modifying it during run-time. On the one hand this is a feature required by applications that run 24 hours a day. On the other hand, this flexibility requires the user to be careful in order to mantain the consistency of the installed application base. Since these changes occur dynamically and they are not explicitly tracked inside the platform, the user is responsible for taking into account the dependencies among the components. This becomes an issue when reusing existing components. The user might want to use a component in a completely different context and not be aware upon which other components it depends. In our paper we show how dependency tracking and consistency among components can be improved by representing the actual platform concepts in XML and using this definition to verify dependencies in an application specific context.

II. Concepts of the Aspect Integrator Platform

To satisfy the main requirements of a process control system, i.e. acquisition and presentation of information on the process status and its history and real time control of the process, the Aspect Integrator Platform (AIP) provides means for information representation and navigation as well as interfaces to connect to the actual process. The main AIP concepts that allow the realization of automation applications based on this platform are Aspect Objects and Object Types, Aspects and

Aspect Types implemented based on COM components, and hierarchical Structures.

Each Aspect Object can be identified within AIP through its global unique identifier (GUID).

Aspect Objects represent concrete or abstract individual concepts relevant for the system to be modeled, for example, plants, machines, devices, or algorithms. An Aspect Object serves as container for Aspects, which describe various characteristics, i.e. behavior and data, of the domain object. An Aspect Object may be instantiated from a template called Object Type. It defines the type and number of Aspects the Aspect Object instantiated from the type contains. It also provides a mechanism for defining an Aspect that is shared by all Aspect Objects of the Object Type. This notion is similar to the notion of class variable in Object Orientation. In the current platform, notion of Object Type is weak; an Aspect Object instantiated from an Object Type can be modified afterwards.

The architecture further provides a means for structuring the set of Aspect Objects pertaining to a controlled process into possibly several hierarchical Structures. The Structures represent dependencies between domain concepts and facilitate information navigation [3], [5]. An Aspect Object may appear as node at any level of a Structure in an arbitrary number of Structures and it stores all references to these Structures. Predefined engineering structures are for example the functional, the location-based, and the control structure. They describe the functional, the location-based or product-oriented viewpoint of a technical system and the networks and nodes to control the process. This concept of multiple hierarchical Structures is based on the standard IEC1346 [6]. It does however not fully comply to the standard for several reasons. First, Aspects instead of Aspect Objects should be the leaf nodes in Structures defining the specific characteristics (functional, location-based etc.). Secondly, there might be an n-to-n mapping between different structures [7]. A function might for example be implemented by a one or several production-oriented Aspects and vice-versa. The representation of such an n-to-n mapping however would make it extremely difficult to navigate between different Structures as stated in the standard.

Objects and their organization in hierarchical Structures can be manually defined in the Plant Explorer, an interactive interface for AIP. The Plant Explorer shows on the left hand side the current Structure selected and on the right hand side in the upper window all Aspects of the currently selected Aspect Object and below the graphical representation of the current Aspect (Fig. 3).

As an example consider a factory producing equipment for the automotive industry. It consists of a shop floor with a series of cells in which one or several robots are producing the equipment. The robots are controlled via specific control software. For each cell exists a daily plan of how much this cell should produce and how the individual robots are to be controlled. The goal is to monitor the entire floor to detect failures as early as possible and report on performance indicators. This scenario is modelled in the AIP environment as follows: the factory, the cells and the robots are represented by Aspect Objects that are organized hierarchically in a functional structure (Fig. 3). The cells and robot characteristics are defined once in the corresponding Object Types *RobotType* and *CellType* (Fig. 2). It is then easy to make use of the control structure in order to add information on the control nodes for each robot.

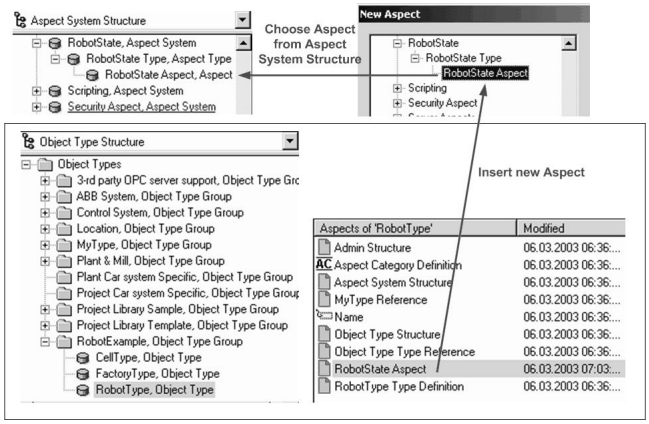

Fig. 2. The shop floor example with the cell and robot object types. Each Aspect Object instantiated from RobotType will contain the Aspect RobotState.

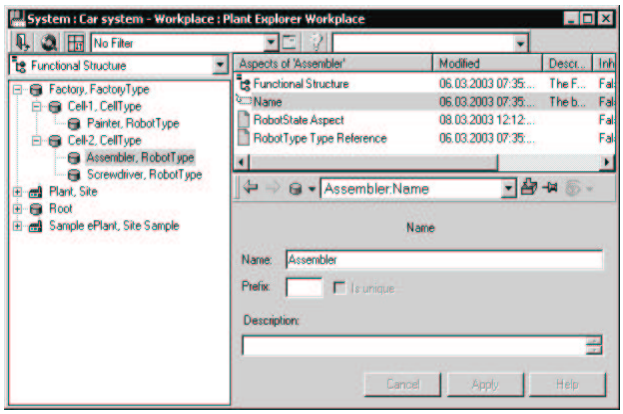

Fig. 3. The shop floor example with cells and robots. Each cell has a graphicalCell Aspect and each robot has a RobotState Aspect, which indicate their status.

The Aspect of an Aspect Object contains the data associated and the corresponding behavior and views (graphical user interface) which entirely describe the characteristics of the Aspect Object. The data and behavior are implemented through software components based on COM technology. In a way, Aspects can be compared to methods in the Object Oriented sense but in addition they also contain the data of an application (a function which is provided by the attributes class and its instances in Object Orientation). An Aspect can be located via the Aspect Object that holds a reference to it. An Aspect itself is purely conceptual and it comes only into existence through the fact that a set of components is registered with the AIP infrastructure. The basic set of components distributed with AIP contains functionality for alarming and event handling, historical data management and trending, viewing web/HTML pages, viewing PDFs, wrapping Win32 applications, and wrapping MS ActiveX components. The

latter is easily configured to encapsulate applications like MS Word and MS Excel. Aspect Types are beside Object Types the main concept for software reuse in AIP. They allow reuse of software components through the AIP infrastructure since in the registering process the components are linked to an Aspect Type to which the Aspect belongs. In addition, the Object Type can define specific Aspects that have to be present in an Aspect Object by defining the corresponding Aspect Types. In addition, Object Types can define the number of children Aspect Objects present in an Aspect Object (instantiation of composite objects).

Services provided by the platform include also an OPC service. It implements the acquisition and control of real time information. OPC is a series of standards specifications defined by the standardization group OPC Foundation [8] dealing with the acquisition of process data, alarm & events, historical data, and batch data.

Returning to the robot example, we require a cell to indicate when one of the robots has a failure. This can be achieved by graphically representing the cell as a rectangle that is green as long as all robots in it are active and that turns red otherwise. Aspects are the concepts for realizing this behavior of the cells and robots. An Aspect called *graphicalCell* with its *grapicalCellType* is implemented for a cell with a property called state to represent the rectangle that changes its color according to its state. A second Aspect *RobotState* with its *robotStateType* is implemented for a robot that also has a property indicating the state of the robot. Its value is retrieved via an OPC interface from the control network that actually controls the robot. Once this variable is set for each robot of a cell the *graphicalCell* Aspect can determine its color depending on the state of all its robots. In our robot example, the COM components implementing the *graphicalCell* and the *RobotState* are registered with the AIP infrastructure. The *grahpicalCell* is linked to an Aspect Type *GraphicalCellType*. The Object Type *RobotType* comprises the Aspect Type *robotStateType* and the Object Type *CellType* holds the Aspect Type *graphicalCellType*. In this example, each *RobotState* is representing the status of an individual robot, therefore each robot requires an instance of this Aspect.

More than one software component can cooperate to implement an Aspect Type. The separation of functionality into user interface, business logic, and persistency enforces decoupling as a prerequisite for reuse and it leads to a more robust design since different parts can be reimplemented separately. The implementation code of a component can make use of AIP infrastructure functionality to navigate to and access software components of other Aspects. This navigation is made possible by the ABB Automation Model, which provides a way to navigate through the Structures to reach the leaves in which Aspect Objects and their Aspects are located. This allows an AIP developer create dependencies between AIP concepts in the source code of the program and also to create or delete instances on the fly. In the robot example the *graphicCell* Aspect uses the ABB Automation model to gather the state of all of its robots through calling the RobotState Aspects.

AIP supports flexible modelling of user-specific Structures since everything can be modelled manually in the Plant Explorer while the system is running. There is a tradeoff beween the aforementioned flexibility and the maintenance of consistency between the different AIP concepts in a project expecially if the domain model contains hundreds of objects (not uncommon in typical ABB applications). In the next section we show how this issue can be tackled without changing the architecture fundamentally.

III. XML REPRESENTATION FOR AIP

In this chapter, we present the original motivation that led us to design an XML representation for AIP. Later on, we show how XML could be used to improve the platform integration at different levels: data, communication and even graphical level.

A. Dependencies between AIP concepts

The different components that build up an AIP project usually have interdependencies among them. We can classify these dependencies according to the following criteria:

- Object instantiation: An Aspect Object is an instance of an Object Type.
- Aspect instantiation: An Aspect depends on its Aspect Type.
- An Aspect Object depends on the Aspects it contains.
- An Object Type depends on the Aspect Types it designates.
- Inheritance: An Object Type can be a subtype of another.
- An Aspect can call a function or use the properties of another Aspect during execution.

The first four types of dependencies are summarized in Fig. 4. These four dependencies, together with the fifth one (inheritance) are *structural* dependencies. By structural, we mean that a component cannot lie in a project if the components on which it depends are not present. For instance, an Aspect Object cannot be correctly instantiated without its Object Type. This is important when installing the components of an existent AIP project into a new system, i.e. exporting the components from one project to another. Structural dependencies affect the way components must be loaded in the platform, since AIP does not support forward references. This means, for example, that an Object Type must be actually present in a system before any of the Aspect Objects of this type is loaded.

Object Types have advanced features that allow a finer grained control of instantiations than that of the classes we can find in usual object oriented languages. Apart from the Aspects it will contain, we can control, if we want to create a composite type, the type and the number of children that an Aspect Object will have. In this way, an Aspect Object has a stronger dependency on its Object Type than just simple instantiation.

The last kind of dependency affects only run-time behavior, we call it therefore a *run-time* dependency. Contrary to what it happens with structural dependencies, the loading order in a system is not important for components with run-time dependencies. The Aspect that has a run-time dependency on other Aspects can be loaded in any order with respect to these latter. Nevertheless, if they are not present when the Aspect is

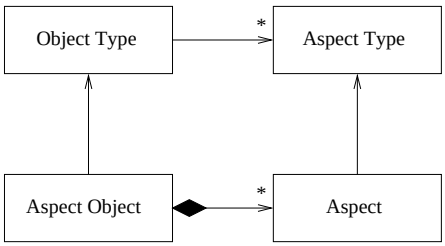

Fig. 4. Direct dependencies of an Aspect Object.

activated, a run-time error will be raised. An Aspect can access the exposed properties and operations of any other Aspect in the system once it has localized it in the system Structures. The path from one Aspect to another is usually stated in the source code of the caller Aspect. That is why this kind of dependency could only be detected by code inspection and, consequently, it is harder to detect than structural dependencies.

As it is the user who must keep track of component dependencies currently with AIP, we decided to implement a tool that could automatically extract the set of structural dependencies for a given component and represent them graphically and that extracts the AIP concepts in a strict order. This can then be used to correctly import and export an application for example. XML was chosen as the intermediate language for stating these dependencies for several reasons:

- XML is a publicly available standard [9].
- XML is platform independent.
- XML allows structured content.
- XML Schemas allow the validation of XML documents.
- A great number of programs and APIs allow the easy creation and manipulation of XML data.

Moreover, XML has mechanisms that allow the identification of single entities within a document. Since each AIP component (as any COM component) has a unique identifier, these mechanisms were the ones used to reference already defined components in the document and detect dependencies. Besides, XML documents present a tree structure suitable for representing the aspect structures defined in the IEC 61346 [6] standard, which are also organized in the form of a tree.

Once the XML document is completed, the dependency information is passed to a visualization tool that displays a dependency graph.

B. Technology evolution

The advent of the Microsoft .NET initiative [10] in June 2000 implied a major impact for all AIP related development. As we have seen in previous chapters, AIP heavily relies on Microsoft COM for its component infrastructure. On the other hand, the .NET platform proposes a new technology for component based development, where the basic unit of reuse is the *Assembly* [11]. Assemblies are no longer compiled to native binary code, but to an intermediate language called MSIL (Microsoft Intermediate Language) that is platform independent (much like Java byte code or Pascal p-code). But what is really important in .NET for us, is that it uses XML extensively for different kind of applications, introducing the concept of XML web services. The XML representation of the platform could be used for more than just stating the dependencies among AIP components.

Moreover, now that COM is a legacy technology, it is probable that the platform will eventually evolve towards a .NET implementation.

C. OPC evolution

The other technology that is closely related to AIP and COM is OPC (which originally stood for OLE for Process Control [8]). OPC was designed as the standard way to communicate plant devices with Windows based applications (or any other system supporting COM). The main advantage of this standard is that it provides a consistent way for accessing the data produced by any device in the plant floor. There is no need to deal with the specifics of different proprietary device drivers as long as the hardware provides an OPC interface. Although OPC allows to specify time parameters for reading data (such as the updating period) and it also provides methods to modify the variables exported by the hardware, it was not conceived for critical real-time control. The time constraints specified should be regarded as guidelines and not as absolute values, since OPC is run under a best effort basis. Nevertheless, it provides a simple and efficient way to retrieve plant information in a timely manner, quick enough for human interaction.

OPC was built upon OLE/COM and it is still closely related to these Microsoft technologies. Some vendors of OPC products and the OPC Foundation have realized that there is a natural evolution of OPC towards .NET. Vendors have started developing .NET assemblies that wrap the functions provided by legacy OPC products and the OPC Foundation has been working on the specification of a new standard: OPC XML-DA. This standard is the evolution of the traditional OPC Data Access standard, which describe how to access to the variables exposed by the plant controllers. The future standard will use XML/SOAP for communication purposes and thus, it will not require any COM knowledge, which is good news for those platforms that do not have native COM support.

IV. Advantages

XML standard nature has many benefits regarding data manipulation, transmission and storage. The W3C (World Wide Web Consortium) has developed several XML related standards that fulfill the needs of very different application fields.

A. Storage and Manipulation

Currently, all information about an AIP system is stored in AFW files (a binary proprietary format). Once loaded into AIP, the information that these files contain is accessible via a programmatic interface and a user interface (the Plant Explorer). The use of third party products for modifying these files directly is highly limited, due to their propietary nature, although it is true that AFW files can always be modified by a program that uses AIP programming interfaces. However, if

this information was stored in XML format, the representation would be human and machine readable and it could be used and easily modified with virtually any programming language or XML tool, without the need of the AIP frameword to be running. Nowadays, a plethora of modern programming languages include libraries (typically based on the SAX or DOM standards) that allow the programmer to parse and process XML documents.

If we focus exclusively on storage, a great advantage derives from the fact that XML can be an excellent intermediate representation for relational data. We can map the XML representation of the system to our favorite relational database, providing a more robust mean of storage than a simple file. There are now several products in the relational and object-oriented database market that provide an XML binding, making this mapping process easier.

If a project has a large number of COM components, scalability can become an issue in AIP. Each time a new component is added to an AIP project, it is registered in Windows. If a project has to model a great number of similar small objects, this solution quickly overloads the registry unnecessarily. The alternative proposed is to exclusively use XML and a database for modelling purposes and treat the functional requirements separately.

B. Communication

Once in XML format, Aspect Objects could be sent through standard Internet communication channels and be addressed to any connected device. By means of XML transformations (XSLT), the XML stream could be converted into an XHTML file. The presentation of the information would depend on the display capabilities of the device. And this would not just be limited to Aspect Objects. Any subset of the system could be sent through the Internet by using this method. Thus, if we continue with the example of the robot plant, we can imagine an operator receiving an alarm in his palmtop from one of the robots in the factory. A more practical example could be, for instance, that a manager had instant up to date information about the production of his plant while he is on a business trip.

As a step further, XML web services could be used for real-time interaction with the objects in the factory. A remote call might be addressed to any object in the factory exposing its services. In this way, SOAP could be used as the communication layer for building HMI (Human to Machine Interface) applications. This is known in AIP terminology as the operator workplace. Security issues should be addressed at this point.

C. Graphics

XML can achieve integration even at the graphical level thanks to the SVG [12] standard. The platform currently uses graphical ActiveX controls to build the operator workplace which are highly integrated with the rest of the COM environment.

If in future development COM is left apart as the driving technology behind AIP, the possibility of using SVG graphics for designing the operator workplace should be seriously considered due to its graphical power, its interaction capabilities and the growing number of supporting products.

V. REALIZATION

In order to represent an AIP project using XML, we consider that at least three documents are necessary:

- A generic XML Schema for AIP.
- An XML Schema specific for the predefined and user defined types.
- An XML file containing all the objects in the system.

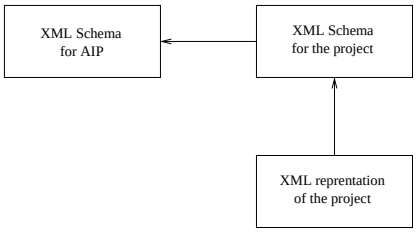

Fig. 5. Representation

We have selected the XML Schema format among other possible solutions for structuring our XML documents. This choice has been made for several reasons. Apart from the fact that XML Schema is a W3C recommendation, its type support and its ability for expressing relationships among elements have made it an essential tool for our project. Other schema languages were rejected because they were in an experimental state or not so widely accepted as XML Schema. The use of DTDs, which is another W3C recommendation, was discarded because of its lack of advanced built-in data types. Besides, XML Schemas are written in XML itself, while DTDs (although SGML compliant) are expressed in a different language (a kind of BNF grammar). That makes the use of XML Schemas for XML formatting purposes more consistent.

The first XML Schema is used to model all AIP concepts and their relationships: Structures, Aspects, Aspect Objects, etc... All of them are defined as complex types in the XML Schema for AIP.

The second XML Schema serves for declaring the predefined and user defined types. This schema is likely to be divided into, at least, two schemas: one that holds predefined types of the platform and one that is specific to the project. It would be possible as well to have several repositories of schemas, each one grouping a set of types specific to different kind of industrial projects.

Finally, the XML file will contain all the Aspect Objects that compose the system, along with their respective Aspects. This XML file will conform to the previous two schemas. Validation tools are available to check the conformity of XML files with XML Schemas, and most XML libraries can also validate XML documents. This will allow the detection of inconsistencies every time a modification takes place. For example, we have seen above (section III-A) that an Object Type can determine the number of children objects that an Aspect Object can hold. Let us imagine that we declare the minimum number

of children to be two and we instantiate the Object Type, resulting in an Aspect Object with two children. Afterwards, we change our Object Type definition, so the minimum number of children is now three. The Aspect Object previously created has been left in an inconsistent state regarding its Object Type definition, since it only has two children. With the XML approach, these types of inconsistencies will be automatically detected as soon as we validate the XML document against its XML Schema.

Let us see the example of the robot cells using a simplified XML model of the platform. The XML Schema that describes the concepts of AIP is shown in Table I. This schema can also be viewed in a graphical form in Fig. 6 and Fig. 7 thanks to the abilities of the XML editor we are using (XMLSpy 5.0 [13]). The first figure shows how a System is divided in Structures and Structures are composed of Aspect Objects. The second figure shows the definition of Object Type. An instance of any Object Type (an Aspect Object) will hold Aspects and children Aspect Objects. In this simplified model, Aspects are represented by a simple string of characters. In practice, different Aspect Types can have a very different nature.

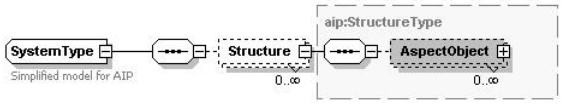

Fig. 6. Decomposition of a System

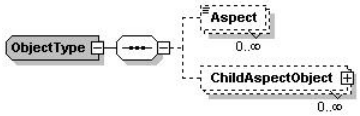

Fig. 7. Type definition of an Aspect Object

Next, we develop the XML Schema for the Robot Cells project. We have divided this schema into two. The first one (Table II) describes the actual System element and Structures that the project is going to handle, deriving them from the generic definitions of the previous XML Schema. The second one (Table III) defines the concrete Object Types that model the Aspect Objects in the project: Factory, Cell and Robot. This latter schema is joined to the first by using an *include* statement.

In order to clarify all these concrete definitions, we show them in two schematic diagrams. In the diagram showed in Fig. 8 we see that the Robot Cells project is composed of two Structures: a Functional Structure (empty at this stage) and a Location Structure where we can find several Factories. In Fig. 9 we can see that a Factory is composed of two Aspects (Name and Description) and several Cells. Cells can have up to four Robots as children.

Last, we elaborate an XML document (see Table IV) holding the instances of our previously declared types. In our example, we can see one Factory containing two Cells: one with a Painter robot and the other with an Assembler and a Screwdriver robots. Validation against XML Schemas assures the consistency of the instances with their type definition. The AIP system that would correspond to this document should be similar to the one shown in Fig. 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://lgl.epfl.ch/AIP"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:aip="http://lgl.epfl.ch/AIP"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

   <xs:complexType name="SystemType">
      <xs:annotation>
         <xs:documentation>
            Simplified model for AIP
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="Structure"
                     type="aip:StructureType"
                     minOccurs="0"
                     maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="StructureType">
      <xs:annotation>
         <xs:documentation>
            Specific view of the system
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="AspectObject"
                     type="aip:ObjectType"
                     minOccurs="0"
                     maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="ObjectType">
      <xs:sequence>
         <xs:element name="Aspect"
                     type="aip:AspectType"
                     minOccurs="0"
                     maxOccurs="unbounded"/>
       <xs:element name="ChildAspectObject"
                   type="aip:ObjectType"
                   minOccurs="0"
                   maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>

   <xs:simpleType name="AspectType">
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
</xs:schema>
```

TABLE I
AIP SIMPLIFIED CONCEPT MODEL

VI. CONCLUSION

In this paper we have seen how an XML infrastructure could help the AIP platform to achieve its objectives of integration at different levels. We have shown as well that XML documents can be used to represent the structures of an AIP system and their content. We found XML Schemas to be an excellent tool for modelling the concepts and the types of AIP. This model allows us to perform consistency checks in the XML documents that contain the instances of these types. We also envisage the use of other XML related technologies in AIP, like SOAP and SVG, so the platform could get advantage of their standard nature, Internet adaptation and uniformity of processing.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://lgl.epfl.ch/RobotCells"
           xmlns="http://lgl.epfl.ch/RobotCells"
           xmlns:aip="http://lgl.epfl.ch/AIP"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

   <xs:import namespace="http://lgl.epfl.ch/AIP"
              schemaLocation="AIP_Concepts.xsd"/>

   <xs:include schemaLocation="RobotObjectTypes.xsd"/>

   <xs:element name="RobotCellsSystem"
               type="RobotCellsSystemType"/>

   <xs:complexType name="RobotCellsSystemType">
      <xs:complexContent>
         <xs:restriction base="aip:SystemType">
            <xs:sequence>
               <xs:element name="LocationStructure"
                           type="LocationStructureType"/>
               <xs:element name="FunctionalStructure"
                           type="FunctionalStructureType"/>
            </xs:sequence>
       </xs:restriction>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="LocationStructureType">
      <xs:complexContent>
         <xs:restriction base="aip:StructureType">
            <xs:sequence>
               <xs:element name="Factory"
                           type="FactoryType"
                           minOccurs="0"
                           maxOccurs="unbounded"/>
            </xs:sequence>
       </xs:restriction>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="FunctionalStructureType">
      <xs:complexContent>
         <xs:restriction base="aip:StructureType">
            <xs:sequence>
            <xs:element name="AspectObject"
                        type="aip:ObjectType"
                        minOccurs="0"
                        maxOccurs="unbounded"/>
            </xs:sequence>
       </xs:restriction>
      </xs:complexContent>
   </xs:complexType>
```

TABLE II

CONCRETE INFRASTRUCTURE OF ROBOT CELLS PROJECT

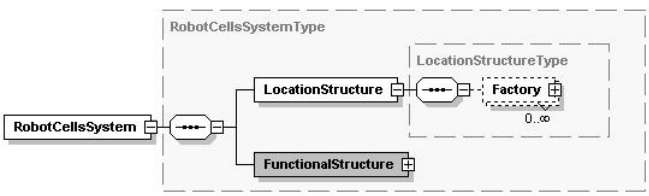

Fig. 8. Robot Cells System

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://lgl.epfl.ch/RobotCells"
           xmlns="http://lgl.epfl.ch/RobotCells"
           xmlns:aip="http://lgl.epfl.ch/AIP"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

   <xs:import namespace="http://lgl.epfl.ch/AIP"
              schemaLocation="AIP_Concepts.xsd"/>

   <xs:complexType name="FactoryType">
      <xs:complexContent>
         <xs:restriction base="aip:ObjectType">
            <xs:sequence>
               <xs:element name="Name"
                           type="aip:AspectType"/>
               <xs:element name="Description"
                           type="aip:AspectType"/>
               <xs:element name="Cell"
                           type="CellType"
                           minOccurs="0"
                           maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:restriction>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="CellType">
      <xs:complexContent>
         <xs:restriction base="aip:ObjectType">
            <xs:sequence>
               <xs:element name="Name"
                           type="aip:AspectType"/>
               <xs:element name="Robot"
                           type="RobotType"
                           maxOccurs="4"/>
            </xs:sequence>
       </xs:restriction>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="RobotType">
      <xs:complexContent>
         <xs:restriction base="aip:ObjectType">
            <xs:sequence>
               <xs:element name="Name"
                           type="aip:AspectType"/>
               <xs:element name="ChildAspectObject"
                           type="aip:ObjectType"
                           minOccurs="0"
                           maxOccurs="0"/>
            </xs:sequence>
       </xs:restriction>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```

TABLE III

OBJECT TYPE DEFINITIONS OF ROBOT CELLS PROJECT

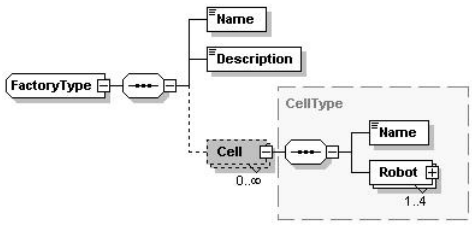

Fig. 9. Factory Type definition

```
<?xml version="1.0" encoding="UTF-8"?>
<RobotCellsSystem
   xmlns="http://lgl.epfl.ch/RobotCells"
   xmlns:aip="http://lgl.epfl.ch/AIP"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://lgl.epfl.ch/RobotCells
                       G:\AIPXML\Robot_Cells.xsd">

   <LocationStructure>
      <Factory>
         <Name>RobotCells Factory</Name>
         <Description>Example Factory</Description>

         <Cell>
            <Name>Cell 1</Name>
            <Robot>
               <Name>Painter</Name>
            </Robot>
         </Cell>

         <Cell>
            <Name>Cell 2</Name>

            <Robot>
               <Name>Assembler</Name>
            </Robot>

            <Robot>
               <Name>Screwdriver</Name>
            </Robot>
         </Cell>
      </Factory>
   </LocationStructure>

   <FunctionalStructure/>
</RobotCellsSystem>
```

TABLE IV

A ROBOT CELLS PROJECT

REFERENCES

[1] I. Crnkovic and M. Larsson, "A case study: Demands on component-based development," in *Proceedings of the 22nd International Conference on Software Engineering*. ACM Press, June 2000, pp. 23–31.

[2] ——, "Challenges of component-based development," *Journal of Software Systems*, Dec. 2001.

[3] O. Preiss and M. Naedele, "Architectural support for reuse: A case study in industrial automation," in *Building Reliable Component-Based Software Systems*, M. L. I. Crnkovic, Ed. Artech House Publishers, 2002, ch. 17.

[4] P. Clements and L. M. Northrop, Eds., *Software Product Lines: Practices and Patterns*. Addison-Wesley, Aug. 2001.

[5] P. Froehlich, Z. Hu, and M. Schoelzke, "Using uml for information modeling in industrial systems with multiple hierarchies," in *UML2002*, 2002, pp. 63–72.

[6] *IEC (6)1346-1 Industrial systems, installations and equipment, and industrial products - Structuring principle and reference designations*, International Electrical Commission (IEC) Std., 1996.

[7] J. Göpfert and M. Steinbrecher, "Modulare produktentwicklung leistet mehr," *Harvard Business Manager, Heft 3/2000*, pp. 20–30, 2000.

[8] Ole for process control. OPC foundation. [Online]. Available: http://www.opcfoundation.org/

[9] Extensible markup language. World Wide Web Consortium. [Online]. Available: http://www.w3.org/XML

[10] Microsoft .net home. Microsoft Corporation. Redmond, WA. [Online]. Available: http://www.microsoft.com/net

[11] P. Tapadiya, *.NET Programming. A practical guide using C#*. Prentice Hall, 2002, pp. 63–110.

[12] Scalable vector graphics. World Wide Web Consortium. [Online]. Available: http://www.w3.org/Graphics/SVG/Overview.htm8

[13] Xmlspy 5.0. Altova GmbH. Wien, Austria. [Online]. Available: http://www.xmlspy.com/products_ide.html