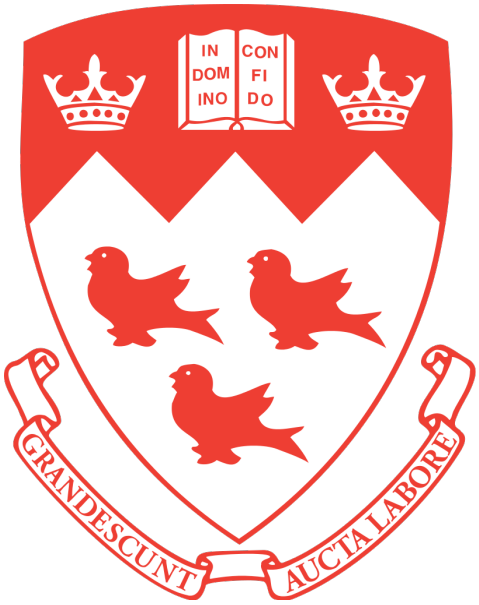

An XML-based Information Model for the Exchange of Laser Powder Bed Fusion (LPBF) Additive Manufacturing Process Plans

By
Sariel R. Coronado
Department of Mechanical Engineering, McGill University

Research Supervisor: Prof. Yaoyao (Fiona) Zhao

A Thesis
Submitted to McGill University in partial fulfilment
of the requirements of the Undergraduate Honours Program

April 12, 2022

Abstract

In the metal Additive Manufacturing (AM) market, Laser Powder Bed Fusion (LPBF) processes contain high complexity in the information technology domain. LPBF process plans currently suffer from deficiencies in the existing data modelling infrastructure, which has yet to overcome major integration challenges. As such, process planning information cannot be readily interchanged between systems due the existence of: (1) competing approaches to its organization, and (2) gaps in understanding of that information. This thesis seeks to address this urgent need by developing an information model, implemented in XML Schema, of LPBF AM process planning information from two LPBF manufacturers: EOS GmbH and Renishaw. As a preparatory step, an IDEF0 functional model is constructed and presented. A comparative analysis of the competing process plans is undertaken, with the result being that several key differences and commonalities are identified and discussed. In the future, these key observations can serve to facilitate the development of process planning information exchange standards.

Dans le marché de la fabrication additive (FA) métallique, les processus de la fusion sur lit de poudre au laser (FLPL) contiennent un grand niveau de complexité dans le domaine des technologies de l'information. Les plans de processus FLPL souffrent actuellement de lacunes dans l'infrastructure de modélisation de données existante, qui doit encore surmonter certains défis d'intégration majeurs. En tant que telles, les informations de planification de processus ne peuvent pas être facilement échangées entre les systèmes en raison de l'existence des : (1) approches concurrentes de son organisation et (2) carences dans la compréhension de ces informations. Cette thèse vise à répondre à ce besoin urgent en développant un modèle d'information, implémenté en XML Schema, des informations de planification de processus FA FLPL de deux fabricants de machines FLPL: EOS GmbH et Renishaw. Comme étape préparatoire, un modèle fonctionnel est construit en IDEF0 et présenté. Une analyse comparative des plans de processus concurrents est entreprise, avec comme résultat que plusieurs différences et points communs clés sont identifiés et discutés. À l'avenir, ces observations clés peuvent servir à faciliter le développement de standards d'échange d'informations sur la planification des processus.

Acknowledgements

The thesis author wishes to extend heartfelt gratitude towards Professor Fiona Zhao for her constancy in the guidance and support she provided. Likewise, thanks are due to Dr. Valentina König, Research Program Manager at ModuleWorks GmbH, for having provided the original motivation for this thesis, as well as for her insightful feedback. The kind people of the McGill ADML (Additive Design and Manufacturing Lab) and $P^2[AM]^2L$ (Powder Processing and Additive Manufacturing of Advanced Materials Lab) are also to be thanked for their assistance and knowledge. Finally, the author extends indebted gratitude towards the researchers at NIST, without whose work the present thesis would not have been feasible. Particularly, researchers Dr. Yan Lu and Dr. Paul Witherell are thanked for their help in private communication.

Contents

1 Introduction

1.1 Metal Additive Manufacturing

The global additive manufacturing (AM) market was estimated at 15.4 Billion US dollars for the year 2020, and is expected to grow to 61.1 Billion US dollars by 2026 [1]. The growing impact of new AM technologies on engineering cannot be understated, far from being restricted purely to prototypes, today's AM machines are increasingly used for functional end-use products for industrial or consumer use. Of particular interest are AM technologies that produce metal parts; high-end metal AM components can equal or even surpass the performance of traditionally machined parts, allowing designers to realize features which could not be otherwise produced. Modern CAD/CAM software packages can perform tasks such as topology optimization, automated build orientation and optimization, support structure optimization, distortion compensation algorithms based on thermal simulation loops, scan-path optimization, and many more. With these tools, engineers are capable of generating ever more efficient designs, benefiting from reduced or even eliminated tooling costs. High-value, low-volume parts with traceability requirements, commonly found in the dental, medical, aerospace and defense sectors, are the prime examples of candidates for production using metal AM technologies. An excellent example of the potential of metal AM can be seen in Figure 1. It shows a prototype heat exchanger, designed by Advanced Engineering Solutions and manufactured on an EOS M290 Laser Powder Bed Fusion (LPBF) machine, using EOS's Aluminium AlSi10Mg metal powder. Its interior structure is a complex labyrinth of gyroid lattices that divide the interior volume into two spaces, allowing the fluids to exchange heat more efficiently [2].

Figure 1: Helicopter heat exchanger by Advanced Engineering Solutions [2]

1.2 Laser Powder Bed Fusion

There exist multiple implementations of metal additive manufacturing, relying on a wide array of different technologies to achieve the same objective. Examples of these technologies include Binder Jetting, Direct Energy Deposition, Extrusion, and most commonly, Powder Bed Fusion. Indeed, the Powder Bed Fusion market represents more than half of the entire metal AM market, as illustrated in Figure 2.

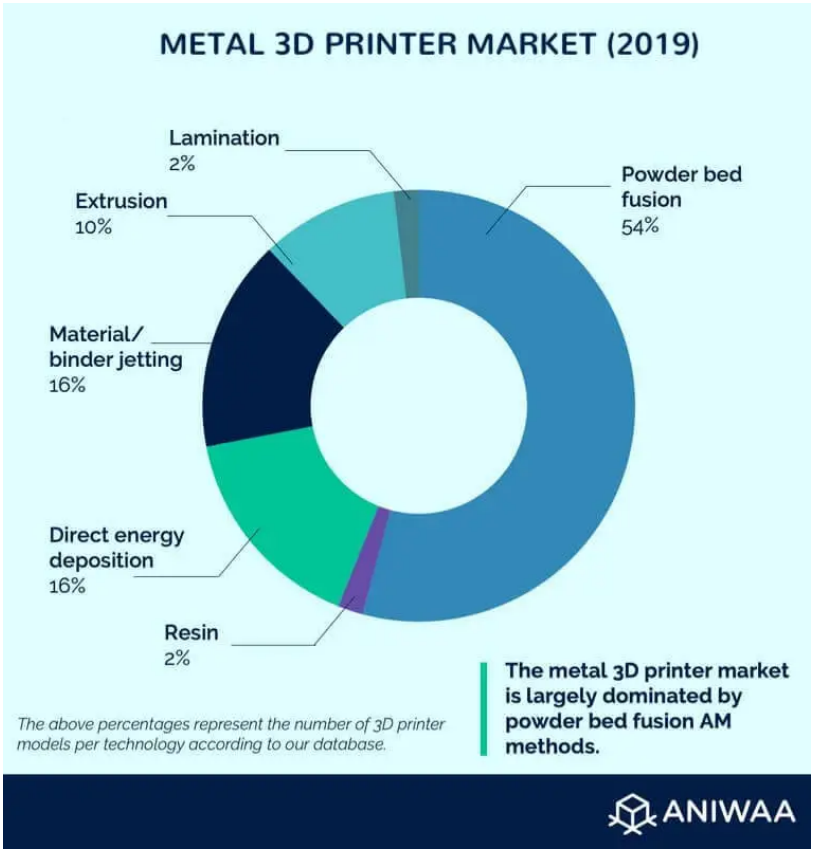

Figure 2: The metal AM market by technology market share [3]

A description of the physical process that underlies Powder Bed Fusion AM is as follows. A focused beam of energy is directed onto a bed of powdered metal by a computer-controlled system to achieve the selective melting or sintering of said metal powder. The solidified metal rests on a platform known as the build plate, and upon which more powder is distributed at regular intervals by a mechanical arm known as a recoater. By focusing the beam of energy on a pre-calculated pattern known as a scan path, a solid cross section is formed. The entire build plate assembly is then lowered by a preset distance (in the order of a few thousandths of an inch) to allow the recoater to re-cover the surface of the solidified layer with more powder. The next cross section can now be produced on top of it, thus repeating the cycle. The

selective melting/sintering of successive layers produces the finished part. The mechanism that imparts energy to the powder can be either a laser or an electron beam, when a laser is used, the process is known as Laser Powder Bed Fusion, or LPBF. An illustrative diagram of the process's components can be found in Figure 3. Since lasers are the most common energy source in Powder Bed Fusion, they are the subject of the present thesis.

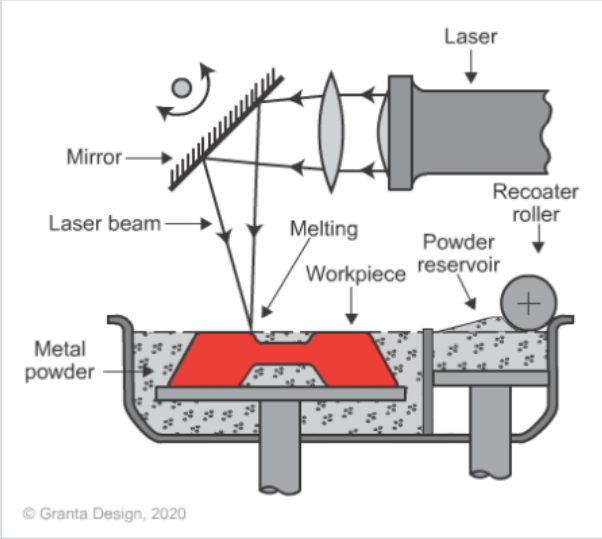

Figure 3: Principal components of an LPBF machine [4]

Some technical terms in the field of LPBF additive manufacturing, worth defining for the understanding of the present work, include "downskin", "inskin", and "upskin". These terms refer to regions of an LPBF part that are positioned such that they either have: no material under them, material under and over them, or no material over them, respectively. Figure 4 is an illustration from a manual [5] that illustrates these concepts.

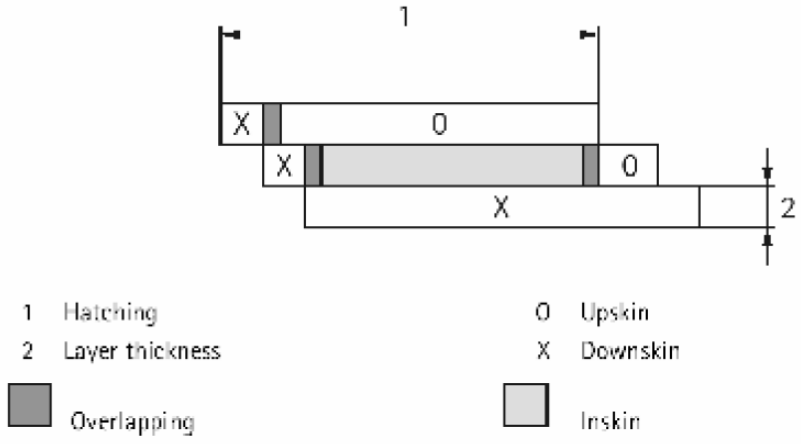

Figure 4: Skin definitions for an LPBF part [5]

1.3 Process Plans

An LPBF process plan is, abstractly, the collection of ordered information and knowledge as embodied by sequences of operations, system configurations, data structures, and hierarchies of parameters required for the complete manufacture of a given part by LPBF means.

More concretely, a process plan is expressed in the LPBF domain by so-called build files, which represent the output of Build Preparation Software. When a part must be manufactured, its geometry and features are encoded in some suitable fashion (.STL files are a traditional choice) and given as input to the Build Preparation Software, which will perform the process planning activity with input from the user, eventually yielding the aforementioned build file. The build file is then transferred, automatically or manually, to the LPBF machine controller, which will interpret the structures of information contained therein to generate all required manufacturing commands. Figure 5 is a graphical representation of the flow of information.

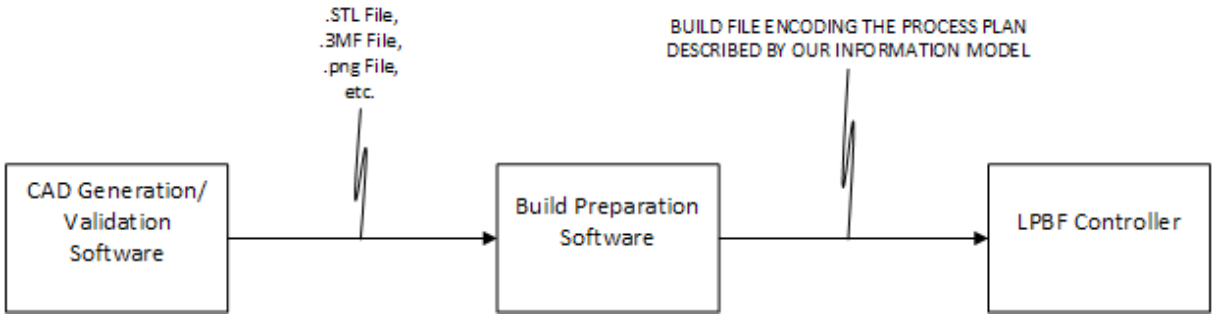

Figure 5: A high-level overview of information flow in the LPBF domain

1.4 Information Modelling

Modern manufacturing is replete with an abundance of data generated by nearly every computerized system involved in it, from the CAD software used to create feature-rich parts, to the Coordinate-Measuring Machine that produces detailed metrology reports, to the Product Lifecycle Management software that monitors the entire process. Additive manufacturing is not an exception, and with so much information being created, read, and transferred between diverse systems, very real issues concerning the interoperability of software components can arise. These issues can only be understood, and indeed resolved, by rigorous exploration of the underlying structures of information and the rules that govern their flow within a system.

The tool used by researchers and professionals to systematically understand the way that information is organized and flows within a system or domain is called Information Modelling, which is a technique to produce a comprehensive synthesis of a domain's data in the form of an Information Model. An information model can therefore be understood as "a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse." [6]. One of the goals of producing an information model is to provide a stable platform for the organized sharing of data between potentially disparate users.

Researchers create information models using an information modelling language. Several information modelling paradigms exist, utilizing distinct methodologies and forms of representation. Different information modelling languages follow different paradigms; some are inherently graphically represented, while others are textual. Examples of common languages include UML, IDEF1X, EXPRESS, and XML Schema. Chapter 2 of Reference [6] is an excellent overview of the characteristics of these modeling languages. A simple example of information modelling of the attributes of a helical thread feature in a part is seen in Figure 6. The modelling was done using EXPRESS-G, which is a common graphical information modelling language.

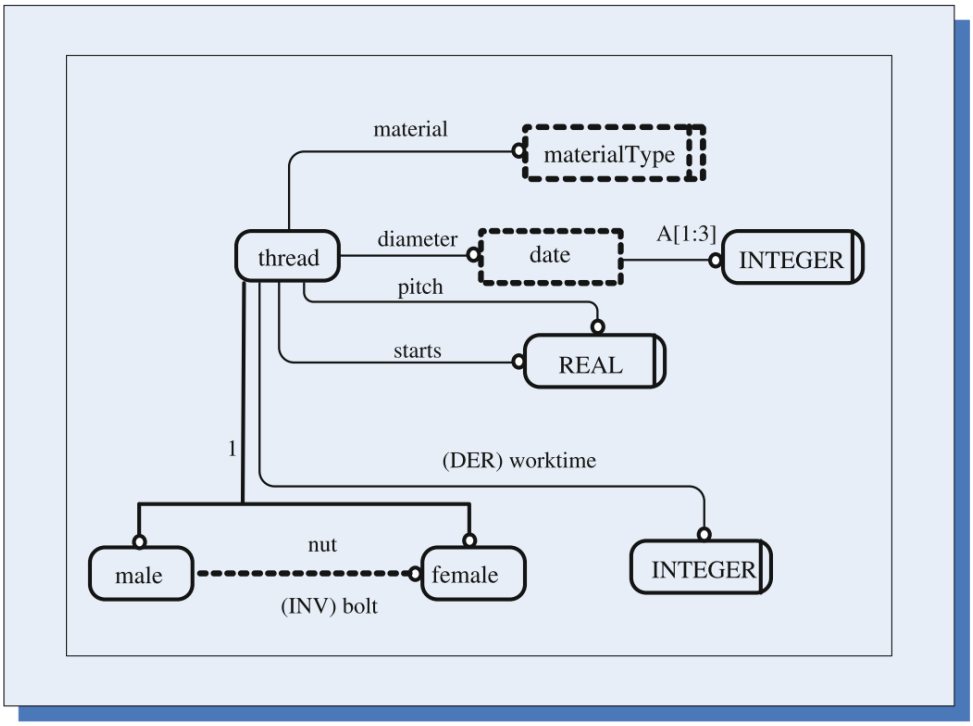

Figure 6: Model of a thread feature [6]

1.5 Functional Modelling

Just as Information Modelling seeks to decompose the information structures of a system into its constituent elements and governing relations, there exists an adjacent concept called Functional Modelling, whose aim is to provide a comprehensive specification of system functionality by hierarchical decomposition of the domain activities into sub-activities, all the while addressing the interconnections between these activities.

Functional modelling is, as the name implies, function-centric. The basic element is therefore the function, or activity. Activities can be abstract, such as process planning, or wholly concrete, such as baking a cake. The inputs and outputs of activities can consequently straddle the material and immaterial domains, as dictated by the activity at hand. This contrasts with information modelling, which lies well within the domain of information technology.

Often, in preparation for information modelling of a domain, functional modelling is performed of relevant activities and sub-activities. The benefit of this practice to the researcher is twofold. Firstly, a ”big-picture” perspective is acquired that can be critical in delineating the exact boundaries of the system to be investigated, which is not always evident at an initial glance, particularly when the degree of interdependency between constituent elements is elevated. And secondly, it can serve to elucidate many of the categories of entities to be included in the information model. The majority of inputs and outputs of the modelled activity can be mapped relatively straightforwardly to an associated entity within the information model to be constructed later. Figure 7 demonstrates a simple activity being modelled using IDEF0, a common functional modelling language. More detail on the semantics of IDEF0 will be provided in a later section.

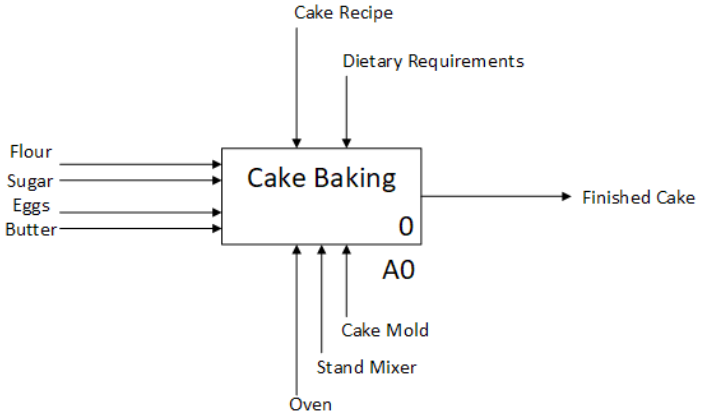

Figure 7: Functional Modelling of Cake Baking

1.6 Motivation

The original motivation for this thesis was originated from discussions with Dr. Valentina König, Research Project Manager at ModuleWorks GmbH, a German software company. ModuleWorks is a leading provider of CAD/CAM components, providing 3 - 5 axis machining and simulation technology which is used by many of the leading CAM systems and by specialized manufacturing companies. There is currently an interest at ModuleWorks to produce software solutions for LPBF process planning, therefore the present research project is intended to yield a comprehensive information model for immediate usage at the company.

However, a broader motivation also exists. As mentioned previously, the over-abundance of data flowing between software components presents severe challenges to interoperability and process reliability. Software components written decades ago continue to be used today, and will need to communicate correctly and consistently with future programs. Absent or conflicting standards, ambiguity, deliberate obfuscation — these are among the enemies of an integrated software environment that is easy to maintain, expand, and performs as originally designed. Only open information modelling can prevent these pitfalls and make sense of otherwise inscrutable arrays of data.

One can consider some of the previous successes of information modelling that lead, eventually, to industry standards in widespread use. ISO 6983 [7] is the international standard that defines the data format and semantics for the numerical control of machines, popularly known as G-code. The CNC machining process plan can be readily interchanged between ecosystems, with only relatively minor modifications that, if present, are well-known in advance.

Another example is the STL file format, ubiquitous in the consumer additive manufacturing market. It has been observed that in recent years, plastic 3D printers have proliferated prodigiously among the general public. How much more difficult would such a wave of additive manufacturing democratization have been without access to file formats that can be readily produced, transferred and interpreted by a wide array of CAD and slicing softwares?

The current state of affairs in the LPBF industry is one of closed-source, patent-protected development. While fused deposition modelling (FDM) AM technology enjoys multiple instances of open-source printers and software packages being readily available, LPBF cannot benefit from the same level of transparency. LPBF AM build files can generally only be opened and understood by proprietary software from their respective machine builders. The end result is that LPBF process planning information, in the current state of affairs, cannot be readily exchanged.

A mildly contrived example can hopefully illustrate the challenges faced by engineers because of this situation. One may consider a company that produces an innovative product using an LPBF machine from one specific brand, "X". Through much effort, they fine-tune their AM process to achieve stringent surface finish or mechanical strength requirements in that specific machine, for that specific part geometry, using that specific material powder formulation. The myriad settings the test engineers have painstakingly optimized are genuinely precious; the company's bottom-line depends on their correctness. Years later, the company is bought by a larger company and now the designs must be produced on LPBF machines of a different brand, "Y". Can the engineers simply transfer the parameters from the "X" ecosystem to the "Y" ecosystem? Currently, the answer is no, not without extensive testing, grief, and effort.

Therefore, there exists an urgent need for a uniform, standard-based information model for people and organizations that want to create, modify, transfer, and verify LPBF process plans for additive manufacturing. It is the aim of the present thesis to contribute to the general effort to address this need, and ultimately, bring about the democratization of metal 3D printing.

1.7 Research Objectives

1. Systematic decomposition, by functional modelling, of process planning and related activities to understand information flow between LPBF-related activities.
2. Development of a comprehensive information model of LPBF process plans via analysis of available LPBF machine manufacturer documentation, implemented in XML Schema.
3. Comparative analysis of different LPBF machine manufacturer process planning activities.

1.8 Thesis Outline

The remainder of the present thesis is organized in the following manner. Section 2 contains a literature review that demonstrates the state of current research. Section 3 is a presentation of the developed functional model, while Section 4 is a presentation of the developed information model, including a comparative discussion. Section 5 contains the conclusions of the thesis.

2 Literature Review

Available research literature on Functional and Information Modelling applied to LPBF process planning can be considered sparse. The present review will discuss two papers that dealt directly with the subject. An additional source of information that proved extremely valuable for this thesis was communications with researchers at NIST in the United States. In these communications, mention was made of a current effort to construct a database for additive manufacturing materials, implemented using XML Schema, called the Additive Manufacturing Materials Database (AMMD). This effort, including the XML Schema file itself, is publicly available [8] and will be discussed in this section insofar as it overlaps with the present work.

2.1 Functional Modelling of LPBF Processes

In 2017, Feng et al. [9] focused on developing a comprehensive activity decomposition of the entire LPBF process, from early design specification to production of quality assurance data on finished parts. The functional modelling language selected by the researchers was the IDEF0 methodology developed by the United States Air Force in the 1970s [10]. This is the same methodology utilized by the current thesis for the purposes of preparatory activity modelling in a capacity ancillary to the main data modelling effort.

The primary results of the article can therefore be understood by inspection of the produced activity model as embodied by the collection of IDEF0 diagrams. The diagrams most relevant to process plan modelling (as defined in the introduction) will now be presented.

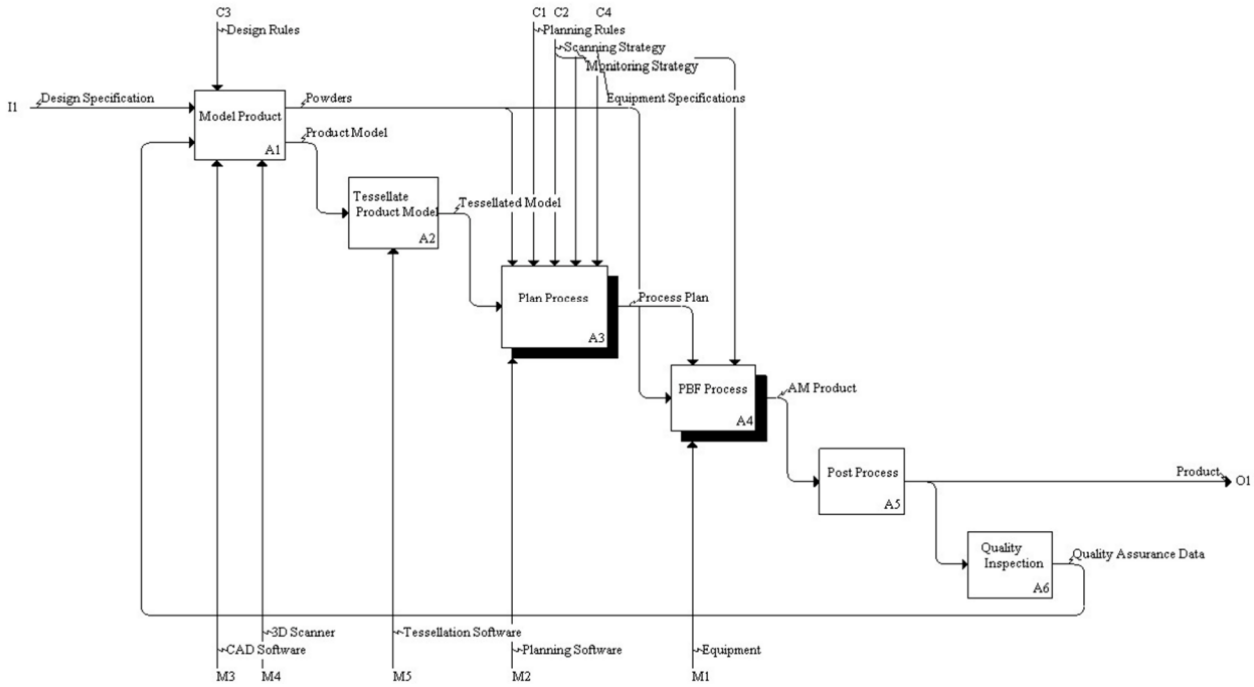

Figure 8: A0 IDEF0 diagram from (Feng et al., 2017) [9]

In the IDEF0 methodology, the overall activity being decomposed is known as A0. The A0 diagram found in Figure 8, therefore, represents the top level of decomposition. In it, the LPBF process has been divided into six sub-activites, numbered from A1 to A6. Two of the activities, A3 and A4, have a black shadow effect to denote that there exist lower-level IDEF0 diagrams that continue the activity decomposition process. Each activity rectangle can have arrows connecting to it from its four sides. The left and right sides are reserved for inputs and outputs, respectively. Inputs are consumed or transformed by the activity to create the output elements. The top side of an activity rectangle serves to indicate a special kind of input that is called ”control”, and represents any information that regulates or otherwise modifies the execution of the activity. Finally, the bottom side of the rectangle connects to so-called ”mechanisms”, which are a special kind of input that represent ”software, hardware and/or data that support the execution of the activity” [9]. With these basic principles, IDEF0 diagrams can be constructed to elucidate tangible relationships between functions and sub-functions in a system’s domain.

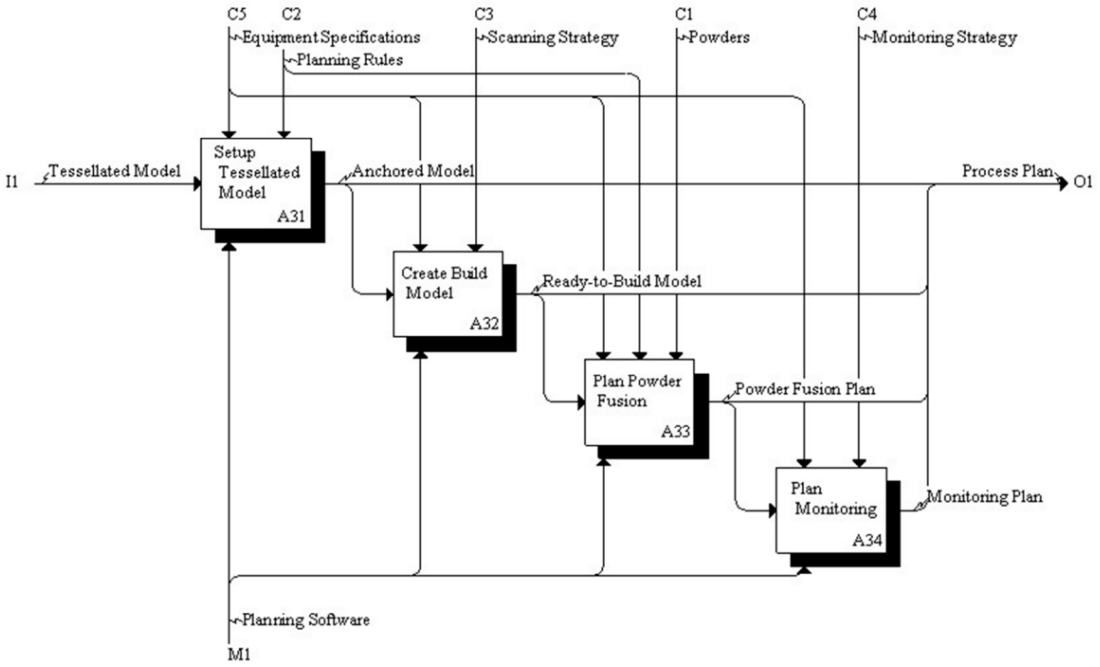

Figure 9: A3 IDEF0 diagram from (Feng et al., 2017) [9]

The A3 diagram is the most important since it responsible for the ”Process Plan” output element. It has an input of a ”Tessellated Model” and makes use of the ”Planning Software” mechanism to achieve its functional goal. In the introduction, Build Preparation Software was discussed, which would take on the role of planning software in this article’s model. Several items of interest should be noted in this diagram. Firstly, the process plan element is defined as the amalgamation of an ”anchored model”, a ”ready-to-build model”, a ”powder fusion plan” and a ”monitoring plan”. The anchored model can be understood as a geometrical definition of the position and orientation of the part to be manufactured. Part orientation is a critical activity in LPBF; the difficulty of printing any given model changes depending on how it is rotated and placed on the build plate. The ”ready-to-build model” represents a sliced version of the geometry that includes the laser scan-paths necessary for its manufacture, but not any material-specific parameters or system configurations. The ”powder fusion plan” output captures the remaining data structures responsible for containing the parameters and settings required to create a fully-defined build file. These settings include items such as laser power settings, oxygen level targets, recoater blade speeds, etc. The monitoring plan establishes feedback mechanisms such as sensors and their corresponding setups. In the present work, such monitoring information did not seem to be included in the build files of the examined brands of LPBF AM machines, and was therefore not included in the information model.

2.2 Conceptual Information Modelling of LPBF

In 2015, Lu et al. [11] presented an initial effort towards a comprehensive data model for AM processes. The authors, affiliated with NIST, described the state of current AM standards as ”unable to provide both the breadth and the depth needed for an integrated AM information model” [11], and therefore proposed a conceptual design for an integrated AM data model which they developed using a modeling methodology known as PPR – Product, Process, Resource.

The information modelling was preceded in this paper by functional modelling of the LPBF process, though only at the top level (A0 level) as shown in Figure 10. Since the IDEF0 methodology has already been elaborated on in the preceding subsection, further discussion will concern the data modelling aspect of this paper.

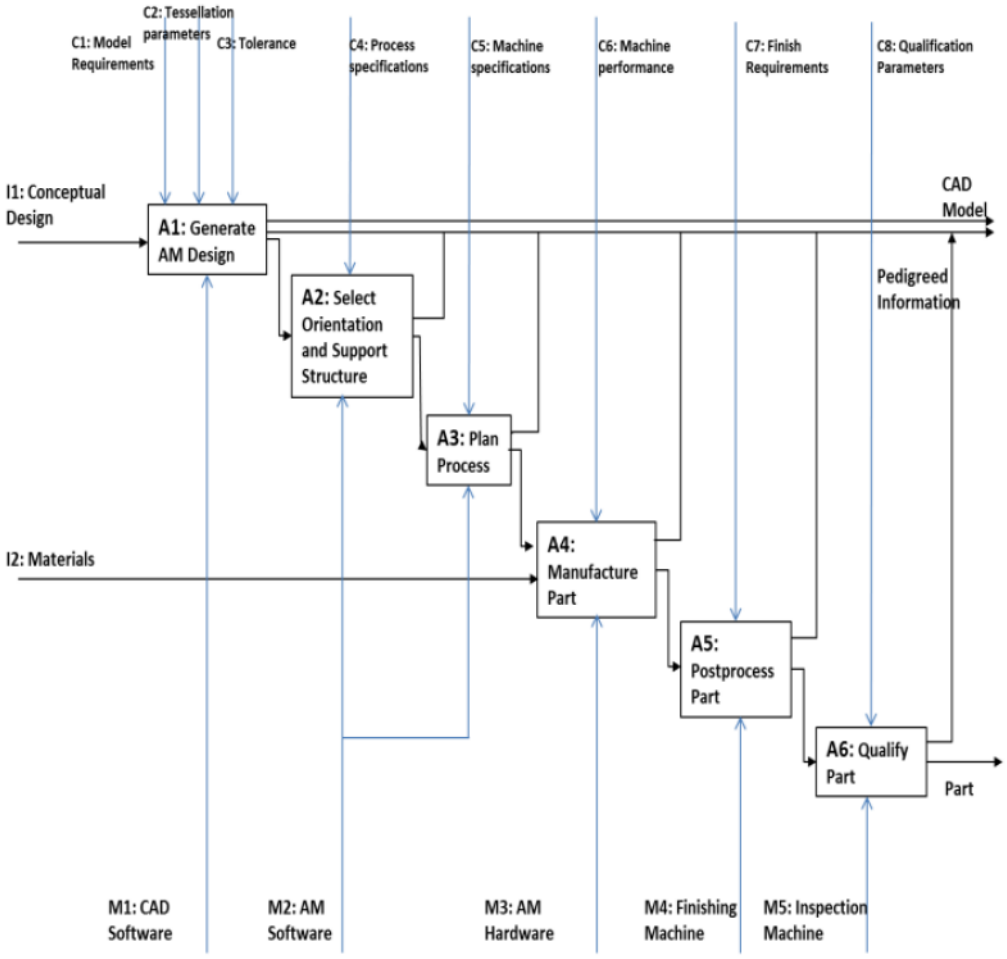

Figure 10: A0 IDEF0 diagram from (Lu et al., 2017) [9]

The scope of the conceptual information model developed in the paper is broad, with intended users in-

cluding design engineers, material scientists, AM equipment manufacturers, and AM machine operators. Product Lifecycle Management (PLM) schemas were used as a starting point for its synthesis. Figure 11 is a view of the data model constructed in UML, another common information modelling language. It is a high-level perspective applicable to not only LPBF, but any AM technology. It is concerned with the modelling of both process and resource domain information, which includes labor and material information. Elements of AM lifecycle and monitoring data are also represented in the model.

As with the previous article, some of this information being modelled lies outside the intended scope of the present thesis, which aims to provide a more low-level and granular perspective on the structures and patterns of LPBF build files than can be obtained from a conceptual model.

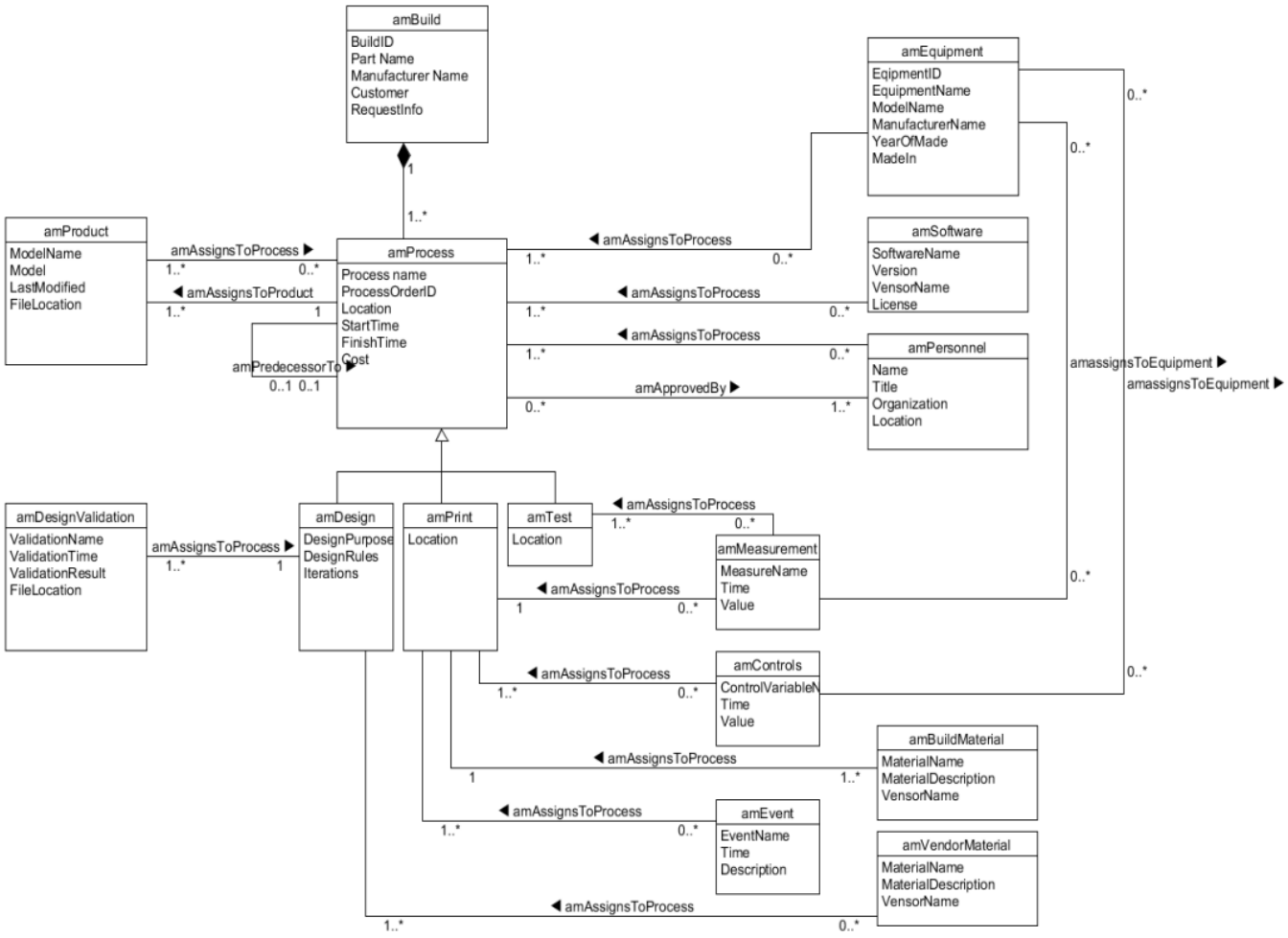

Figure 11: Conceptual Model from (Lu et al., 2017) [9]

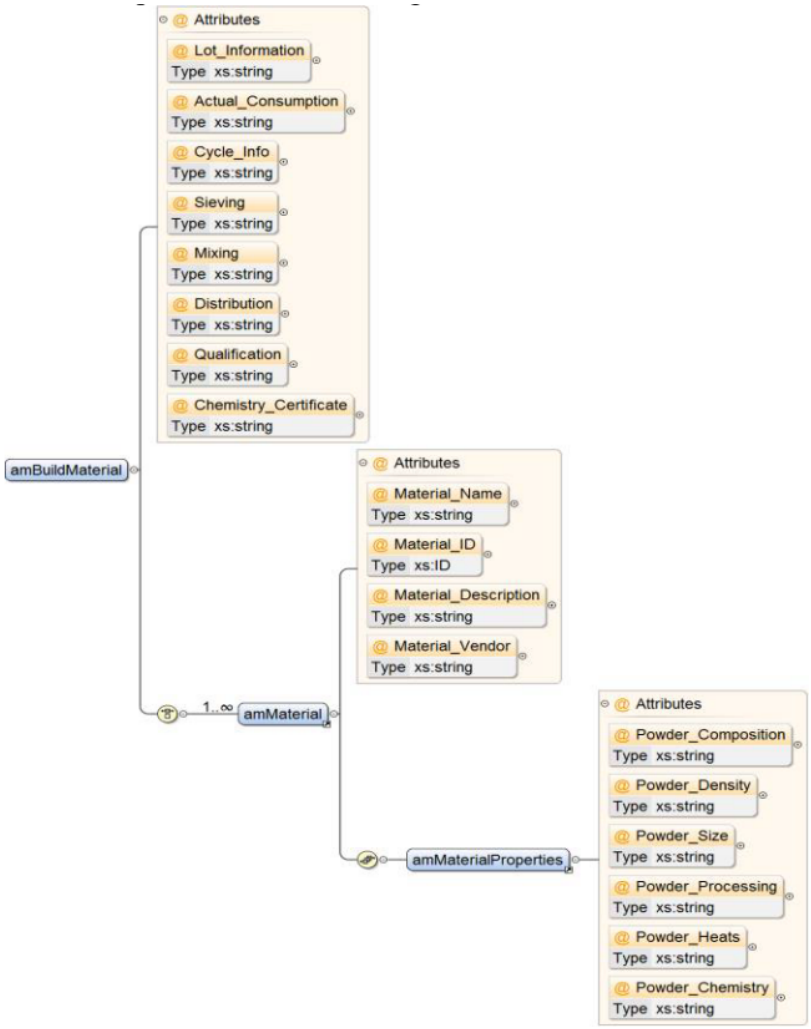

Figure 12: Build material XML Schema from (Lu et al., 2017) [9]

Notably, the authors performed a case study of the conceptual model by applying it to the characterization of material properties in the manufacture of a test part. An XML Schema, shown in Figure 12, is the result of this application. Though simple, it demonstrates a more detailed view of specific information structures, which is one of the goals of the present thesis. No further XML schema examples were presented in the paper.

2.3 Reported XML Schema of LPBF

As briefly mentioned at the beginning of this section, a most valuable resource for this project was NIST's AMMD XML Schema, a large file of some several thousand lines of code. It is a continuation of the earlier efforts mentioned in section 2.2. Before continuing, a brief overview of XML Schema and its use in information modelling will now be given in order to clarify some key concepts.

The eXtensible Markup Language (XML) is a very common text-based language used for the storing and transmission of information in a way that is understandable by both humans and computers. It is ubiquitous on the Internet due to its flexibility in representing arbitrary sets of data in a standardized way. XML Schema (XSD) [12] is a meta-language extension of XML used to create files called "schemas" that represent a "formalization of the constraints, expressed as rules [...] that apply to a class of XML documents" [6]. These formalized rules are machine-enforceable, which facilitates the validation of files. Given how information modelling is essentially about constraints on information, XML Schema is an excellent tool to concretize and implement information models. An added bonus is that XSD files can be easily edited by both open-source and commercial software.

As mentioned in the introduction, there are alternative tools for information modelling, some of which are inherently graphical, such as UML or EXPRESS-G. While the XML Schema language is text-based, which is arguably a disadvantage for the purposes of presentation, it nevertheless lends itself readily to graphical representations of its modelled structures thanks to its hierarchical composition. XML files are essentially sets of elements, where each element can either represent a fundamental datatype such as a boolean or an integer, or it can represent a collection of other child elements. These parent elements are referred to as complex types, and can take on different forms, such as sequences of elements, or a choice between elements. Elements can be optional, mandatory, or have user-defined multiplicity. Consequently, XML Schema files can be seen as containing one or more directed trees from graph theory, and therefore illustrations of them will be tree-like in appearance.

Returning to the subject of the AAMD schema, it contains an element called ”AMProcessPlan” which represents the overlap between the NIST model and the present thesis’s scope. This element contains a ”choice” (in the XML sense) for the process planning information between three LPBF machine brands. However, the schema is a work-in-progress, and at the time of writing, only the process plan for LPBF machines made by EOS is fully modelled.

EOS GmbH is a German manufacturer of metal AM PBF machines and one of the largest by market share, as seen in Figure 13. At McGill University, the author of this thesis had access to a Renishaw LPBF machine and its build preparation software documentation. Through correspondance with NIST, equivalent documentation was obtained for the EOS software. These two software manuals were principal sources for the process plan information modelling performed, and together, they cover nearly a third of the machines in the metal AM market.

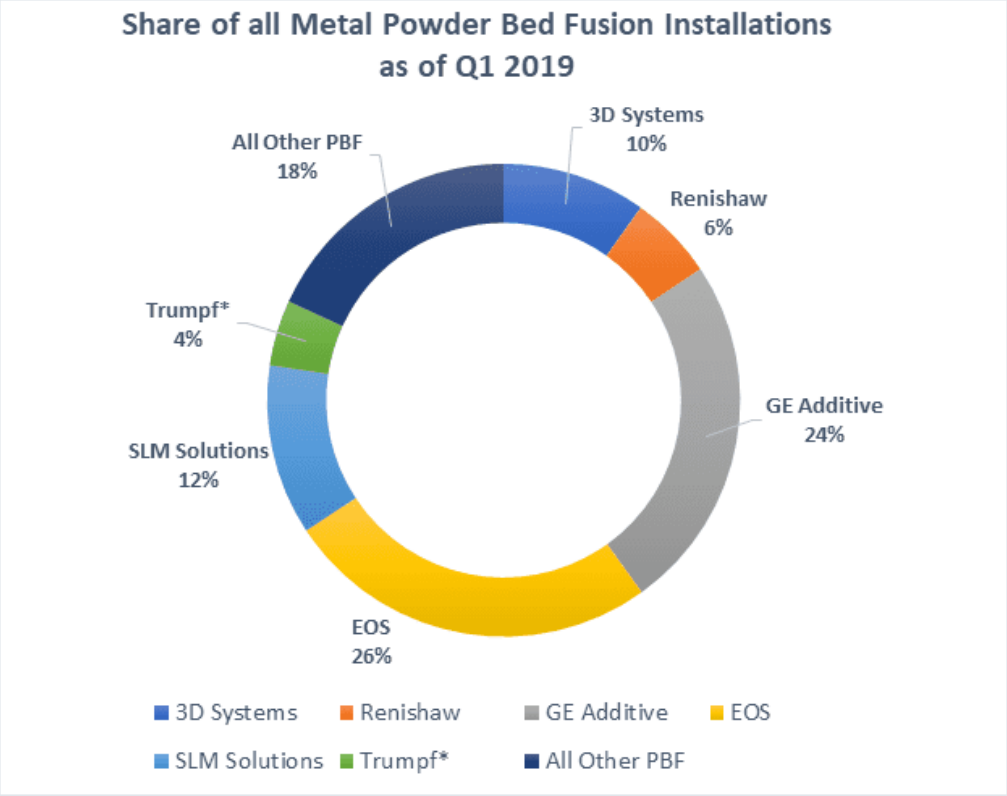

Figure 13: Q1 2019 metal PBF market share [13]

A selected subset of the child elements of the AMMD ”AMProcessPlan” element will now be presented to illustrate the approach taken to model EOS process planning information. All XML Schema figures were generated using the commercial software XMLSpy.

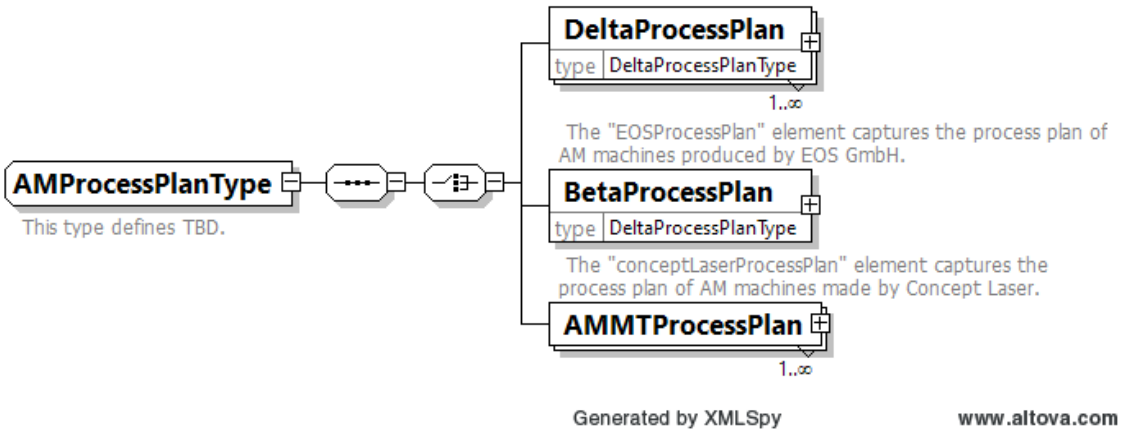

Figure 14: AMMD's AMProcessPlan element [8]

Note that the addition of the "-Type" suffix to the name of an element simply indicates that, strictly speaking, what is being shown is the abstract type of an element instead of a specific instance of that type. When an XML type only has a single instance, having an explicit type definition is not absolutely necessary, and the practical difference between the type and its element becomes minimal for the purposes of information modelling. When a set of elements are all instances of a given type, then having an explicit type definition is very useful.

The AMProcessPlan element in the AMMD schema is a Sequence composed of a Choice of three possible process plan implementations. DeltaProcessPlan contains the EOS process plan modelling. BetaProcessPlan is intended to model Concept Laser AM machines by General Electric, but currently that is not modelled, since the element is an instance of type DeltaProcessPlanType, which refers to the EOS process plan. AMMTProcessPlan refers to the AM Metrology Testbed, which is an open-platform LPBF system in developed by NIST for research purposes [14]. However, the element is not as complete as the one for EOS machines.

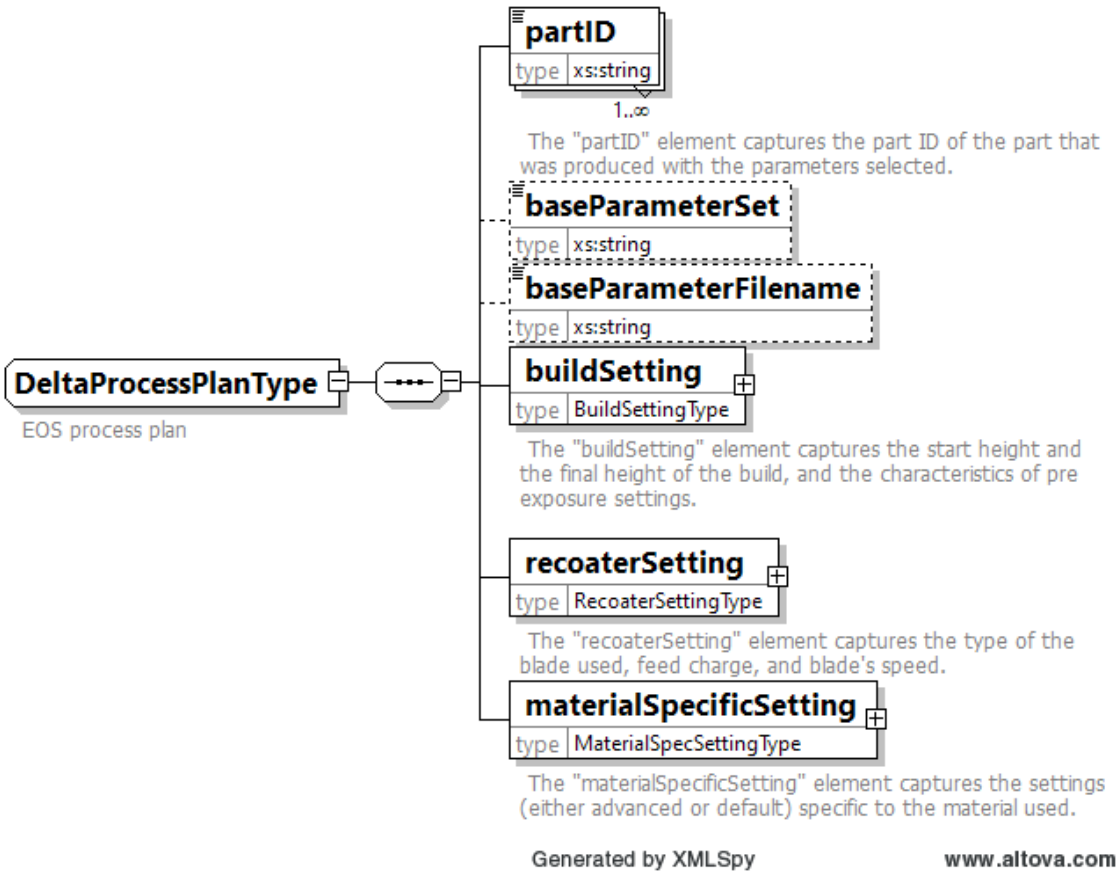

Figure 15: AMMD's DeltaProcessPlan element [8]

Inspecting the DeltaProcessPlan element shows a sequence of setting elements organized by the area they modify. Of particular interest are the buildSetting, recoaterSetting and materialSpecificSetting elements.

Inside buildSetting, floating point datatype elements capture the start height and final height of the build process, as well as the thickness of each layer in millimeters. Layer thickness has an important impact on the microscopic grain structure and the mechanical properties of the finished part [15]. Additionally, a family of settings related to an EOS feature called DMLS (an acronym for Direkt Metall Laser Schmelzen) are included.

Inside recoaterSetting, a string datatype element encodes the type of blade used, and an annotated float element captures the amount of metal powder that is introduced by establishing the upwards movement of the feed-bed as a percentage of the movement of the build-bed. Two float elements capture the speed of the recoater blade when spreading the metal powder and when retracting, and an enumeration element with values {ON, OFF} is used to encode whether the feed-bed performs additional movements to avoid contact with the recoater blade, in order to prevent potential damage.

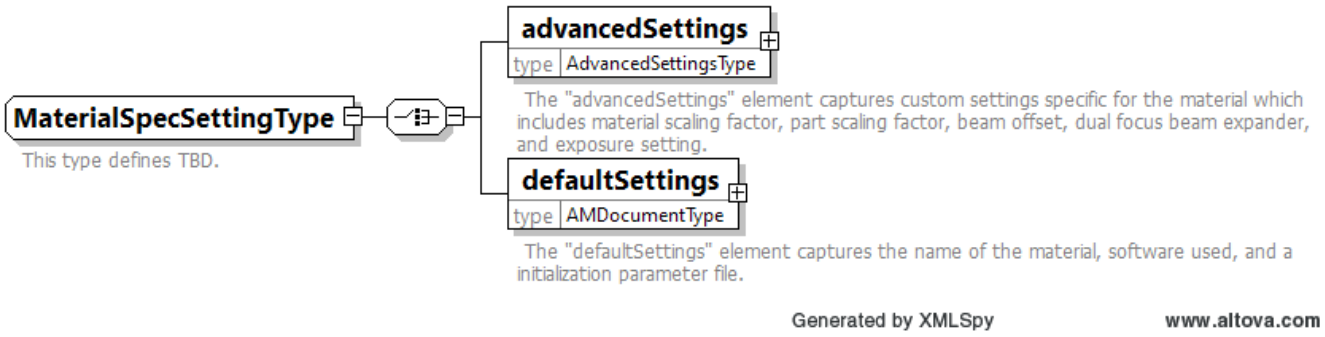

Figure 16: AMMD's MaterialSpecSetting element [8]

The MaterialSpecSetting element is a Choice of an advancedSettings type element or a defaultSettings element of type AMDocumentType. The latter is interesting since the AMDocumentType is an XML model of an electronic document, and in the context of the material specific settings, it represents a file containing default parameters with which to initialize the build file. This would allow users to specify their desired process planning configuration externally and therefore opens the door to potential integration of a separate system to handle this decision. Figure 17 expands the AMDocument definition.

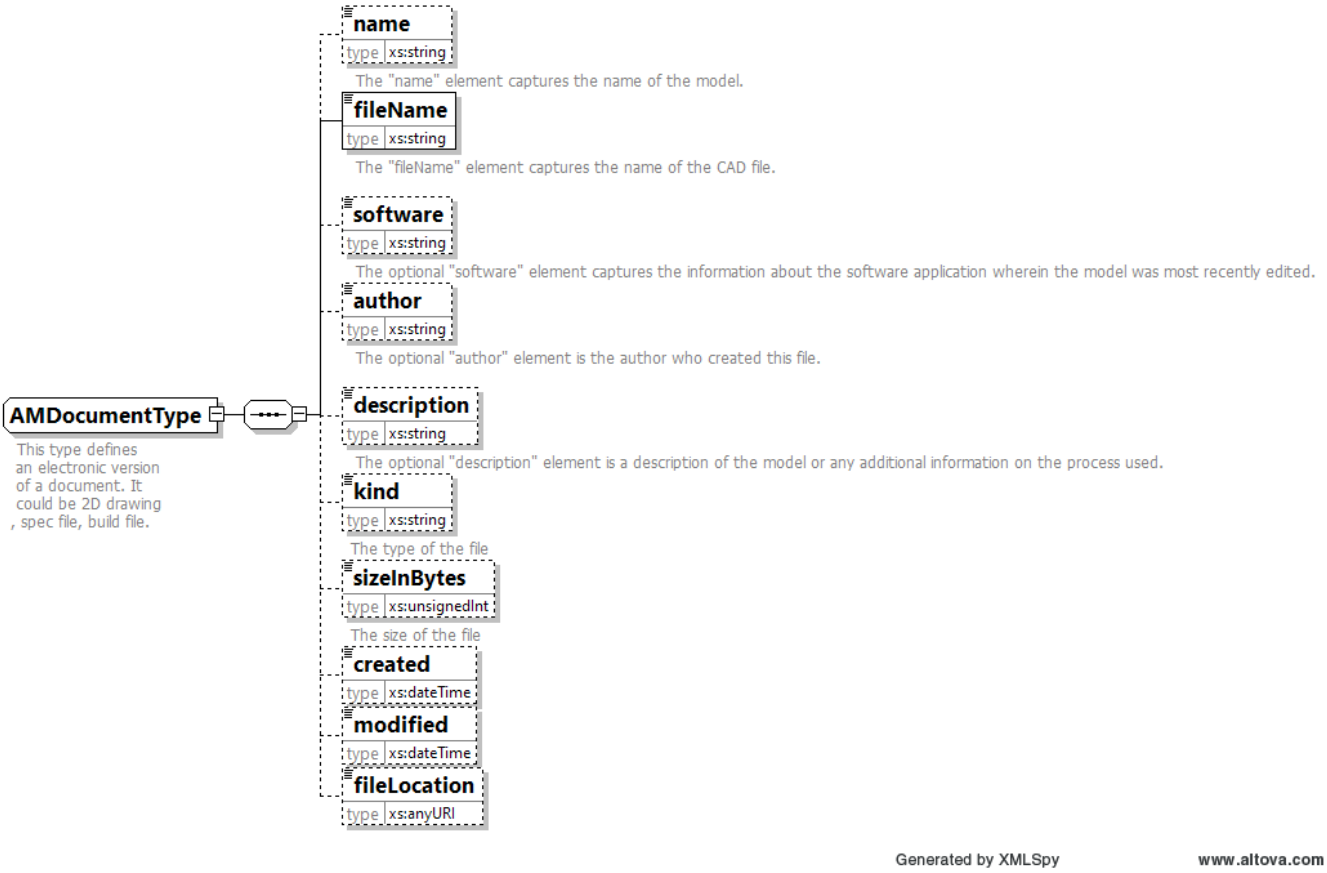

Figure 17: AMMD's AMDocument element [8]

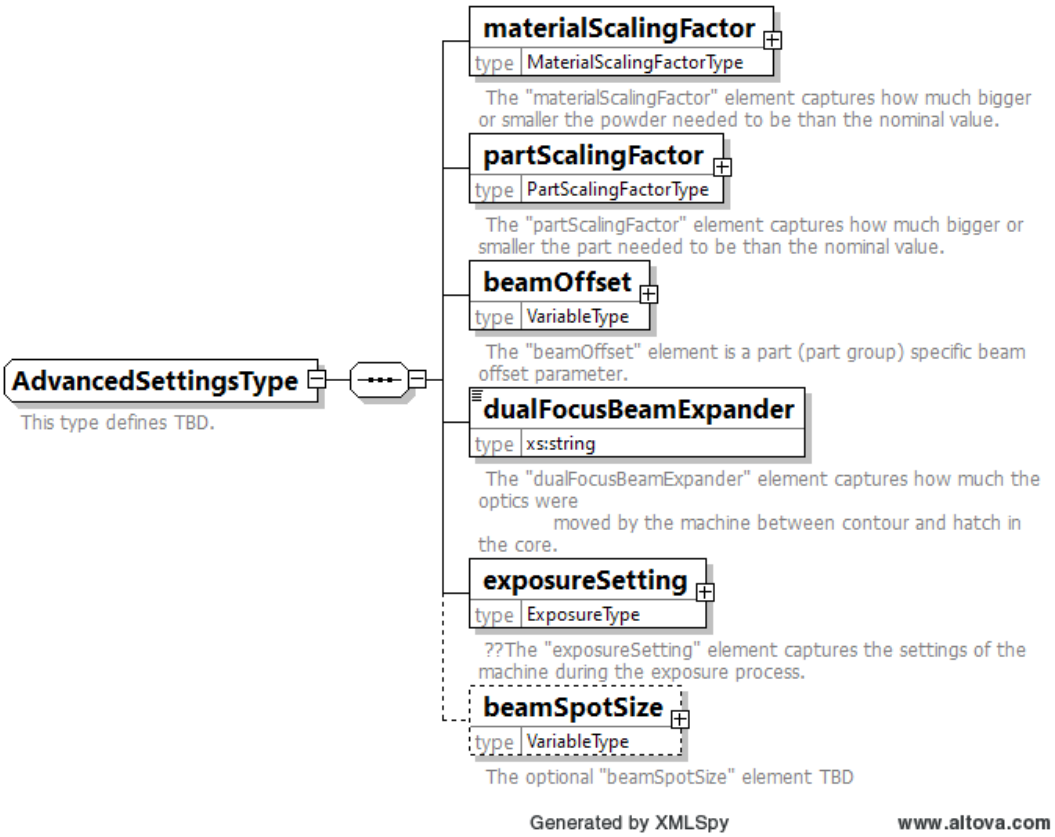

Figure 18: AMMD's AdvancedSettings element [8]

Figure 18 expands the AdvancedSettings element, which is arguably the most complicated child element of the EOS process plan. In it are contained a wide array of parameters that control the specifics of how scanpaths are calculated and executed, how geometry is interpreted and compensated, and with what settings the laser exposes the part's layers.

In the exposureSetting child element, sets of parameters capture how each region of the part is exposed. Each layer is exposed in four "phases", which are pre-exposure, skin-exposure, core-exposure and post-exposure. For each phase, a corresponding element captures the sets of settings that define the scanpath and laser behavior that are applied to it, and each set of settings has an {ON, OFF} enumeration element to capture whether it is active or not.

The *element* describing the behavior of the pre-exposure phase is optional. Note that this is not equivalent to specifying that the pre-exposure *phase* is itself optional, but it could be that the absence of the element may be interpreted as such. The remaining elements relate mainly to floating point quantities that specify the behavior of functions that define what regions are considered skin and which are not.

3 The Developed Functional Model for LPBF Processes

The functional model, implemented in IDEF0, will now be presented partially, focusing on the most relevant activities. The remainder of the IDEF0 diagrams are available in Appendix A.

3.1 A-0 Diagram

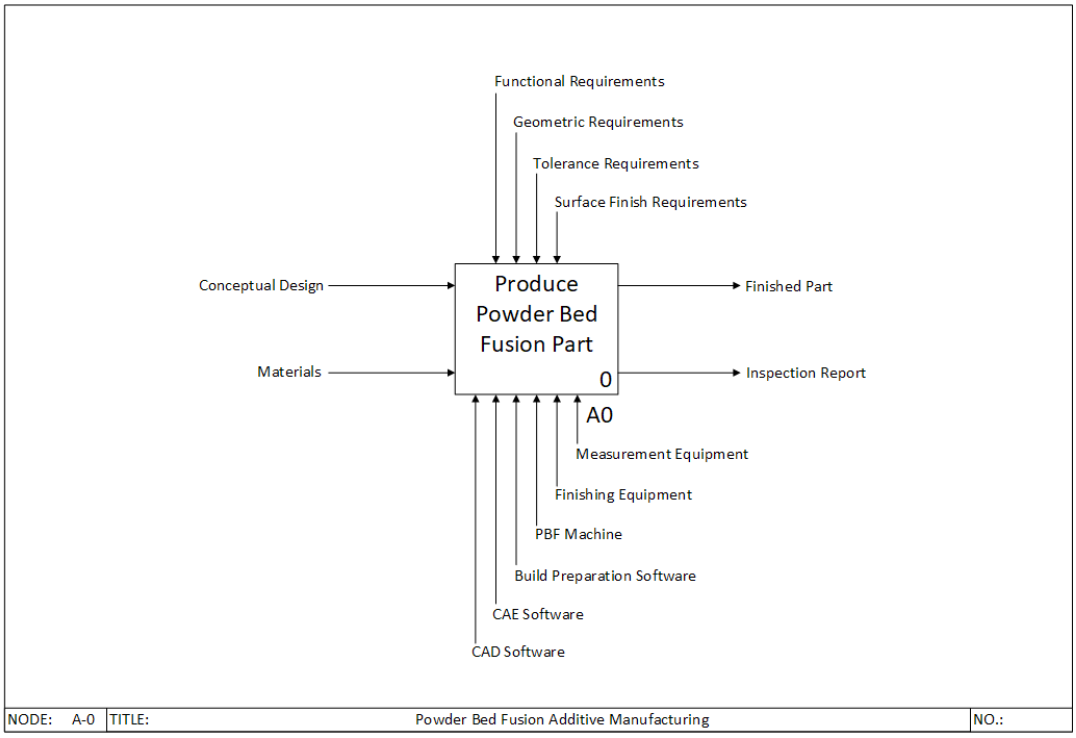

Figure 19: The A-0 (”A Minus Zero”) IDEF0 Activity Diagram

The A Minus Zero (A-0) diagram is a special case of IDEF0 diagram. No activity decomposition has yet taken place – instead, a top-level view examines the process being studied from the perspective of a black box. For LPBF additive manufacturing, this black-box approach tells us that we input information (in the form of a purely Conceptual Design) and physical metal powder (denoted by Materials) to yield the Finished Part, ready for use, along with an Inspection Report attesting to its conformance with any required specifications. What guides and modifies the specifics of the LPBF process are design requirements which can be divided into Functional, Geometric, Tolerance and Surface Finish requirements. The means, or Mechanisms, by which the activity is accomplished are the suite of engineering software and manufacturing, processing and metrology equipment found in modern manufacturing plants.

3.2 A0 Diagram

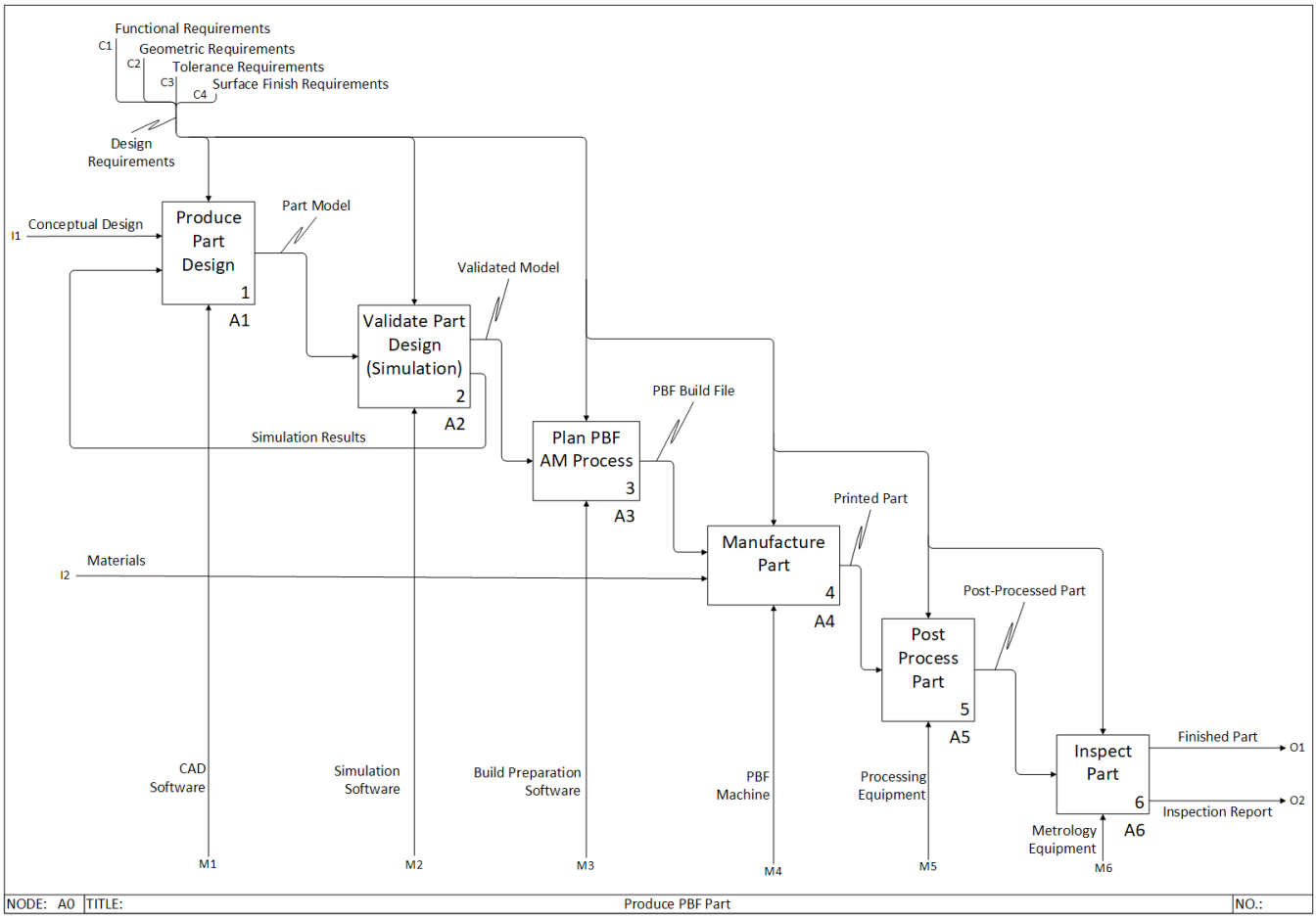

Figure 20: The A0 IDEF0 Activity Diagram

The A0 diagram is the first level of activity decomposition. By observing a Renishaw LPBF machine produce a test part from start to finish, the six sub-functions of Figure 20 were identified, along with their inputs and outputs. The design requirements, ultimately, pervade every step of the process – even if each has a different degree of influence on each sub-activity, they are intimately interconnected, and cannot be considered in complete isolation. Therefore, they are control inputs to all sub-activities. The simulation results that are output from *A2: Validate Part Design (Simulation)* are nearly always used to guide the fine-tuning of a part's design, hence the design-simulation loop between A1 and A2. However, some modern CAD/CAM software is capable of automating the loop with tools such as topology optimization and stress-aware generative design [16]. In such a case, the A1 and A2 activities would be combined into one.

3.3 A3 Diagram

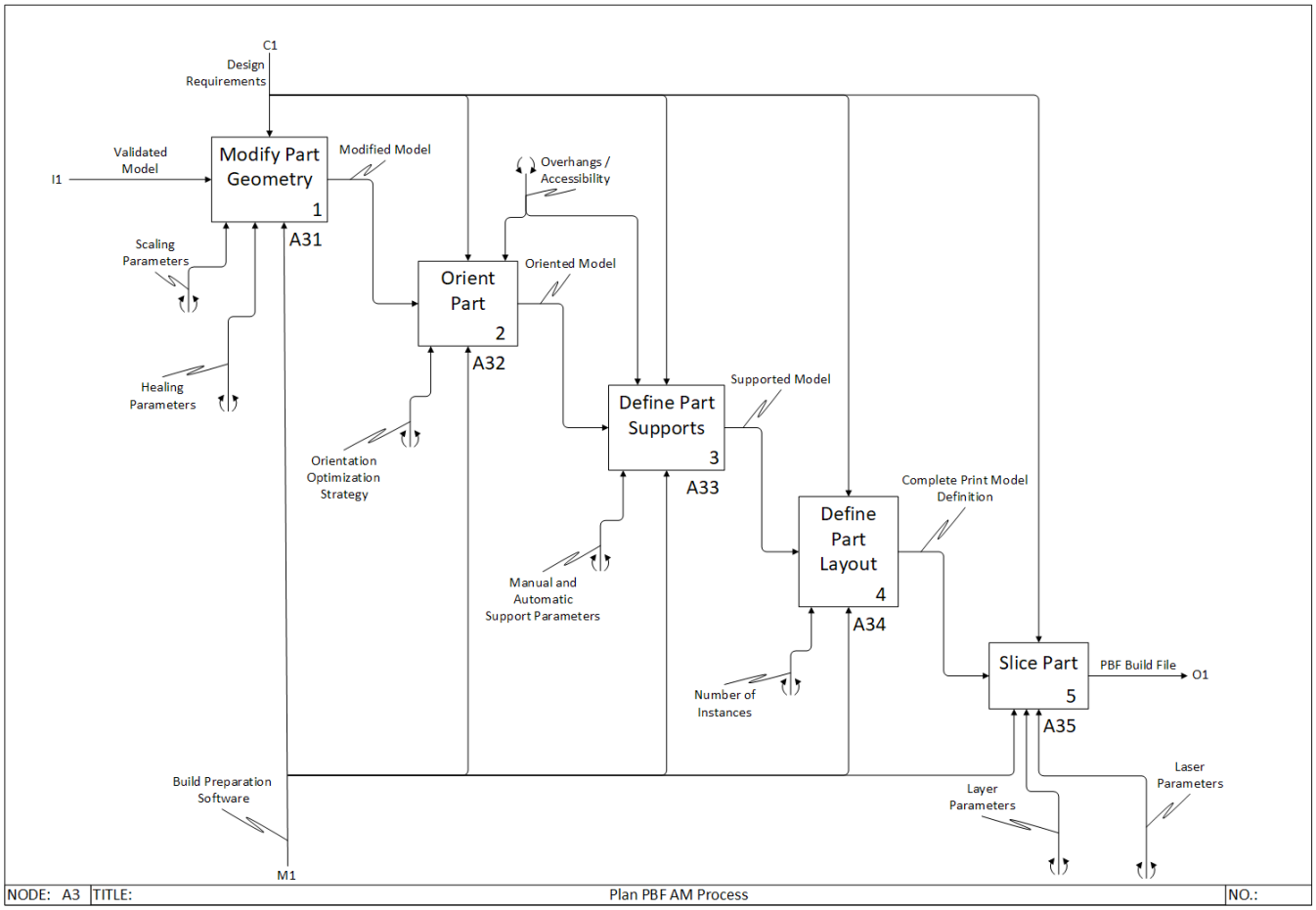

Figure 21: The A3 IDEF0 Activity Diagram

The A3 diagram is a second level of decomposition, focusing on the *Plan PBF AM Process* activity.

The first sub-activity, *A31: Modify Part Geometry*, captures basic pre-processing of an imported geometric model. It introduces two new Mechanisms, Scaling Parameters and Healing Parameters, that are used to modify the model to suit the needs of the user. Scaling need not be uniformly applied; different axes can be scaled differently from one another, and even non-linearly. Healing refers to the practice of ”repairing” surface-based geometry definitions (e.g. STL files) which contain problems such as missing or inverted elements [17]. *A32: Orient Part* represents rotations of the model to achieve proper access of important regions and a minimization of overhangs. Such accessibility concerns are modelled as a Control, and the possible use of an optimization strategy for orientation selection is modelled as a Mechanism. *A33: Define Part Supports* is tightly coupled with the previous activity; overhangs and feature accessibility also

strongly influence the location and quantity of printing supports. *A34: Define Part Layout* is meant to capture the option of a user to create a pattern of parts from an imported model so that multiple instances of the desired part can be produced at once. Such a strategy confers significant productivity benefits. Finally, *A35: Slice Part* captures the definition of a sequence of fully-defined scanpaths, including all relevant Layer and Laser Parameters, necessary for the manufacture of the part by the LPBF machine. The final output is therefore a Build File, embodying the global LPBF process plan.

4 The Developed Information Model for LPBF Process Plans

The information model, implemented in XML Schema, will now be presented partially. Similar to section 2.3, graphical representations generated in XMLSpy will be used, focusing on the most important elements. The full set of element diagrams are available in Appendix B. It is repeated that the model was developed by investigation of three sources: the build preparation software manual from Renishaw, the build preparation software manual from EOS, and the sections of the AMMD Schema that overlapped with the present work.

4.1 PolymorphicProcessPlan Element

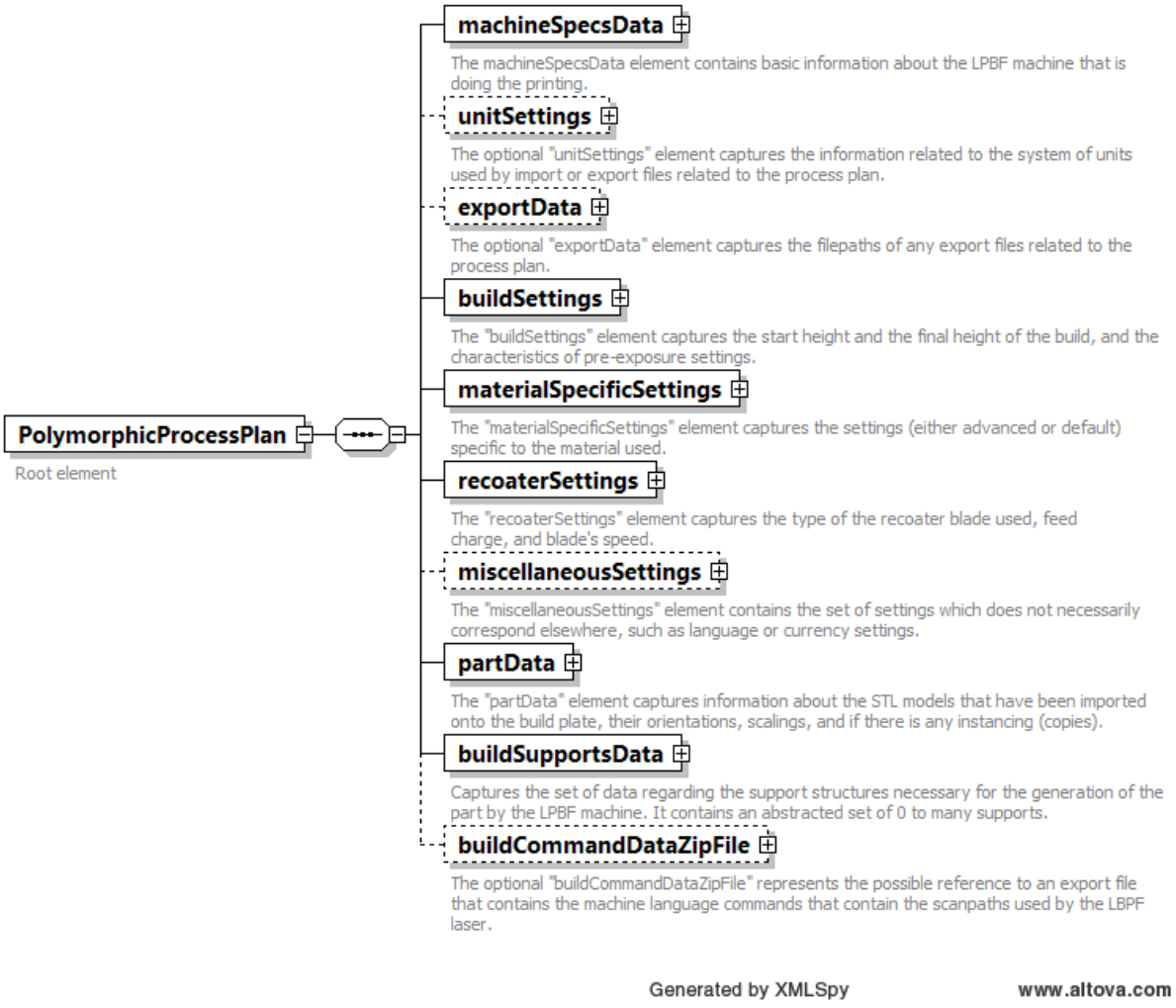

Figure 22: The PolymorphicProcessPlan element

The root element of the XML Schema is called ”PolymorphicProcessPlan”. The term ”polymorphic” is used in its computer science interpretation to denote an entity which can serve as an encompassing interface for multiple different types. In the current context, the meaning is that whether a given process plan was developed by one LPBF manufacturer or another, the ”PolymorphicProcessPlan” model can be applied to it. Upon application, the model will adapt to the process plan's specifics, thereby taking on its morphology – hence, polymorphism.

A schema definition of an LPBF process plan must contain and organize (1) the data structures, (2) the printing parameters, and (3) the system configurations that comprise a build file.

The PolymorphicProcessPlan is a Sequence of ten elements. Among them, elements exportData, partData, buildSupportsData, and buildCommandDataZipFile are used to model the data structures of the LPBF process. The printing parameters are captured by buildSettings and materialSpecificSettings. Finally, the system configurations are contained within machineSpecsData, unitSettings, recoaterSettings and miscellaneousSettings.

Four of the ten elements, unitSettings, exportData, miscellaneousSettings, and buildCommandDataZipFile, are considered optional. The unitSettings element is designed to capture, via appropriate enumeration datatypes, the system of units used by any imported or exported files used by the Build Preparation Software that generated the build file. Since such files may or may not exist, the element is optional. The exportData element is related, and captures the names and locations of ancillary files produced by the software to accompany the primary build file. These ancillary files might also, potentially, contain process planning information. However, they are not always used and therefore are modelled as optional. The miscellaneousSettings element is a catch-all location for settings like language selection or the currency used, if any. The buildCommandDataZipFile element, originally from the AMMD schema, represents an abstract model of a zip file which contains the set of scanpath commands used to control the LPBF laser.

4.2 buildSettings Element

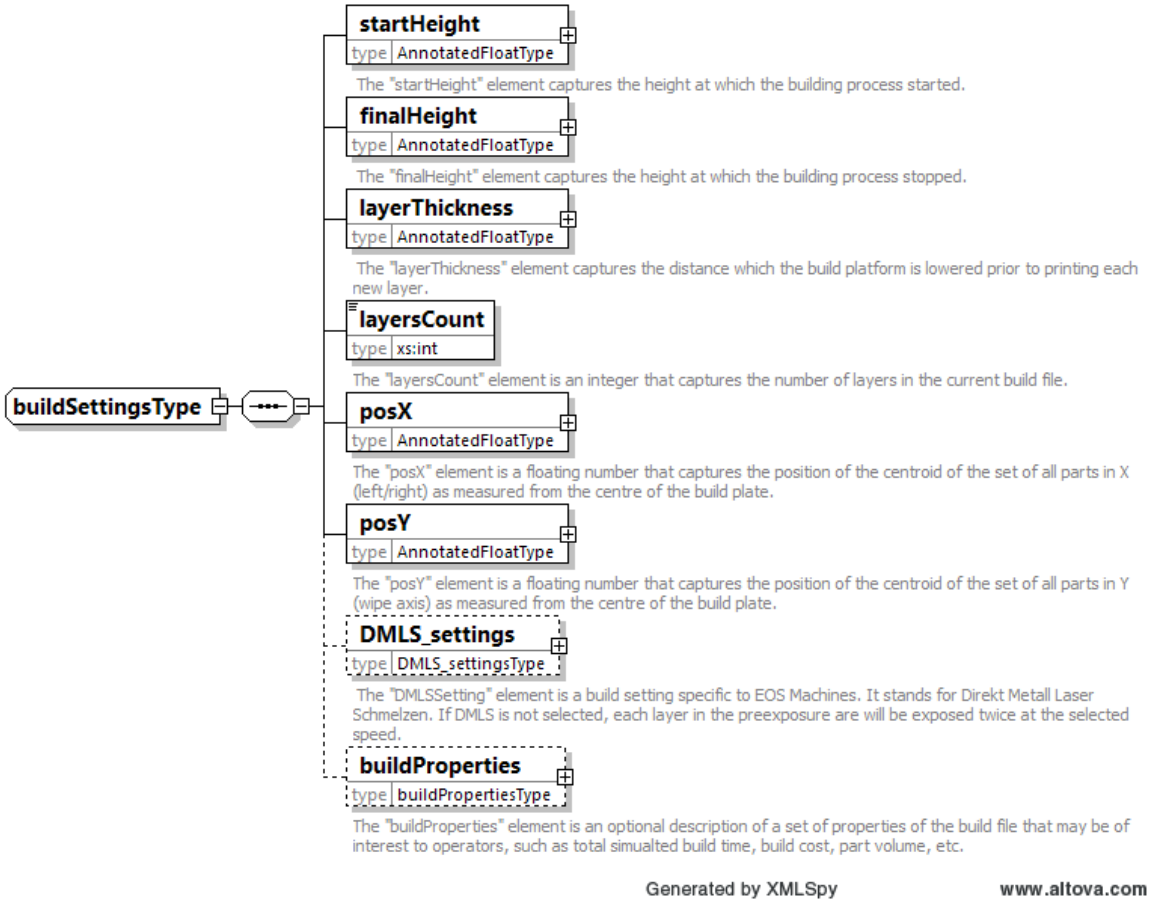

Figure 23: The buildSettings element

The buildSettings element was modelled as a means of including the basic set of build parameters and properties encountered in the Renishaw and EOS process plans. These parameters include floating point numbers that encode the thickness of each layer, the start and final heights of the printing process, etc. Note that whenever a quantity is a floating point number that requires a unit to be fully meaningful (e.g. 3.5 mm), the AnnotatedFloatType XML type is used. This type is a Sequence of elements that contain, at a minimum, the floating point value and a string datatype describing the unit of measurement. In this way, the information model can capture process plans written in any set of units. The buildSettings element also captures a set of settings unique to EOS (hence its optionality) that allow for layers to be doubly exposed, named DMLS. Finally, the model has provision for an optional buildProperties child element which is used to capture any useful statistics outputted by the software, such as an estimation of build time, build cost, the volume of support structures, etc.

4.3 materialSpecificSettings Element

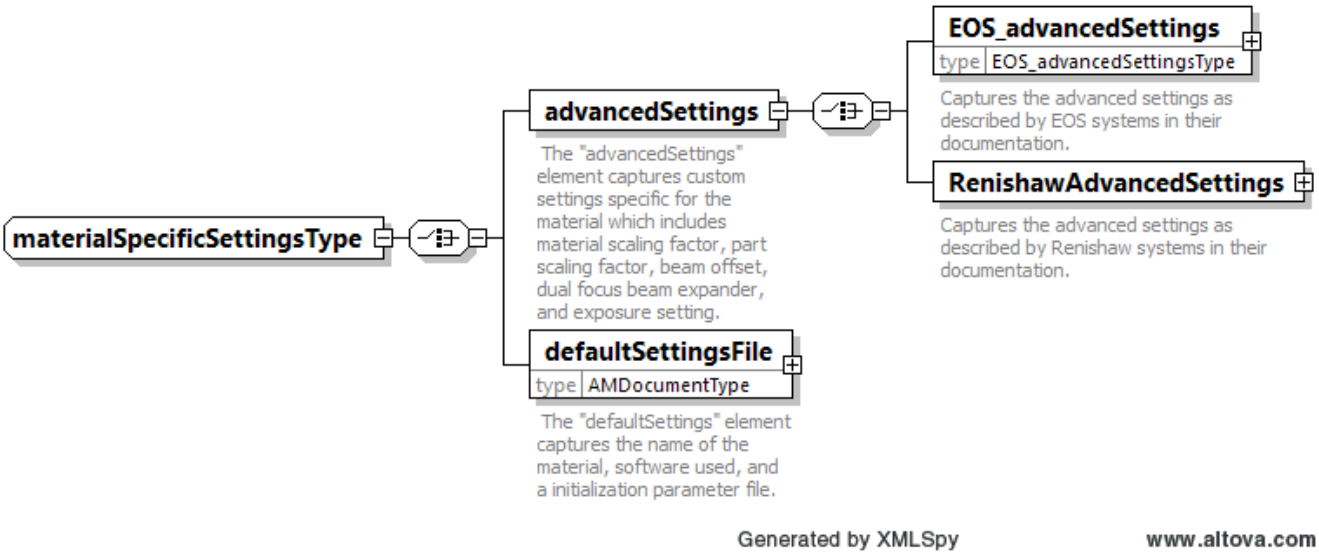

Figure 24: The materialSpecificSettings element

The materialSpecificSettings element is meant to polymorphically capture the set of advanced printing parameters which are specific to a given metal powder. There are two Choices nested in this element. The first Choice concerns whether the printing parameters are specified by user input, in which case the advancedSettings element is used, or whether they are specified externally by some instance of the abstract AMDocumentType. The AMDocumentType is originally from the AMMD schema and was further explained in section 2.3.

The second Choice is between the EOS view of the advanced settings and the Renishaw view. There was some hope at the outset of the present thesis that all parameters, even the subtler printing parameters, could be satisfactorily modelled with the aid of generic categories. By generic categories, it is meant that one could conceivably create some universal list of AM settings to which, somehow, the individual lists of manufacturer-specific AM settings could be mapped with a bijective (one-to-one) correspondence. Unfortunately, such a universal list (and bijective mappings) do not appear to exist in reality. If they did exist, and the schema modelled them, they would constitute a very strong form of polymorphism.

Therefore, the only logical approach remaining was to model them separately, but provide a Choice feature for the schema to remain flexible.

4.4 EOS_advancedSettings Element

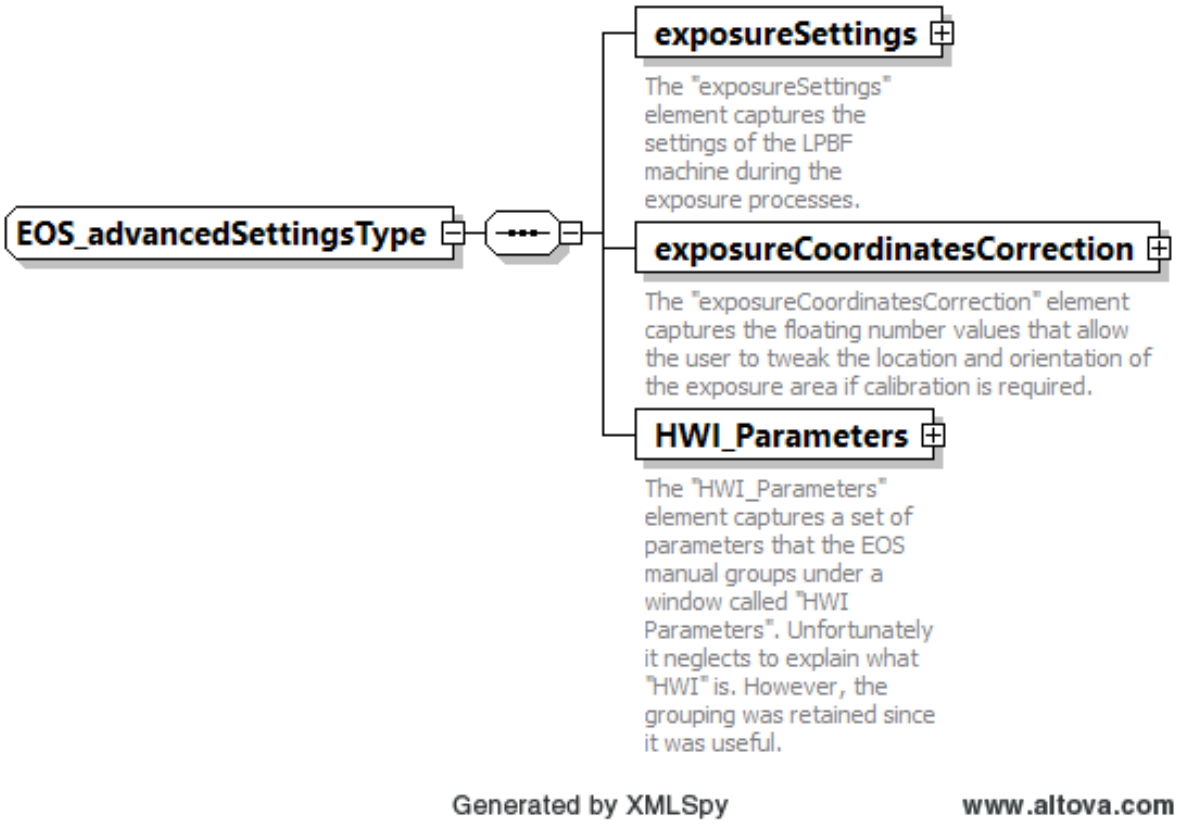

Figure 25: The EOS_advancedSettings element

The EOS_advancedSettings element captures the user-specified, material-specific, laser-exposure and other associated parameters of the EOS process plan. The second child element captures an interesting calibration feature called Exposure Coordinates Correction, which allows for compensation of translation and rotational offset errors of the exposed area on the build plate [5]. The HWI_Parameters element contains some system configuration parameters that involve environmental quantities (e.g. O2 levels in the build chamber, air supply enabling, build platform nominal temperature, etc.).

The exposureSettings child element governs the specifics of how the LPBF machine uses the laser to expose each layer of the part. Given its importance, this element will be further discussed in the next subsection.

4.5 exposureSettings Element

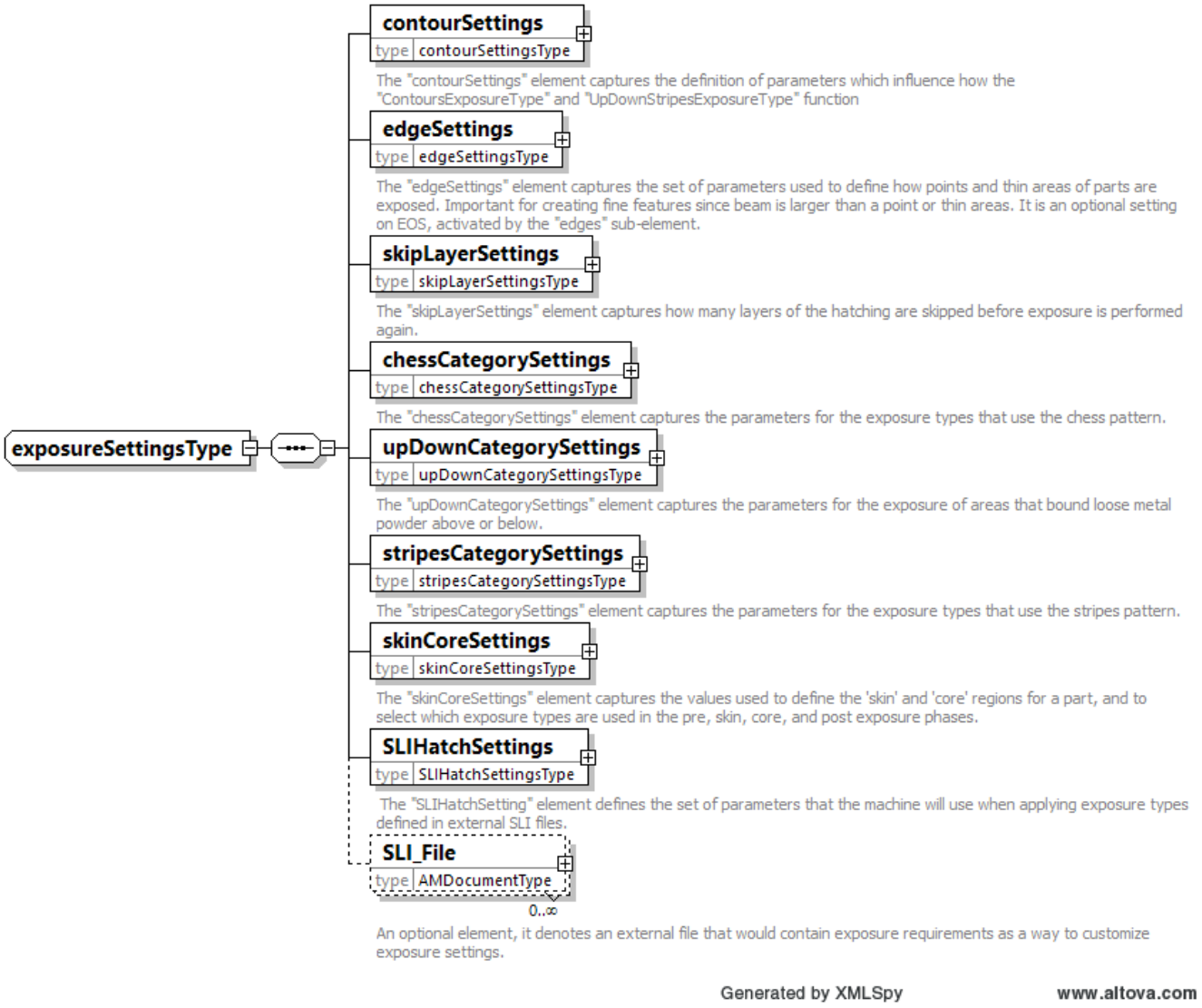

Figure 26: The exposureSettings element

The exposureSettings element is critically important. It is responsible for specifying exactly how the laser is to expose the metal powder, from which scanpath strategy to choose, to how that scanpath is optimized, what speed and power the laser utilizes, how much overlap there is between hatching passes, by what distance contour geometry is to be offset, and much more. However, the way it specifies this behavior is not entirely straightforward.

Based on the thesis author's understanding of the software manual [5], in the EOS ecosystem, layers are exposed in four phases: pre-exposure, skin-exposure, core-exposure, and post-exposure. It seems that the pre-exposure and post-exposure phases are for the contours of a given layer, the skin-exposure phase is for the skin regions of the layer, and the core-exposure is for the interior core regions of the layer.

The user can assign, to each of these four phases, an *Exposure Type*. An Exposure Type represents a certain *class* of behaviors that describe precisely how the laser exposes the regions of the associated phase. Now, what governs the specific numerical parameter values used by each Exposure Type are the groups of settings modelled as the first six child elements of exposureSettings. In other words, the first six child elements are sets of settings specified by the user, which are then used to define how a different set of settings called Exposure Types (not directly controlled by the user) behave. Then, it is those Exposure Types that the user can select to be applied to the four exposure phases. In this manner, the user controls the exposure settings in an indirect fashion.

The remaining child elements of exposureSettings are skinCoreSettings, SLIHatchSettings, and SLI_File. The skinCoreSettings element will be discussed in detail in the next subsection. The SLIHatchSettings element defines a set of parameters which are used by the LPBF controller to apply Exposure Types that have been defined in a set of optional external files called SLI Files. Those SLI Files are therefore modelled abstractly as instances of AMDocumentType, with the name SLI_File, and with unbounded multiplicity.

4.6 skinCoreSettings Element

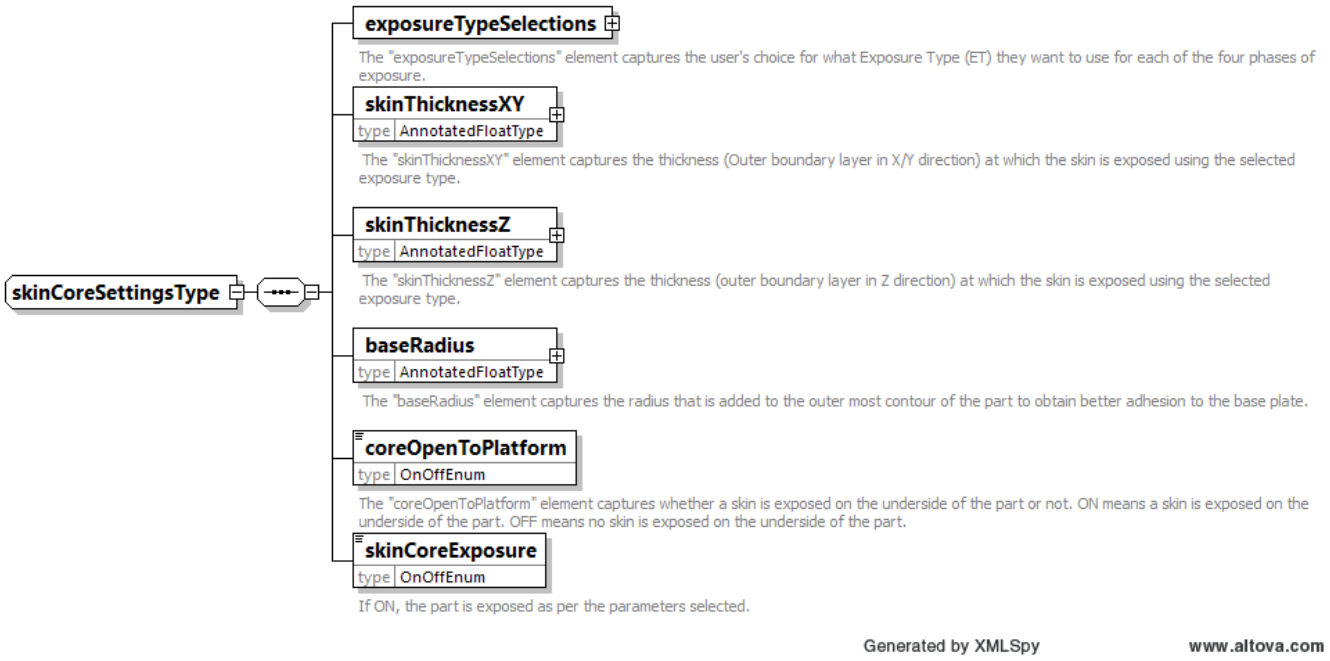

Figure 27: The skinCoreSettings element

The skinCoreSettings element captures a few important things. Firstly, in its exposureTypeSelections child element, it models the user's decision on which Exposure Types are assigned to each of the four exposure phases, as described in the previous subsection. Secondly, it captures parameters that affect the LPBF controller's treatment of the skin regions. Thirdly, its child element baseRadius can affect the likelihood of a successful print by specifying the radius that is added to the outermost contour of the part to yield improved adhesion to the build plate. These parameters are modelled as instances of AnnotatedFloatType to avoid possible ambiguity regarding the units used.

Notably, the skinCoreExposure child element is an {ON, OFF} enumeration datatype that, according to the EOS manual, controls whether "the part is exposed as per the parameters selected" [5]. This is very peculiar, since if it is OFF, then that would mean one of two possibilities. Either the part is *not* exposed, which would mean no part is being produced at all, or the part *is* exposed, but not according to the parameters selected. If the latter case is what is meant, it begs the question: what parameters are used, then, if not the ones specified by the user?

4.7 exposureTypeSelections Element

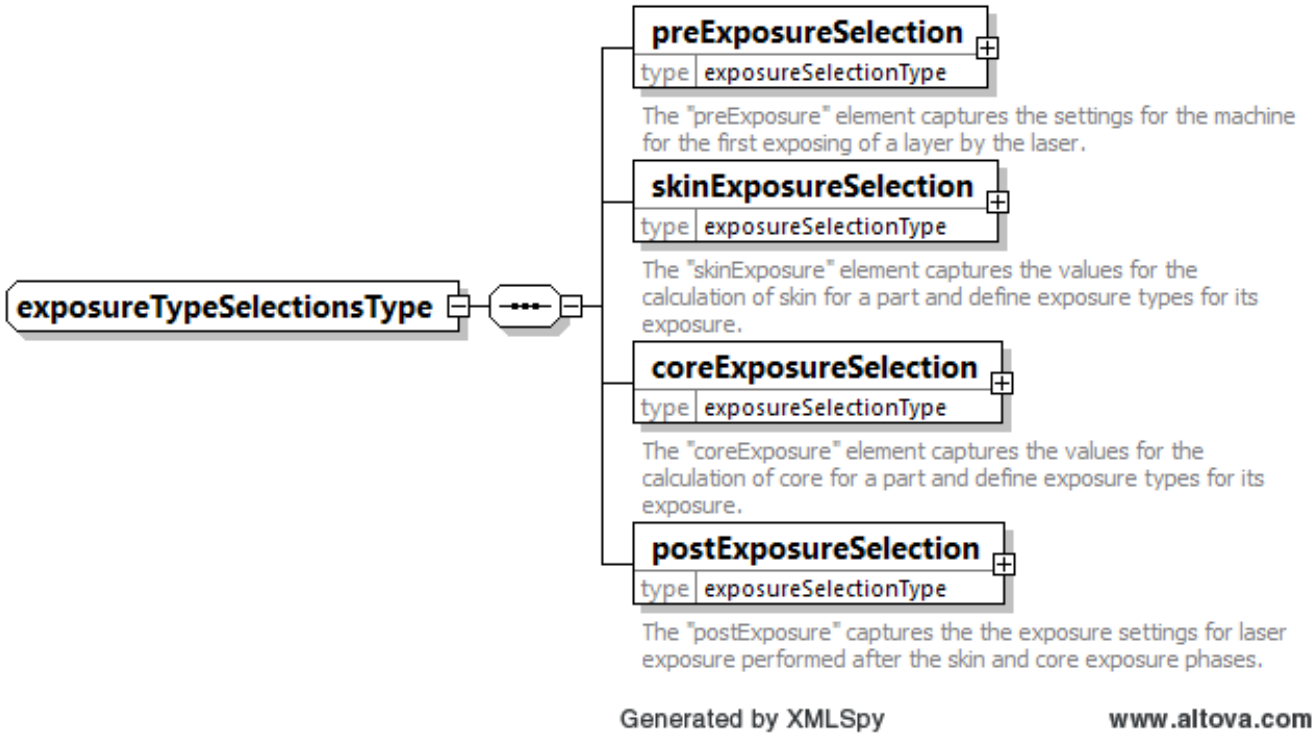

Figure 28: The exposureTypeSelections element

The last element concerning the EOS process plan that shall be discussed is the exposureTypeSelections element. As mentioned previously, it is responsible for capturing the user's selection of which Exposure Type is assigned to each of the four exposure phases. The possible options are named: Contours, Sorted, Unsorted, UpDownskin, Stripes, StripesLx, UpDownStripes, Chess, ChessLx, Squares, SquaresLx, SLI_Hatch, SLI_HatchLx, and No_Exposure. Interestingly, the last Exposure Type allows for users to simply elect not to perform one of the exposure phases.

Appendix B.17 contains descriptions of what each Exposure Type does.

4.8 RenishawAdvancedSettings Element

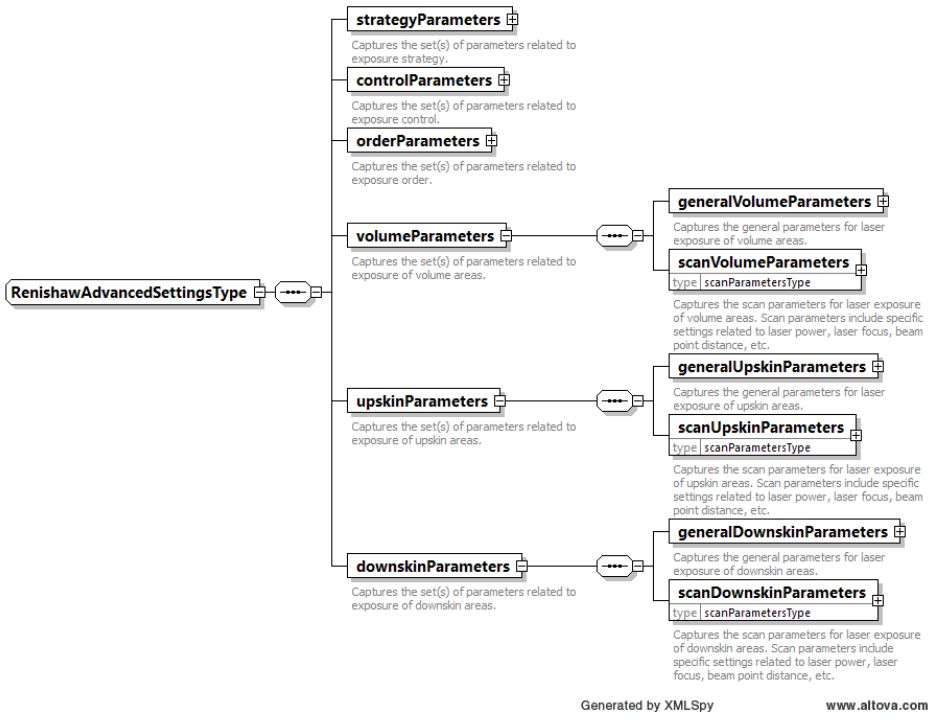

Figure 29: The RenishawAdvancedSettings element

We shall now switch the focus of discussion towards the modelling of Renishaw process plans.

In the RenishawAdvancedSettings element, there are six child elements. The first three, strategyParameters, controlParameters, and orderParameters, are applied in a ”global” sense. That is, the settings contained therein will control the behavior of laser exposure for the entire set of parts on the build plate. This is in contrast with the last three child parameters, which are applied when the laser is exposing their respective areas. Therefore, volumeParameters applies to volume areas, upskinParameters applies to upskin areas, and downskinParameters applies to downskin areas. The three last child elements are each subdivided into two: general[...]Parameters and scan[...]Parameters. The latter element contains settings that control the laser's inherent properties (e.g. laser power, laser focal plane distance, laser exposure times, distance between laser points, etc.) and the former element contains parameters that control any other quantities (e.g. geometric offsets, hatching angles, startng scan angles, etc.).

The logic behind modelling the Renishaw process plan in this manner was that it closely follows the way in which the parameters are structured within the build preparation software itself, which was already satisfactory from a data modelling perspective.

4.9 strategyParameters Element

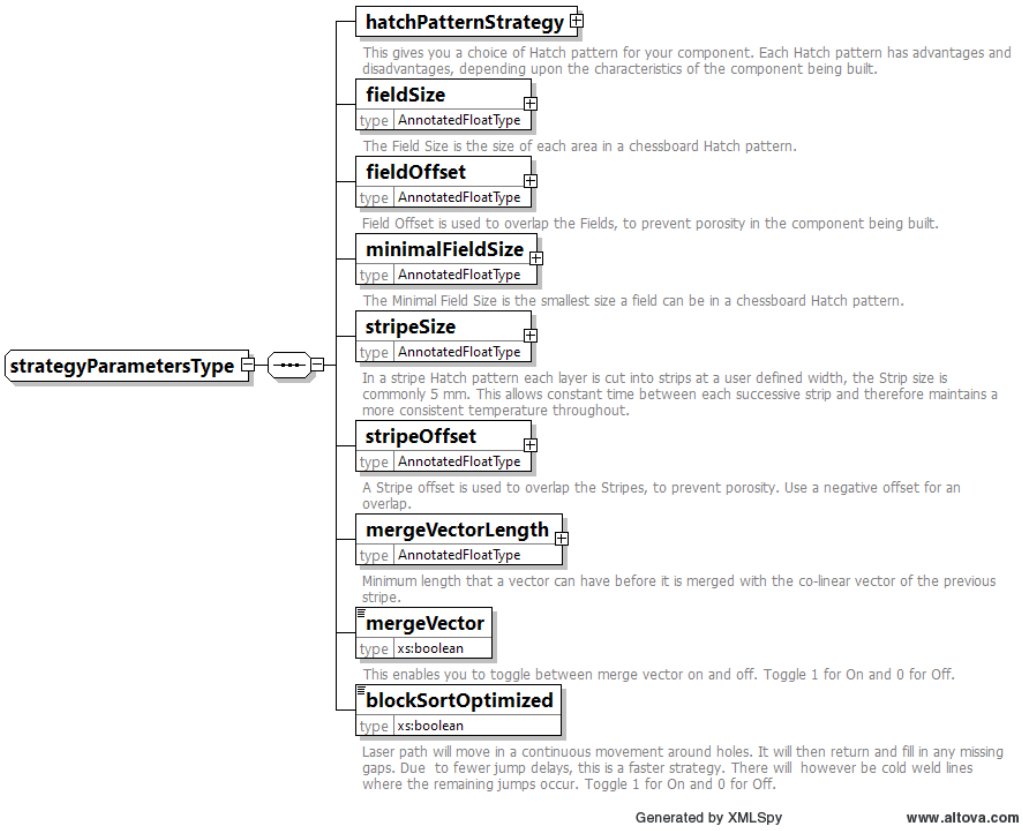

Figure 30: The strategyParameters element

The strategyParameters element is included in the select subset of presented schema elements to provide both an example of the lowest level of LPBF printing parameters, and because it is responsible for selecting the hatching scanpath strategy in the Renishaw system.

Regarding the low-level parameters, these are all simple datatypes with no children elements. Mostly, they are instances of the AnnotatedFloatType, but some are booleans as well. Figure 30 contains descriptions of what each setting does, and it is worth noting how much control the user has over the minutiae of the LPBF process. Indeed, there are over one hundred of such highly-specific parameters, which illustrates the difficulty of process planning exchange.

Regarding the scanpath strategies, there are three choices in the Renishaw system: Meander, Chessboard, and Stripe. Meander is a zigzag pattern that skips holes in the part. Chessboard is based on square blocks whose scan lines are normal to scan lines in adjacent blocks. Stripe is like Meander, except the scan lines are broken up into fixed length stripes.

4.10 recoaterSettings Element

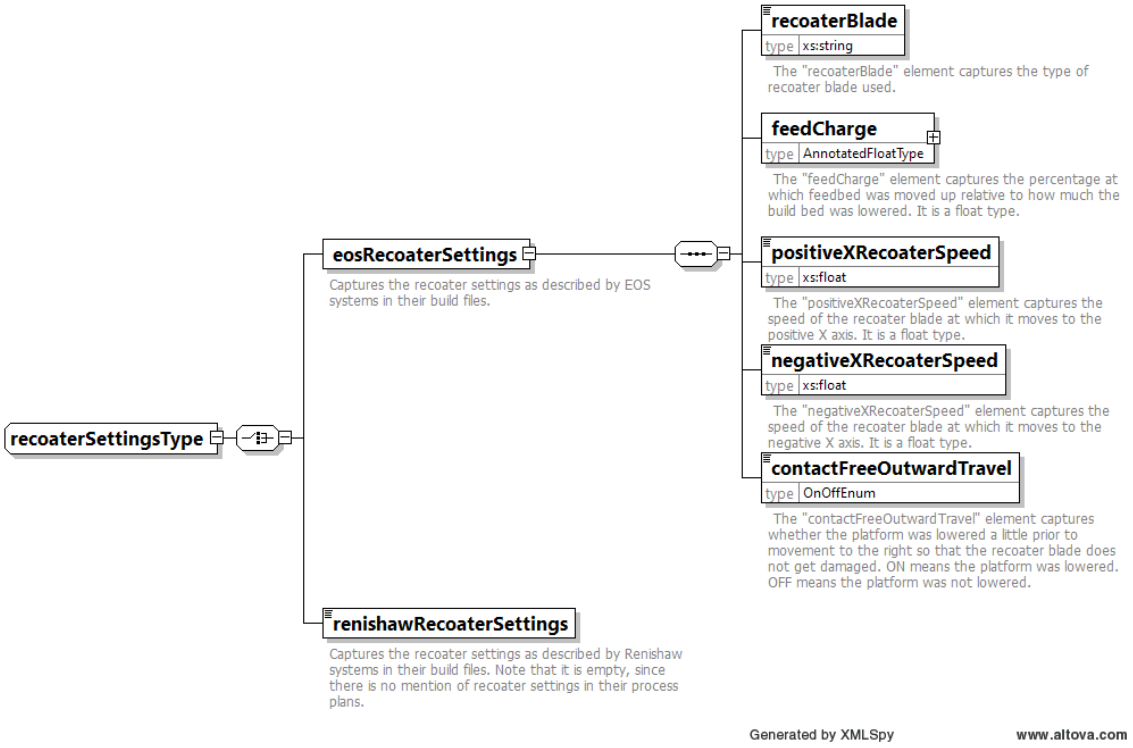

Figure 31: The recoaterSettings element

The recoaterSettings element is an example of information that belongs to the system configuration category, as outlined in section 4.1. The element has a Choice of one child element, among two possibilities: the EOS version and the Renishaw version. The eosRecoaterSettings element's modelling is the same as in the AMMD schema, and has been described previously in the discussions found in section 2.3.

What is most notable is that the renishawRecoaterSettings element has no child elements. It is purposefully empty; the build preparation software does not request any input from the user regarding the recoater configuration and associated settings [18]. Nevertheless, the Renishaw LPBF machines must somehow specify the behavior of their recoaters. Such settings, along with build chamber environment settings, are controllable directly from the machine HMI (Human-Machine Interface). It is the understanding of the thesis author that when a build file is loaded onto the LBPF controller, it can detect the specific material used for the metal powder, and set the unspecified recoater and environmental parameters according to corresponding entries in an internal material database. Since Renishaw manufactures its own AM metal powders, this is achievable through testing that was performed in advance by the manufacturer.

4.11 partData Element

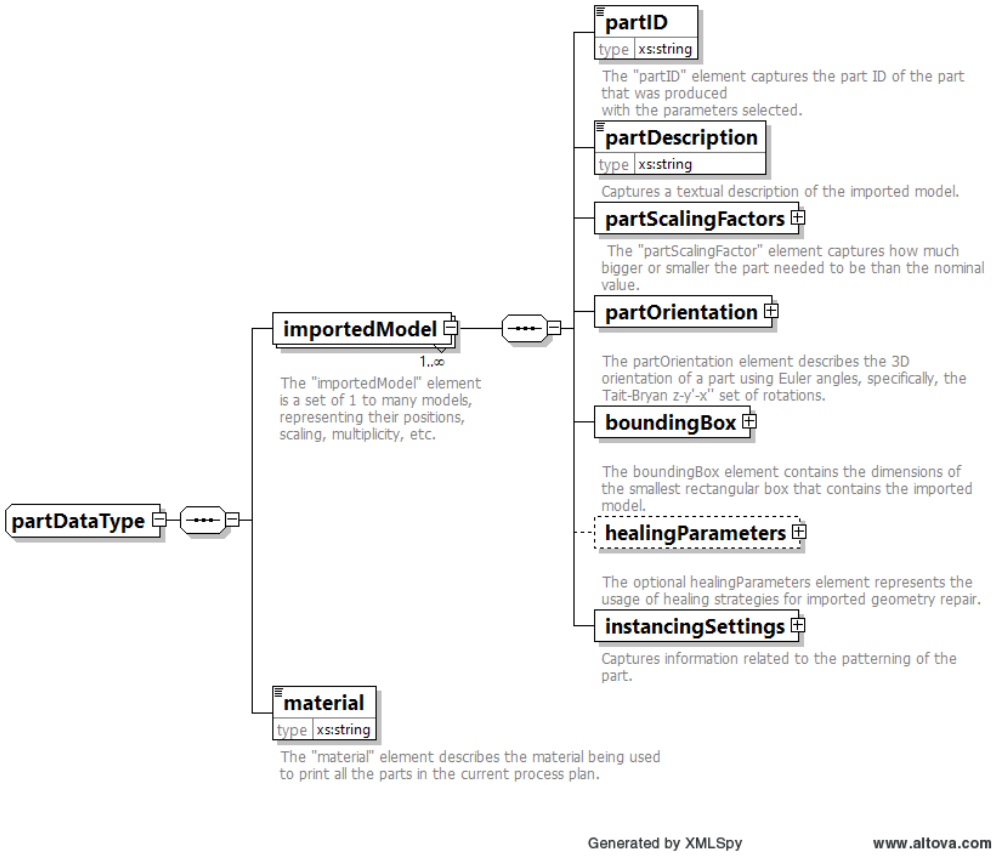

Figure 32: The partData element

The modelling of information regarding all the parts being printed on the build plate is handled by the partData element. It is a Sequence of two elements: ”material”, which is a string datatype containing the name of the metal powder, and a set of any number of instances of the ”importedModel” element.

The importedModel element already models the possibility of multiple copies of a given part existing on the build plate – this will be further discussed in the next subsection. Therefore, the reason for having the schema allow for an unbounded multiplicity of importedModel instances is that entirely different parts can be printed in a single printing session. This is common practice in AM for time saving purposes.

The children elements handle any scaling of the imported models, as well as their location and orientation with respect to the build plate area. It was decided to model the part rotation with Euler angles, assuming by default that rotations are in the z-y’-x” order, but this can be changed by modifying an XML element attribute.

4.12 instancingSettings Element

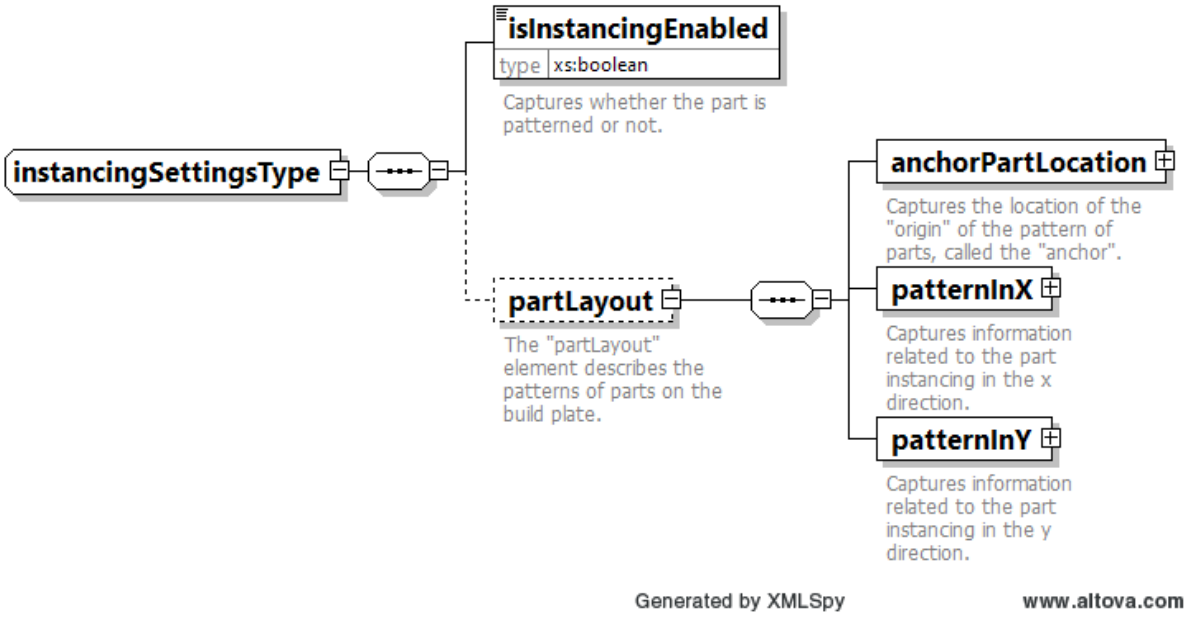

Figure 33: The instancingSettings element

The instancingSettings element captures firstly whether any copies of a given imported model are to be printed through the isInstancingEnabled boolean element. If it is enabled, then a child element called partLayout exists and is a Sequence of a location for the pattern to start from, and the parameters that describe the rectangular pattern in which copies are to be arranged (i.e. the spacing distance and the number of copies in each direction).

Only the rectangular kind of pattern is modelled since it is the only model that was observed in the documentation. Conceivably, other types of patterns, such as circular patterns, are feasible. If another manufacturer is found that uses new pattern types, these can be added under an XML Choice feature.

The schema assumes that the orientation of all copies of a given model are identical, though strictly speaking this need not be the case. However, this can be achieved by adding a separate model onto the build plate that is generated from the same geometry file as the original part. Then this ”new” copy will have its own unique orientation, and can be patterned as desired to achieve two sets of copies with different rotations and anchor locations on the build plate.

4.13 buildSupportsData Element

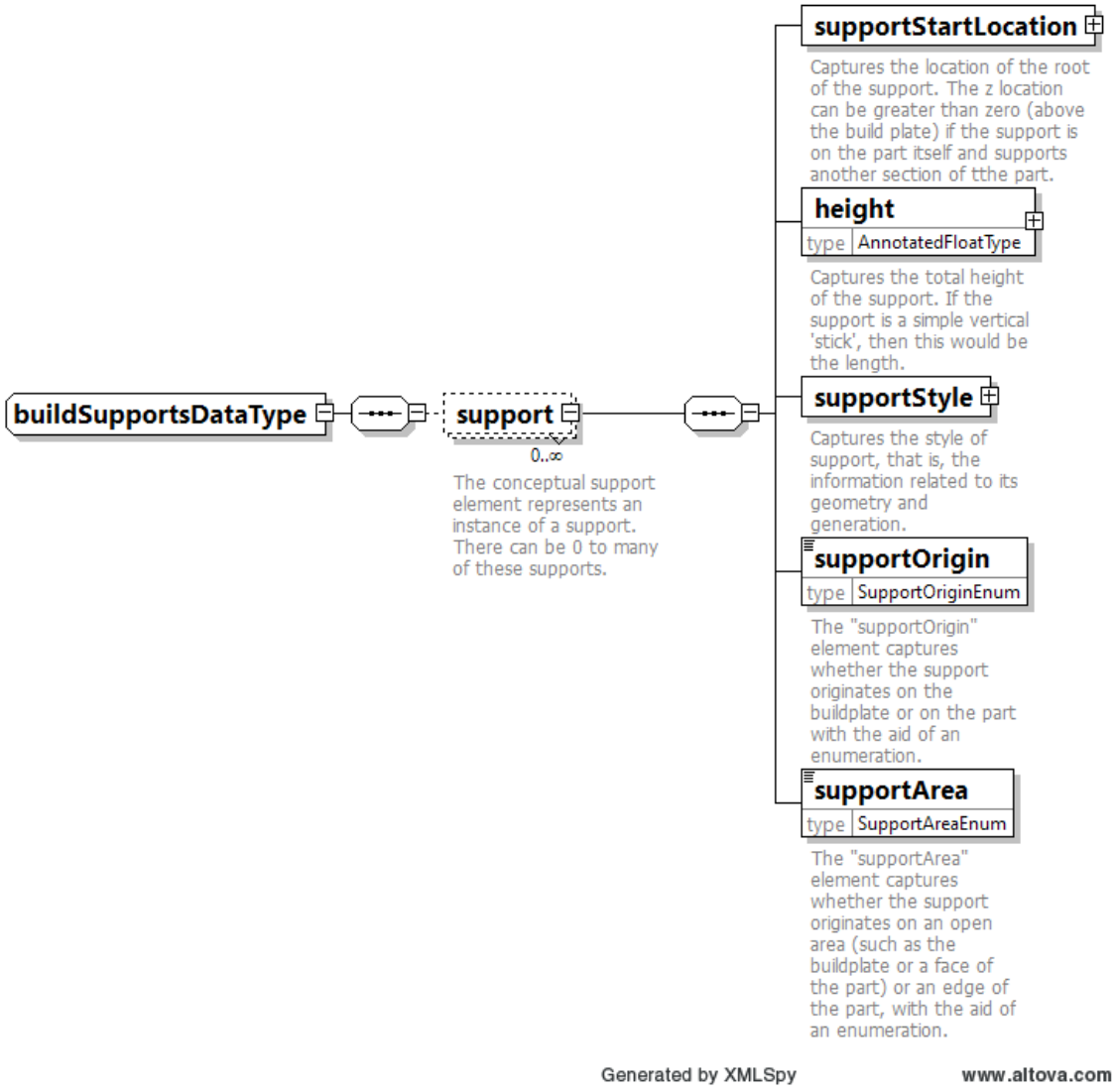

Figure 34: The buildSupportsData element

The last element that will be discussed is the buildSupportsData element.

The modelling of this AM feature is based on an abstract, generic view of supports. That is, with the knowledge that all printing supports must necessarily start somewhere and end somewhere, we can create elements to capture the starting position (”supportStartLocation” element) and the vertical extent of the support (”height” element). Similarly, support structures must either originate on the build-plate surface or else on the part itself – hence the ”supportOrigin” enumeration datatype used to capture this.

The most complex child element is the supportStyle element. It is a Choice of two elements: renishawSupportStyle and eosSupportStyle. The former is a Sequence of sets of parameters that define how the algorithms used by build preparation software behave to generate advanced support structures like trees and clusters. However, the latter element (eosSupportStyle) is an empty element. This mirrors the situation described previously regarding the recoaterSettings element, where it was the Renishaw version of the element that was empty. In this case, the EOS build preparation software does not appear to prompt the user to input the values of support generation parameters. It is possible that this is done automatically, or else that this is handled by a separate file.

4.14 Comparison of EOS and Renishaw Process Planning Information

The development of the schema highlighted several commonalities and differences between the approaches used by EOS and Renishaw to populate their build files.

Beginning with the commonalities, both manufacturers have similarly divided their laser parameters when they must be modified depending on the layer's current region type. That is, both approaches have parameters explicitly dedicated to downskin, upskin, contours/borders, and the volume/core regions. There are options in both process plans to optimize scan paths, for example by adjusting how and where laser exposure should be skipped, or which algorithm will calculate the trajectories. Since both build preparation softwares are at the professional level, they provide similar levels of granularity and customizability for the printing parameters. The ”top-level” settings, which represent quantities that would be common to all LPBF machines (e.g. part positioning, layer thickness, etc.), are more or less identical as well.

In what concerns the differences, terminology is naturally going to be among them. For example, when the laser must create the solidified outside boundary of a part, it is necessary to shift the laser some amount inwards so as to not create a part that is larger than designed. This shift in the Renishaw terminology is called ”Beam compensation” [18], while in the EOS terminology, it is referred to as the ”Beam offset” [5]. There are many hundreds of parameters, and some will actually be equivalent means of referring to the same physical quantity, but it might not be obvious due to different vocabularies. Furthermore, some concepts do not have equivalents from one ecosystem to the other. For example, as explained earlier in this section, the EOS build plan contains four user-selected Exposure Types, each controlling the specifics of laser exposure when applied to four ”phases”: pre-exposure, skin-exposure, core-exposure and post-exposure. Those two concepts, Exposure Types and ”phases” of exposure, cannot be translated to the Renishaw ecosystem for lack of an equivalent mechanism. Environmental parameters, such as build chamber oxygen levels or build plate heating temperature targets, appear to be explicitly user-queried by the build preparation software in the EOS case, while in the Renishaw case this might be handled in an external fashion. The same is true for the recoater configuration and settings. Regarding the parameters for printing support generation, the converse is true: the Renishaw software has the option of direct user input for settings which will control the algorithms that generate the supports, while there is no mention of this in the EOS manual.

5 Conclusions

The functional model developed in the present work was successful in providing a broad perspective of the LPBF domain. Several benefits were obtained from this preparatory step. By explicitly focusing on the manner in which the overall process is decomposed into sub-activities, there was achieved an improved understanding of the different types of information and the patterns into which they were organized. The IDEF0 diagrams allowed for the drawing of preliminary observations regarding the categories of process planning parameters. Finally, functional modelling permitted the construction of a clearly-delineated boundary for the information modelling activities that followed.

Using XML Schema, a moderately complex information model of LPBF process plans was constructed. The mentioned aim of ”polymorphism” was tentatively achieved despite inherent difficulties posed by competing definitions of data structures. The AMMD schema proved to be an invaluable aid, having already performed data modelling of aspects of the EOS process plan. In the lowest levels of the EOS decomposition, when dealing with individual parameters defined explicitly in the EOS documentation, the present model is in fact mostly identical to the AMMD schema. The larger structures of organization that contain these individual elements, however, differ between both.

By having highlighted commonalities and differences between the EOS and Renishaw process plans, it is hoped that future work in this field will benefit from this knowledge. However, the schema here presented is yet to be tested through case studies for the purposes of validation. The current state of the model is the result of a face-value understanding that proceeds directly from software documentation; only real-world verification of the data modelling concepts contained herein can raise that understanding to higher levels of usefulness. The most important future work, therefore, is the conducting of these case studies. In a secondary capacity, expansion of the current schema with information from different LPBF manufacturers would simultaneously increase the value of the model and aid in confirming its validity. If, for example, process planning information from General Electric Additive’s LPBF machines could be used to augment the current model, the market share that would be covered by the schema would rise above 50%.

In the long term, it is hoped that the combination of these open efforts can contribute towards the democratization of metal additive manufacturing, just as it has occurred with plastic 3D printers.

References

[1] Research And Markets, "Additive Manufacturing & Material - Global Market Trajectory & Analytics." researchandmarkets.com. https://www.researchandmarkets.com/reports/4804521/additive-manufacturing-and-material-global?utm_source=BW&utm_medium=PressRelease&utm_code=dv5mzh&utm_campaign=1591433+-+%2415.4+Billion+Worldwide+Additive+Manufacturing+%26+Material+Industry+to+2027+-+Latest+Advancements+and+Innovations&utm_exec=jamu273prd (accessed Dec. 1, 2021).

[2] Additive Manufacturing Media, "3D Printed Heat Exchanger Uses Gyroids For Better Cooling." additivemanufacturing.media. https://www.additivemanufacturing.media/articles/3d-printed-heat-exchanger-uses-gyroids-for-better-cooling-the-cool-parts-s (accessed Apr. 4, 2022).

[3] aniwaa, "Best metal 3D printer in 2022: comprehensive overview." aniwaa.com. https://www.aniwaa.com/buyers-guide/3d-printers/best-metal-3d-printer/ (accessed Apr. 4, 2022).

[4] Ansys Granta, "Additive Manufacturing Technologies Library." additivemanufacturing.media. https://www.grantadesign.com/education/teachingresources/ongoing-development/additive-manufacturing/ (accessed Apr. 4, 2022).

[5] EOS, *Process Software PSW Manual*, 2008. pp. 9.1-9.58.

[6] Y. Zhao, R. J. Brown, T. R. Kramer, and X. Xu, *Information Modeling for Interoperable Dimensional Metrology*. London, United Kingdom: Springer, 2011.

[7] ISO 6983-1:2009, "Automation systems and integration — Numerical control of machines — Program format and definitions of address words — Part 1: Data format for positioning, line motion and contouring control systems," standard, International Organization for Standardization, Geneva, CH, Dec. 2009.

[8] NIST, "AM Material Database." ammd.nist.gov. https://ammd.nist.gov/schema_viewer/ (accessed Apr. 4, 2022).

[9] S. C. Feng, P. Witherell, G. Ameta, and D. B. Kim, "Activity model for homogenization of data sets in laser-based powder bed fusion," *Rapid Prototyping Journal*, vol. 23, no. 1, pp. 137–148, 2017.

[10] IDEF0, "INTEGRATION DEFINITION FOR FUNCTION MODELING (IDEF0)," Standard, National Institute of Standards and Technology, Gaithersburg, MD, USA, Dec. 1993.

[11] Y. Lu, S. Choi, and P. Witherell, "Towards an integrated data schema design for additive manufacturing: Conceptual modeling," in *Proceedings of the ASME Design Engineering Technical Conference*, vol. 1A-2015, 2015.

[12] W3C, "XML Schema Part 0: Primer Second Edition." w3.org. https://www.w3.org/TR/xmlschema-0/ (accessed Dec. 1, 2021).

[13] SmarTech Analysis, "AM Material Database." smartechanalysis.com. https://www.smartechanalysis.com/blog/category/metal/ (accessed Apr. 4, 2022).

[14] NIST, "AMMT-TEMPS — NIST." nist.gov. https://www.nist.gov/el/ammt-temps (ac-

cessed Apr. 4, 2022).

[15] S. Shamsdini, S. Shakerin, A. Hadadzadeh, B. S. Amirkhiz, and M. Mohammadi, "A trade-off between powder layer thickness and mechanical properties in additively manufactured maraging steels," *Materials Science and Engineering A*, vol. 776, 2020.

[16] PTC, "Innovate Faster With Generative Design and AI." ptc.com. https://www.ptc.com/en/technologies/cad/generative-design (accessed Apr. 4, 2022).

[17] Dassault Systèmes, "STL File Problems: Common Problems Identified in the STL File Healing Process." blog.spatial.com. https://blog.spatial.com/stl-file-healing-process (accessed Apr. 4, 2022).

[18] Renishaw, *Renishaw QuantAM Build Preparation Software User Guide*, 2016.

Appendices

A IDEF0 Functional Decomposition Diagrams

A.1 A-0 Diagram

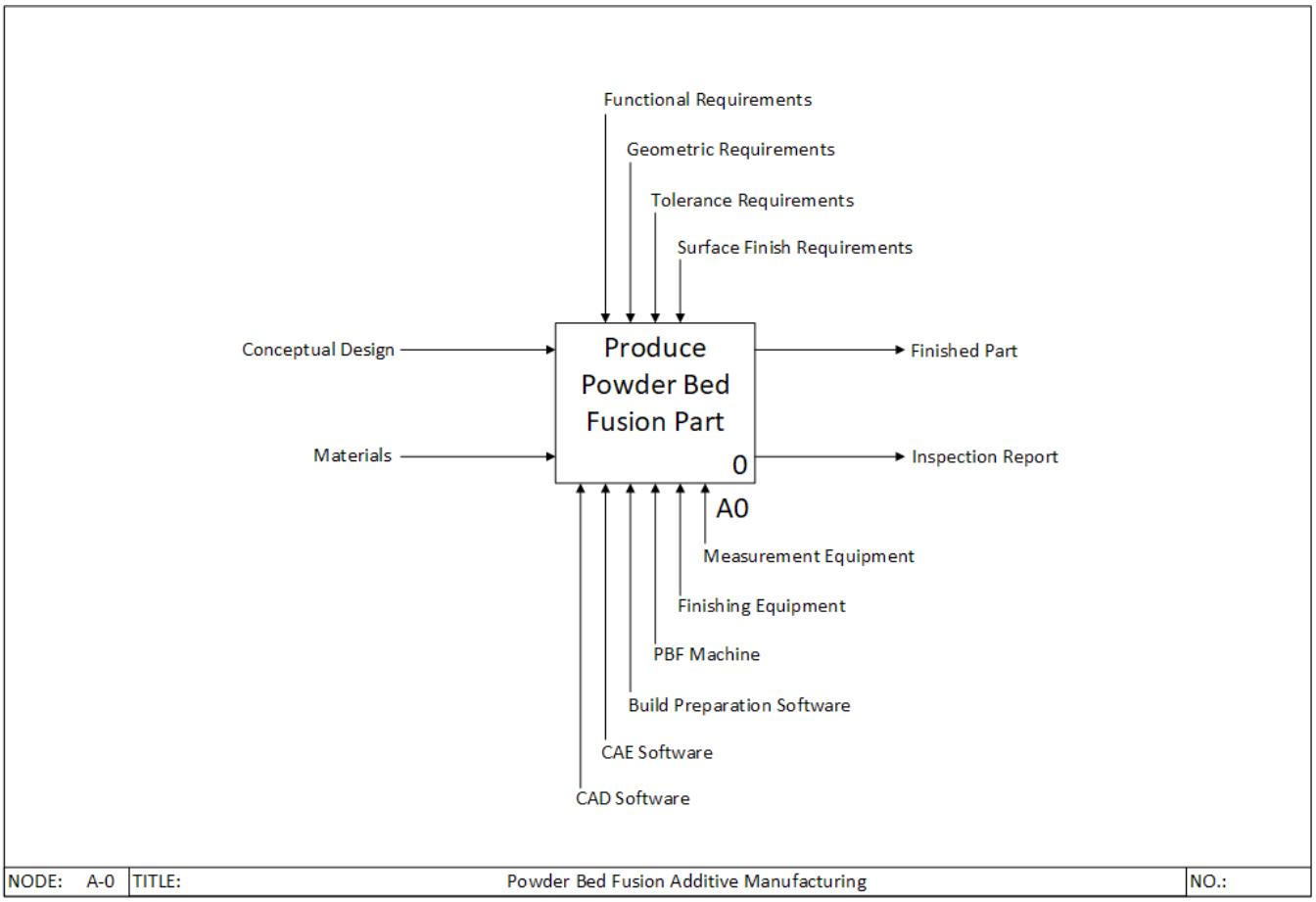

Figure 35: IDEF0 A-0 (”A Minus Zero”) Diagram

A.2 A0 Diagram

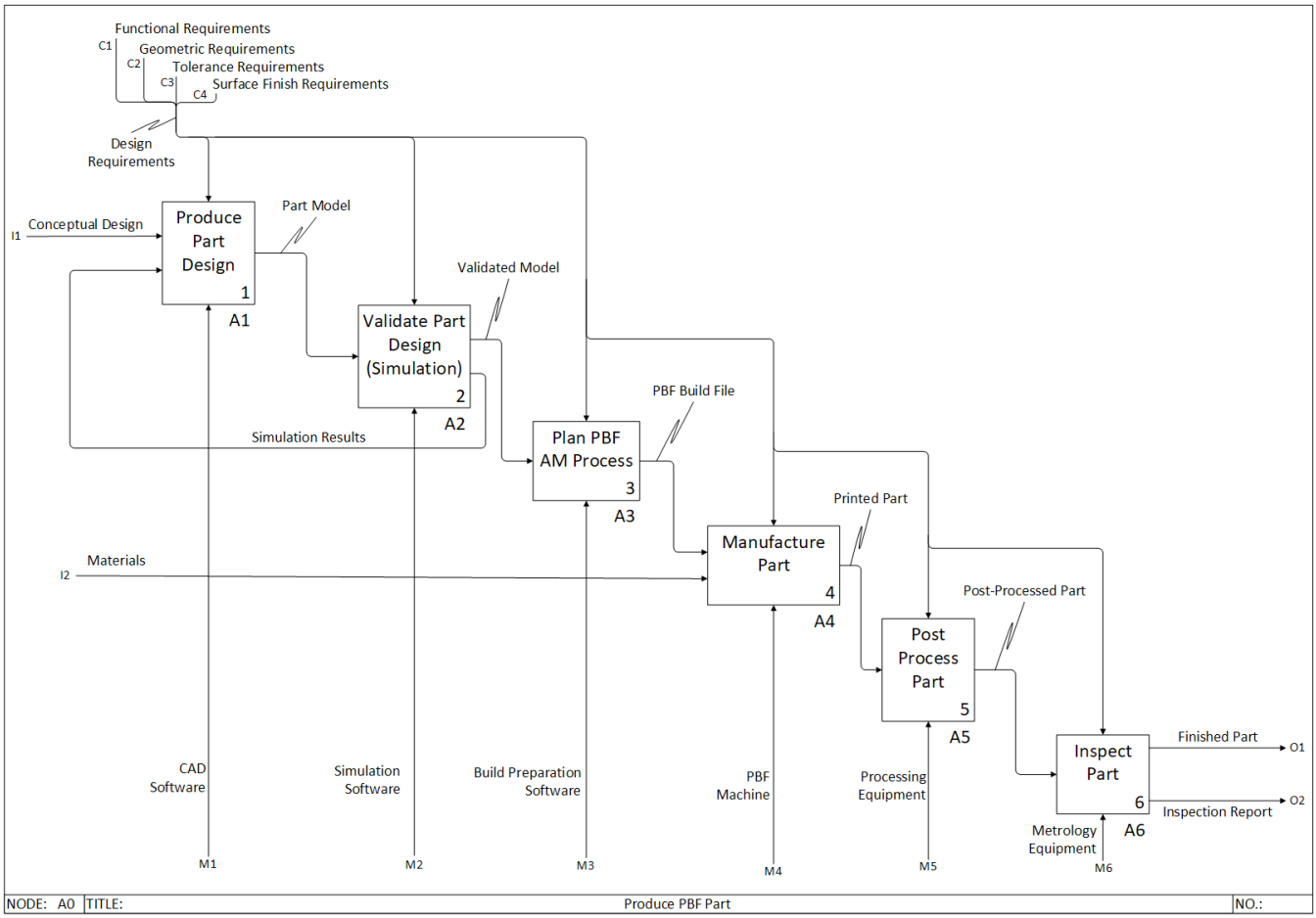

Figure 36: IDEF0 A0 Diagram

A.3 A3 Diagram

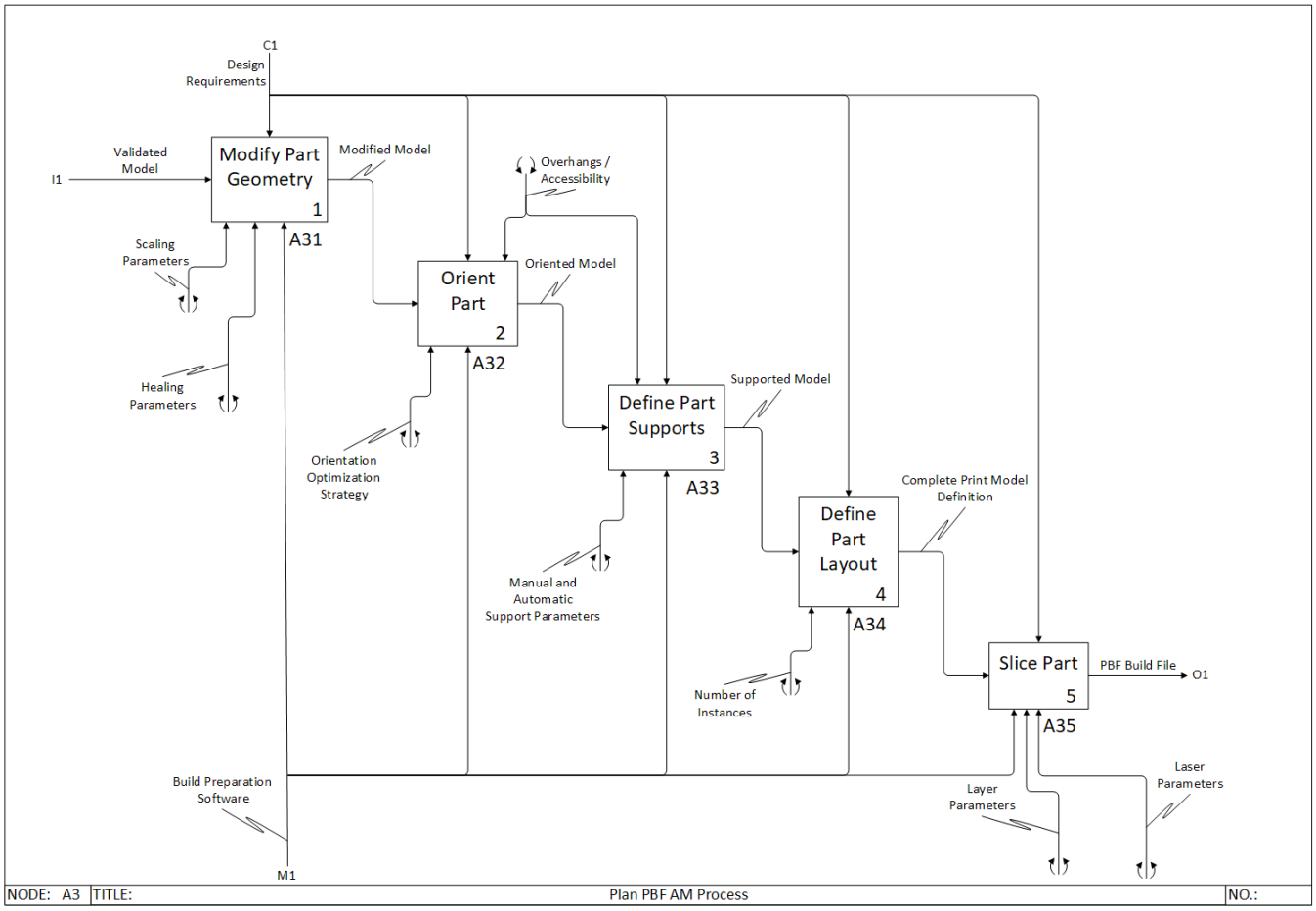

Figure 37: IDEF0 A3 Diagram

A.4 A4 Diagram

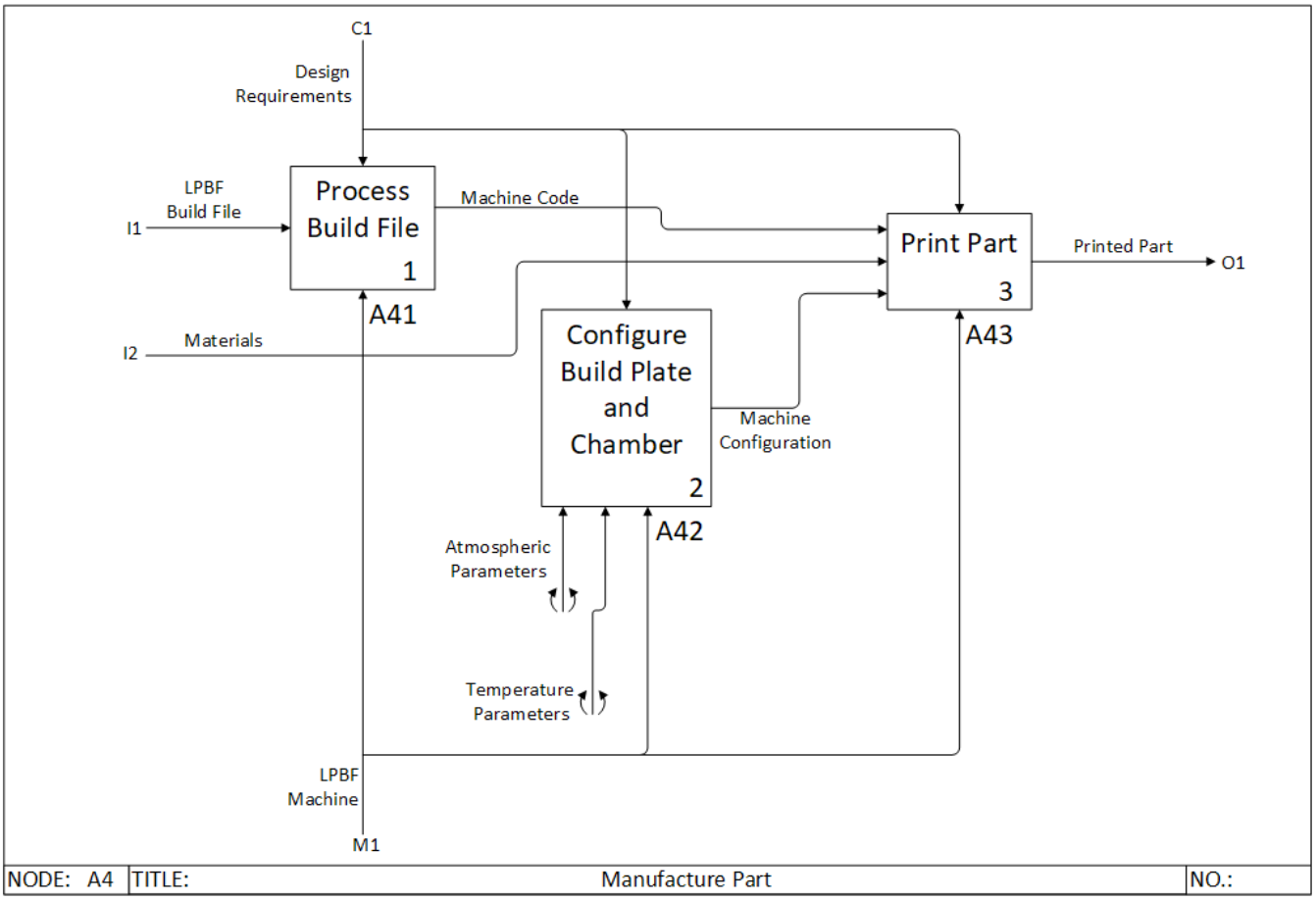

Figure 38: IDEF0 A4 Diagram

A.5 A5 Diagram

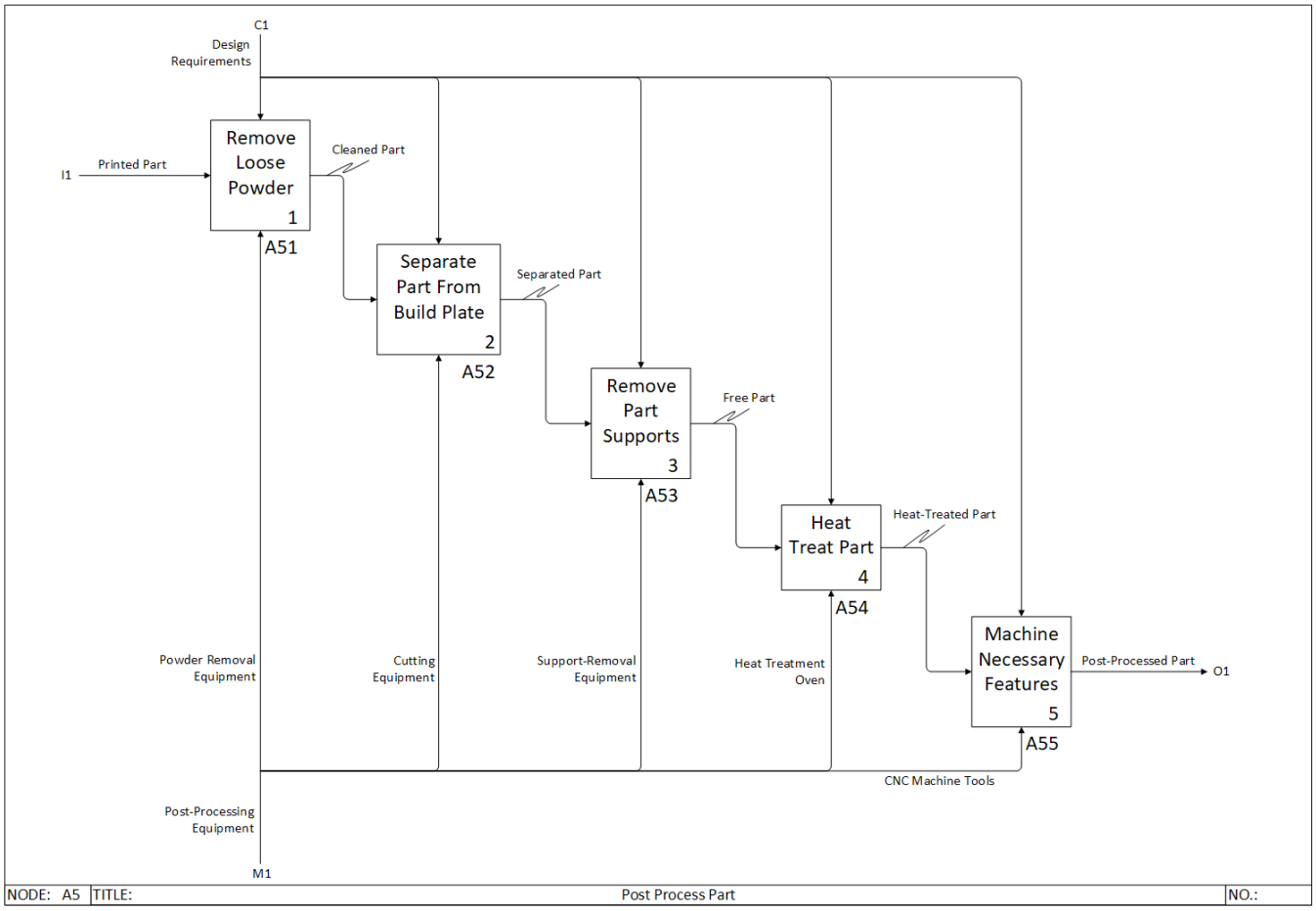

Figure 39: IDEF0 A5 Diagram

B Complete Set of Graphical Views of Information Model

B.1 AMDocumentType

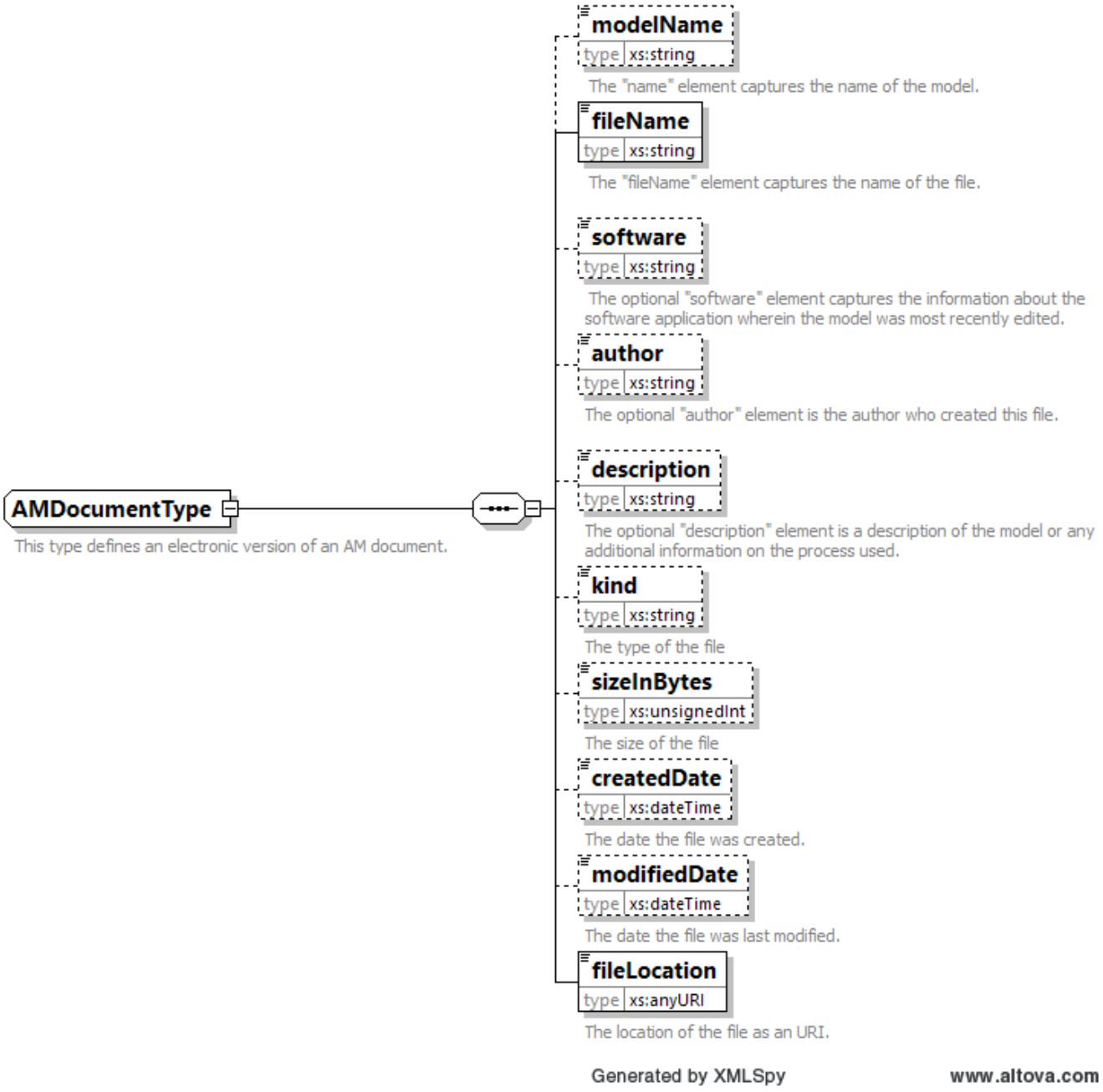

Figure 40: AMDocumentType element

B.2 anchorPartLocation

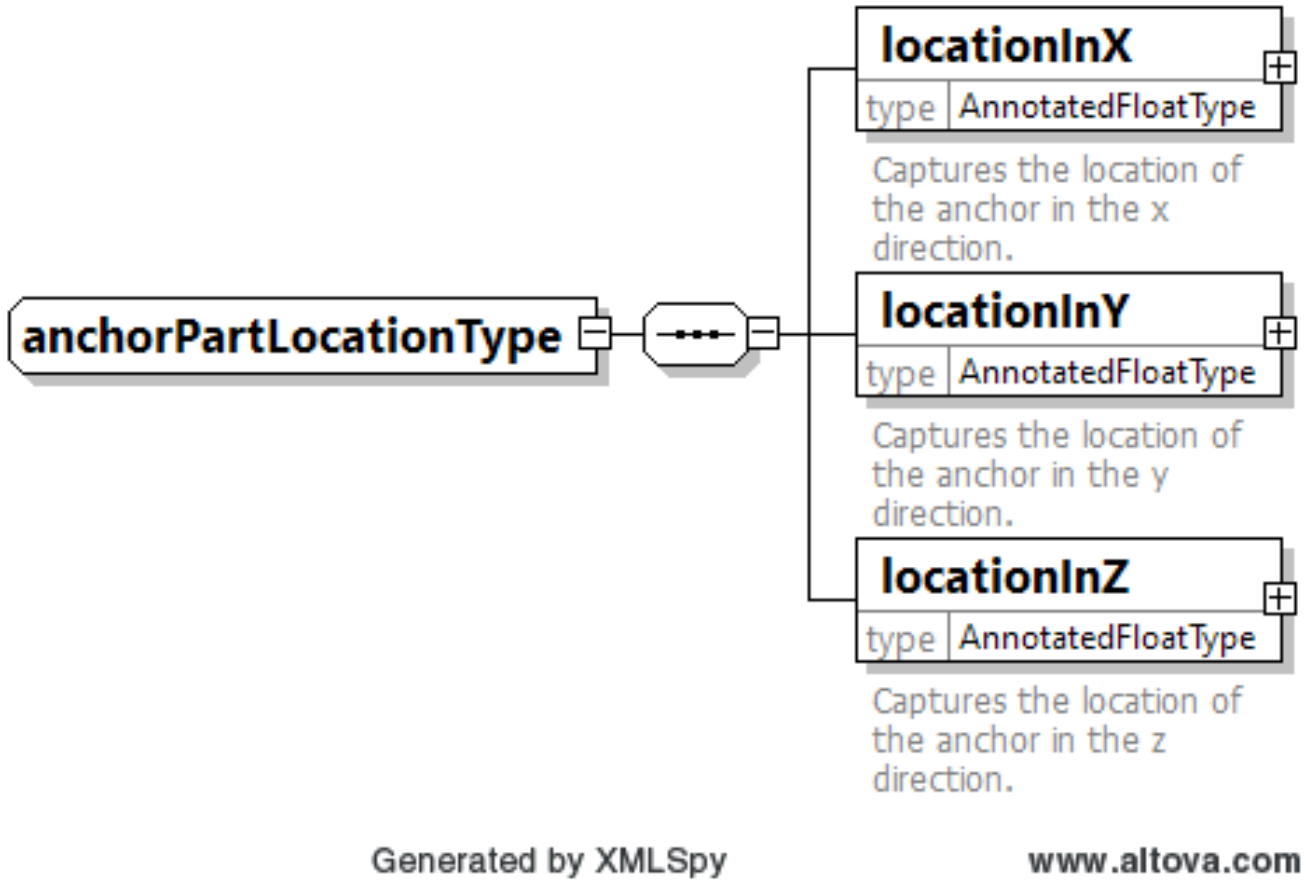

Figure 41: anchorPartLocation element

B.3 AnnotatedFloatType

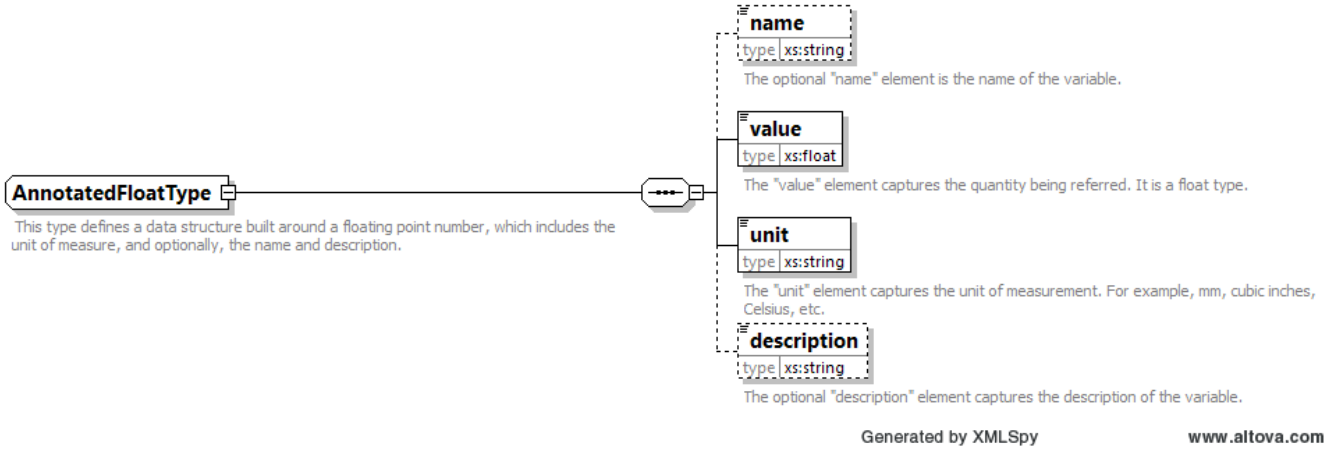

Figure 42: AnnotatedFloatType element

B.4 boundingBox

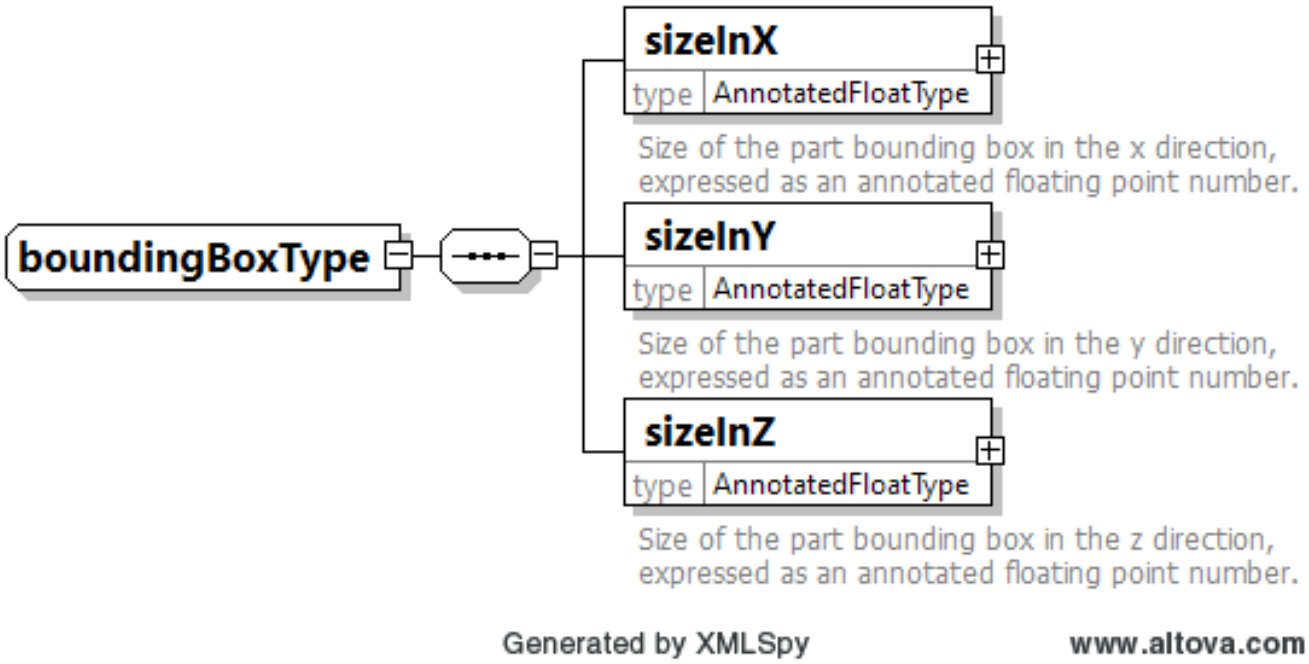

Figure 43: boundingBox element

B.5 buildProperties

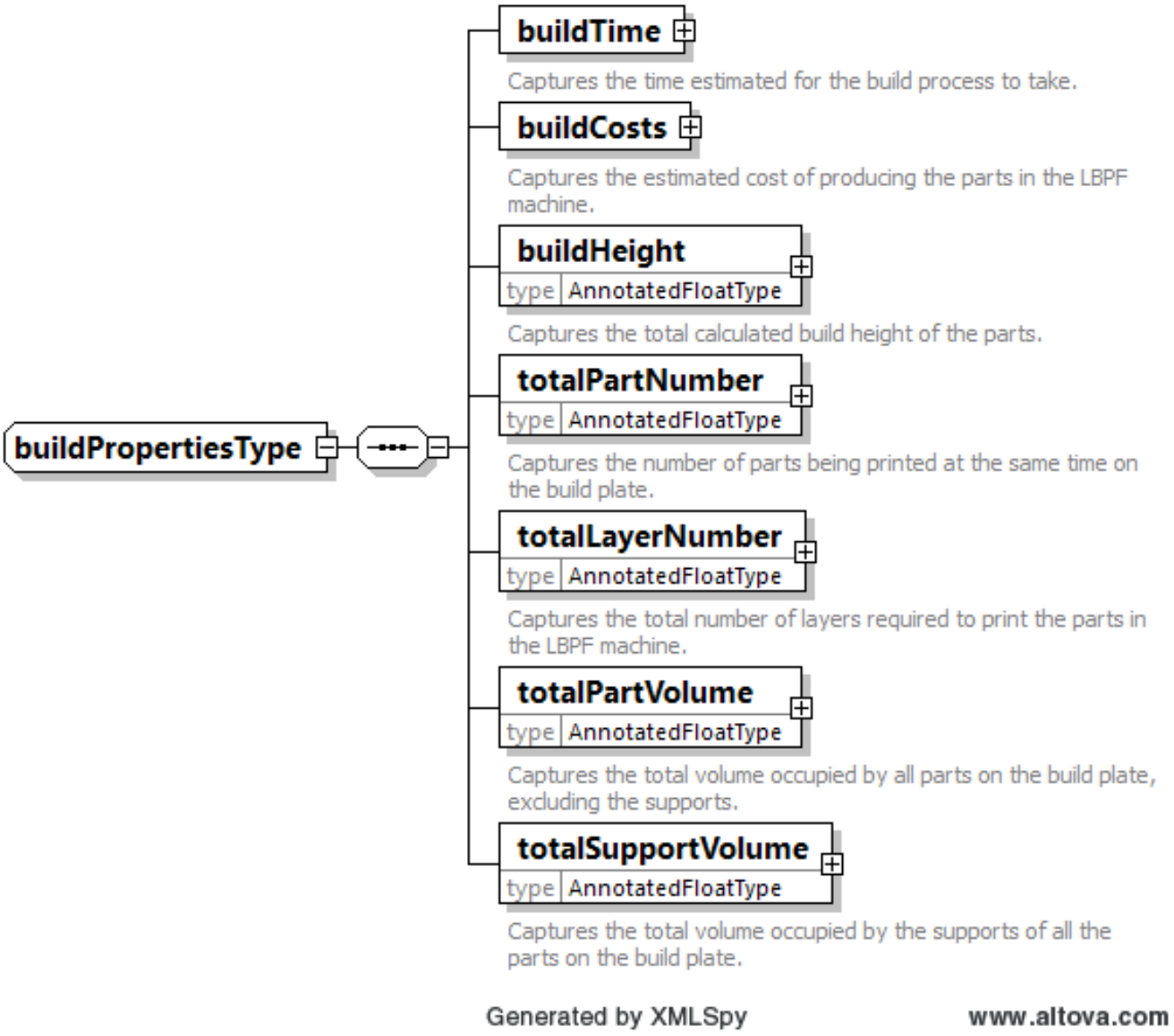

Figure 44: buildProperties element

B.6 buildSettings

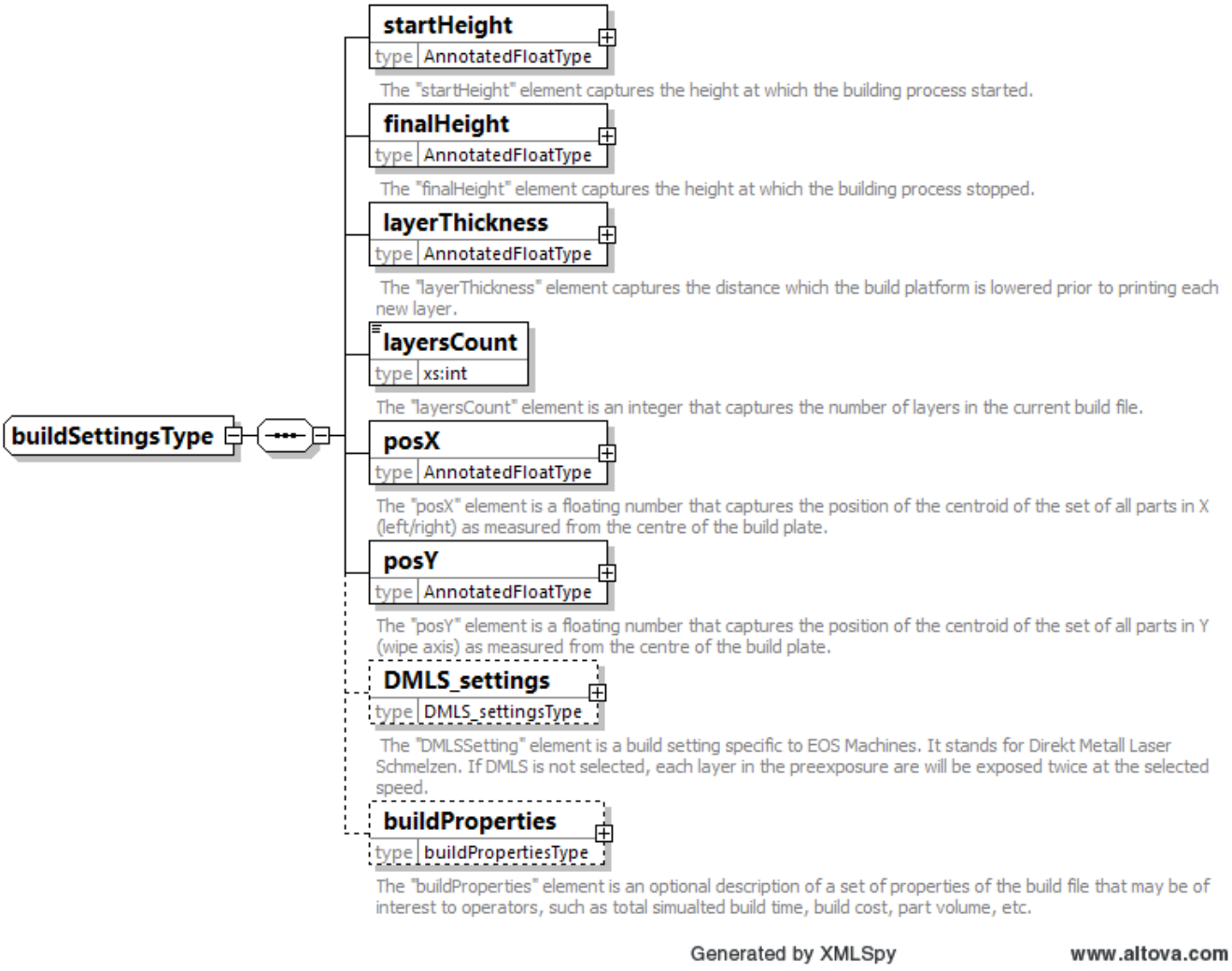

Figure 45: buildSettings element

B.7 buildSupportsData

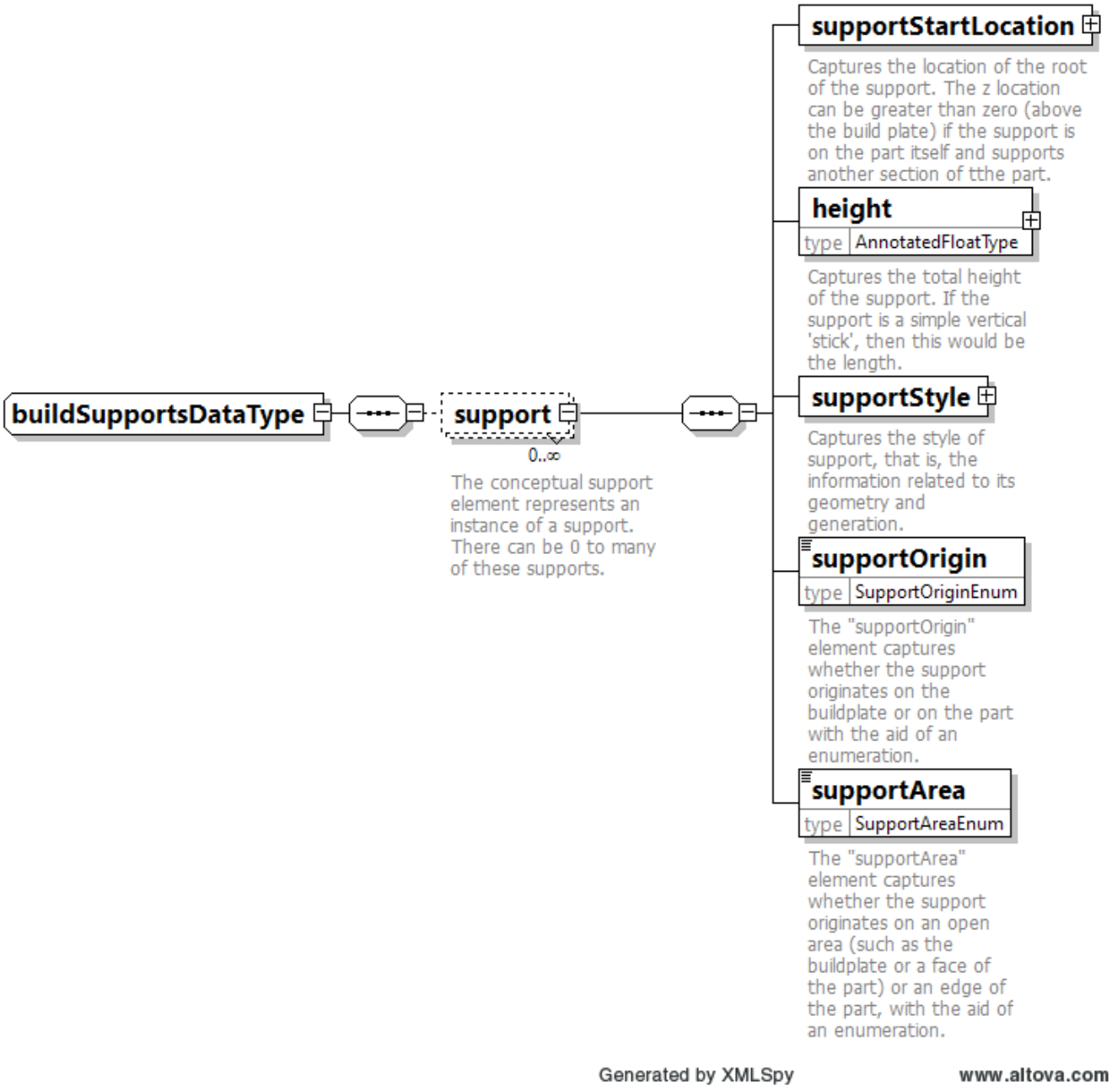

Figure 46: buildSupportsData element

B.8 buildVolume

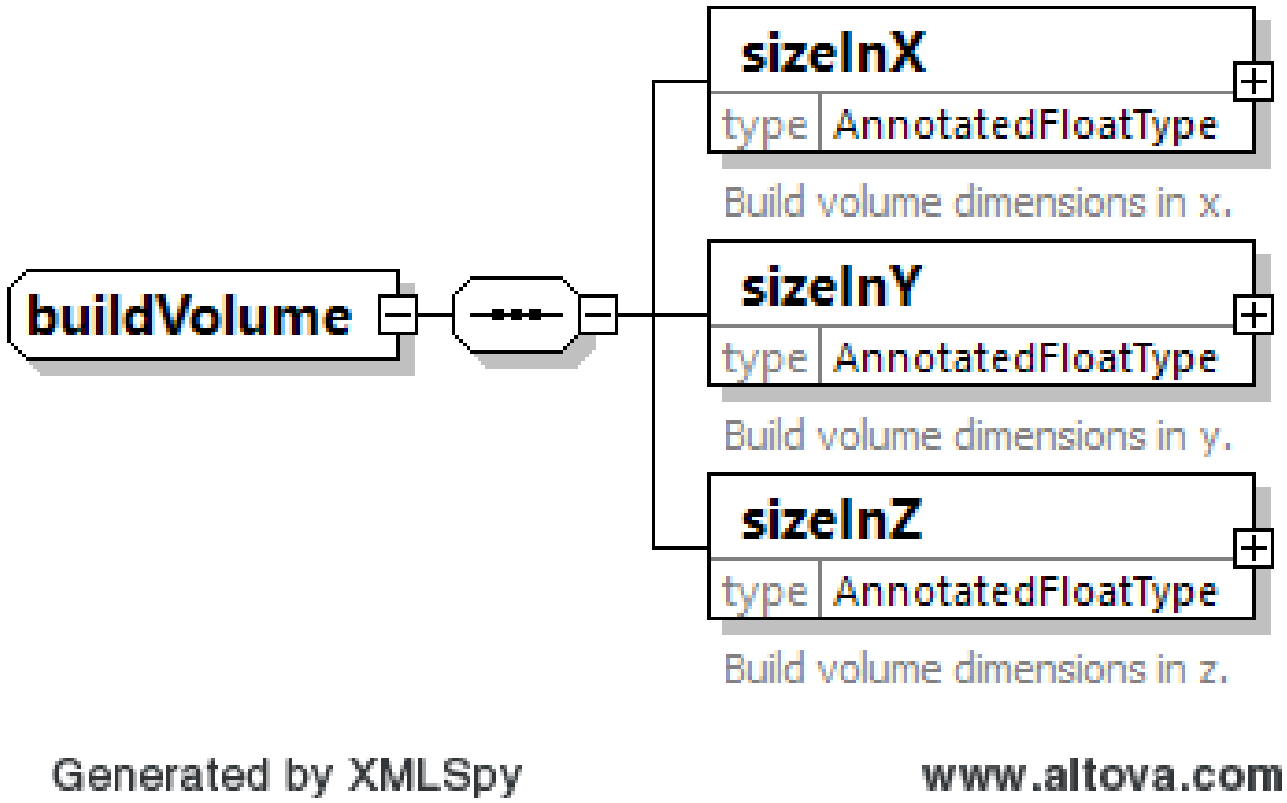

Figure 47: buildVolume element

B.9 chessCategorySettings

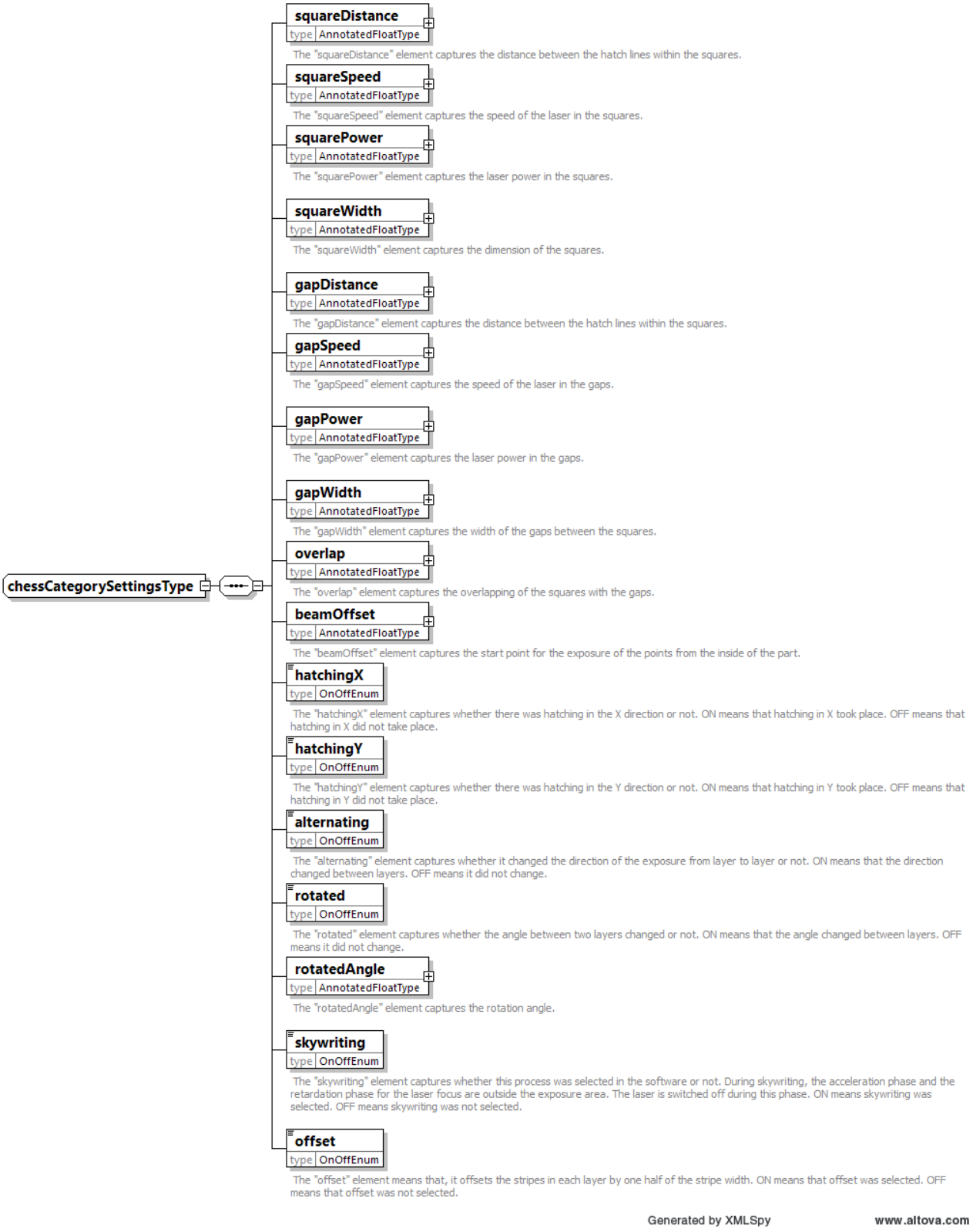

Figure 48: chessCategorySettings element

B.10 clusterSupportSettings

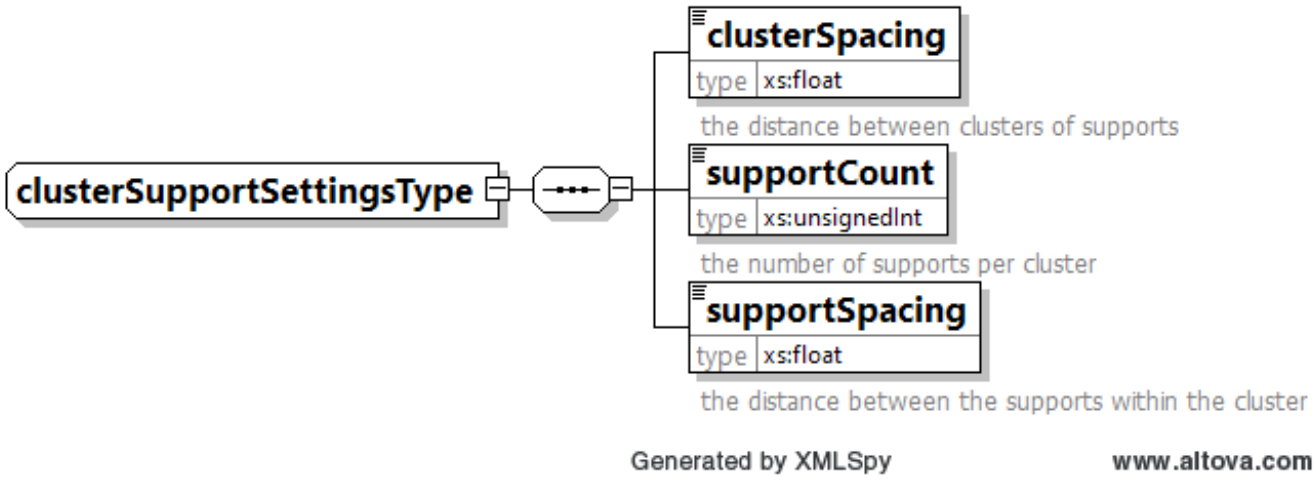

Figure 49: clusterSupportSettings element

B.11 contourSettings

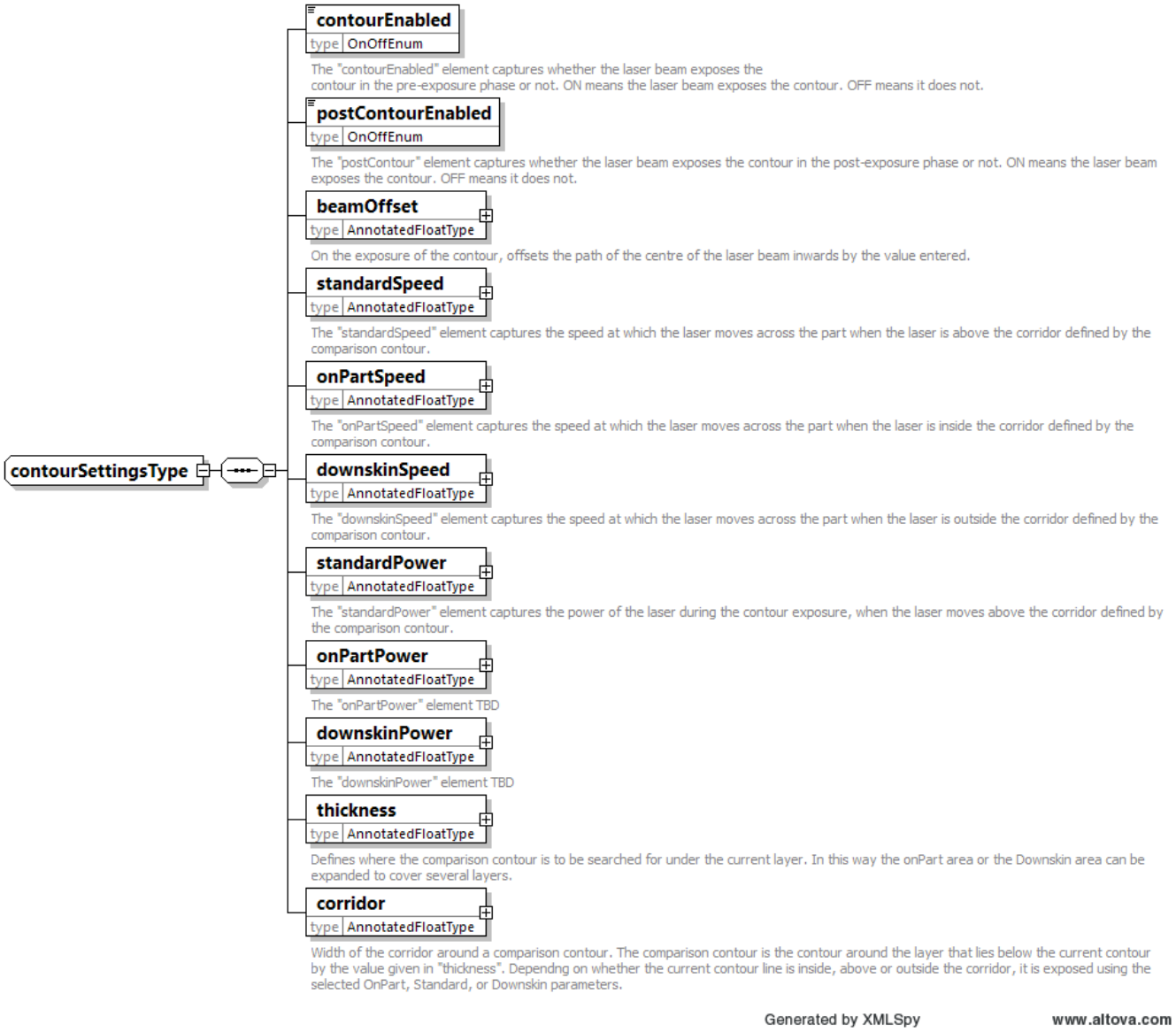

Figure 50: contourSettings element

B.12 controlParameters

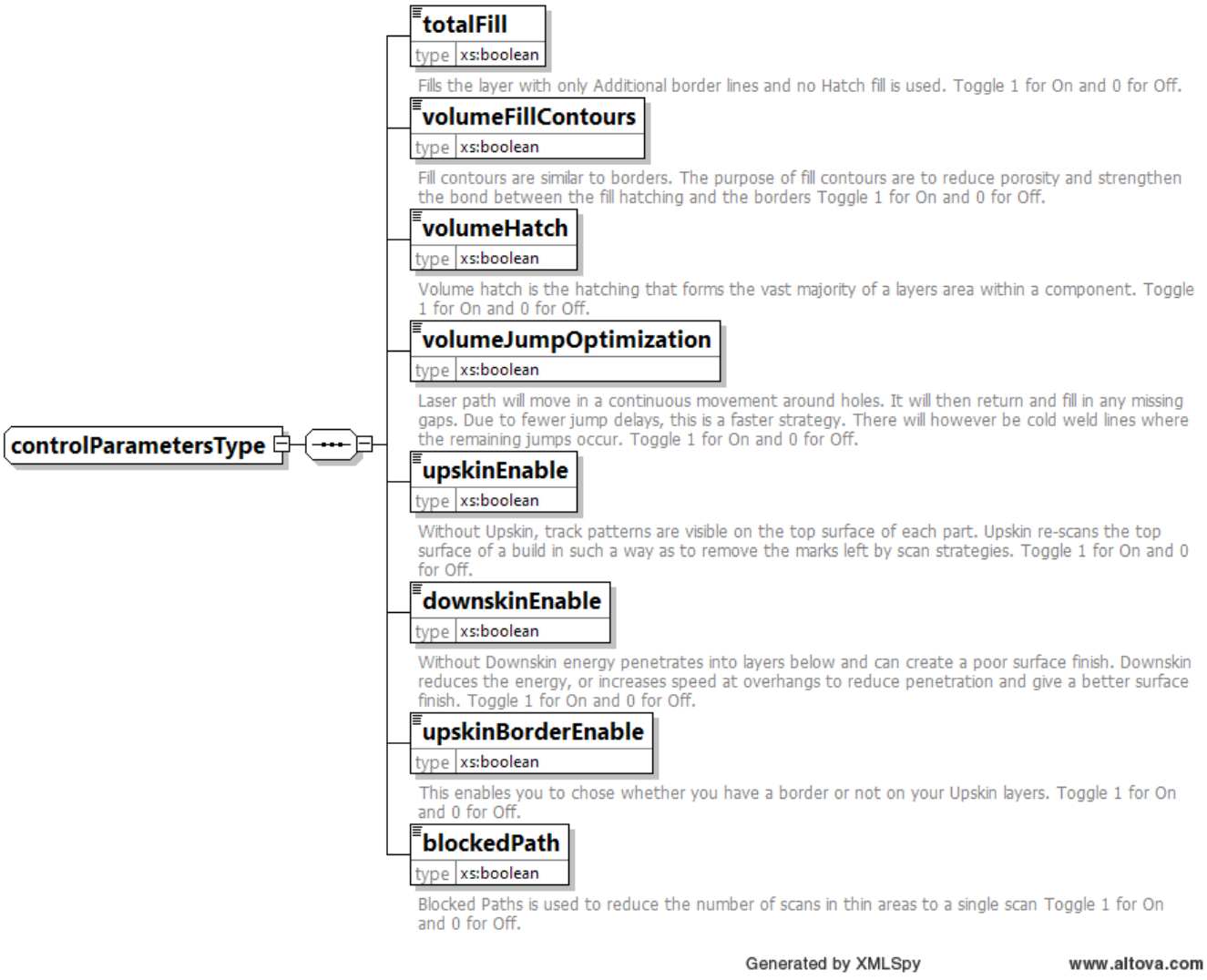

Figure 51: controlParameters element

B.13 DMLS_settings

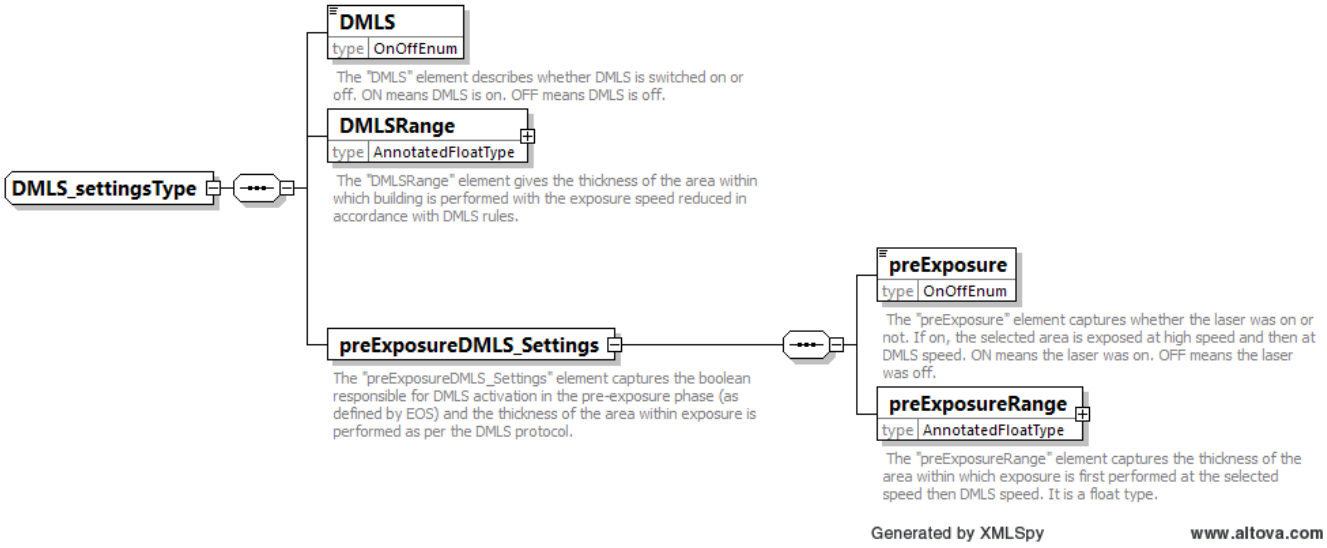

Figure 52: DMLS_settings element

B.14 edgeSettings

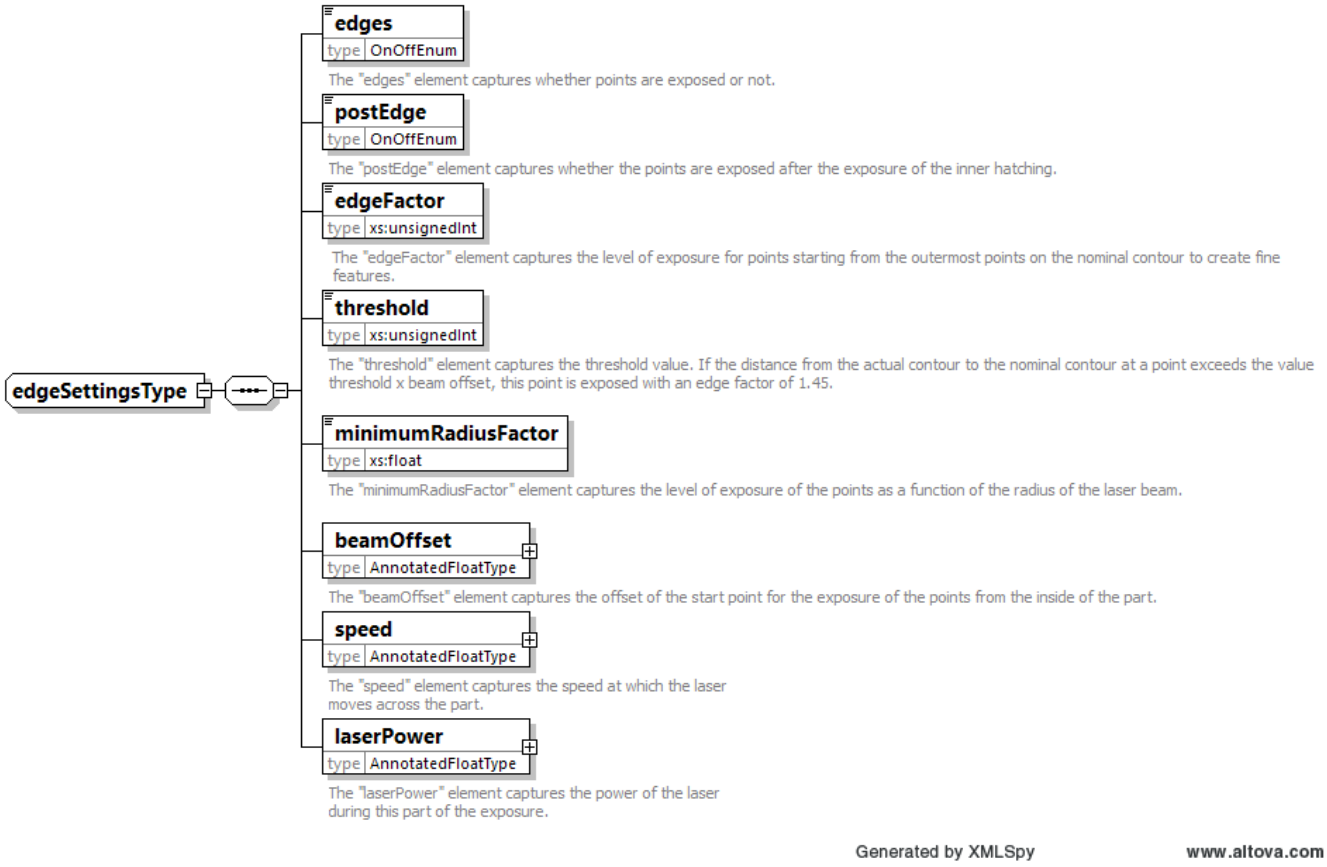

Figure 53: edgeSettings element

B.15 EOS_advancedSettings

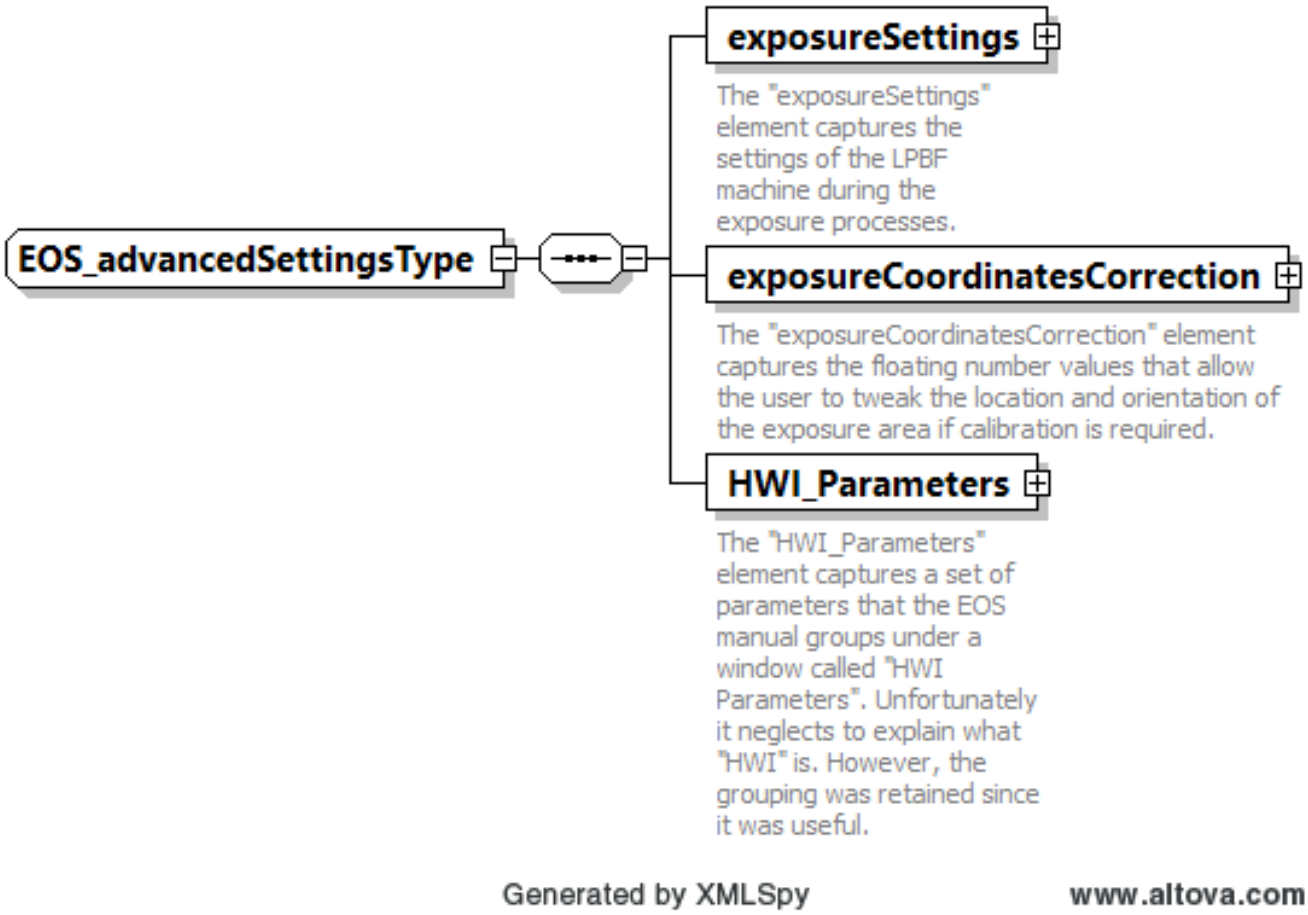

Figure 54: EOS_advancedSettings element

B.16 exportData

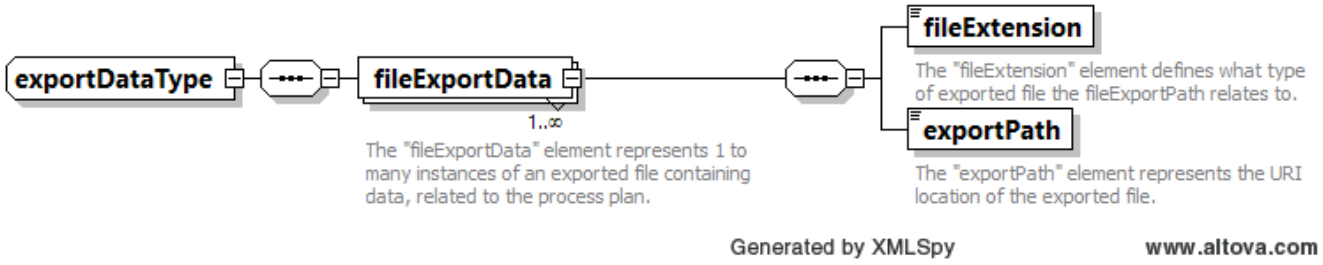

Figure 55: exportData element

B.17 exposureSelectionType

Figure 56: exposureSelectionType element

B.18 exposureSettings

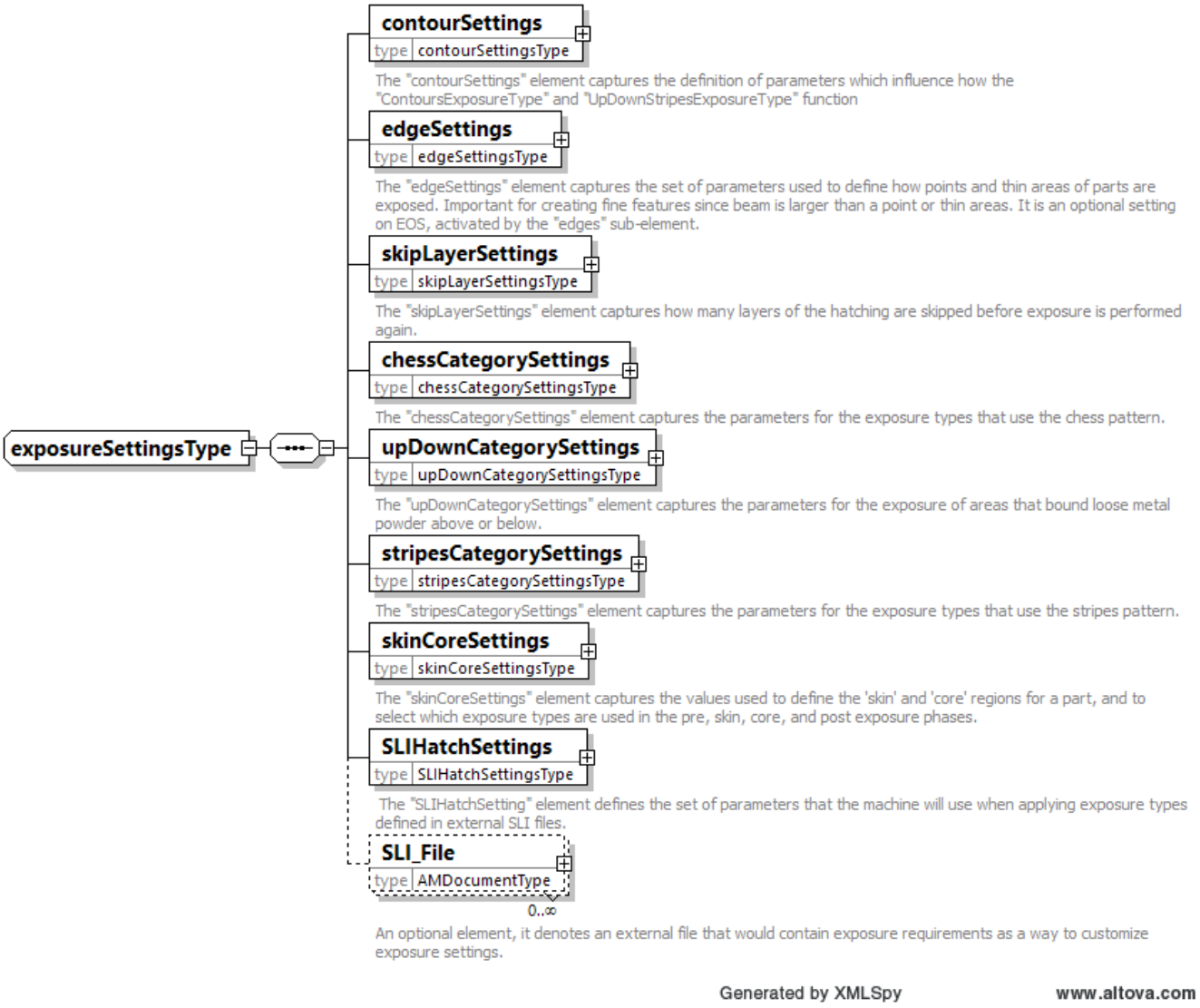

Figure 57: exposureSettings element

B.19 exposureTypeSelections

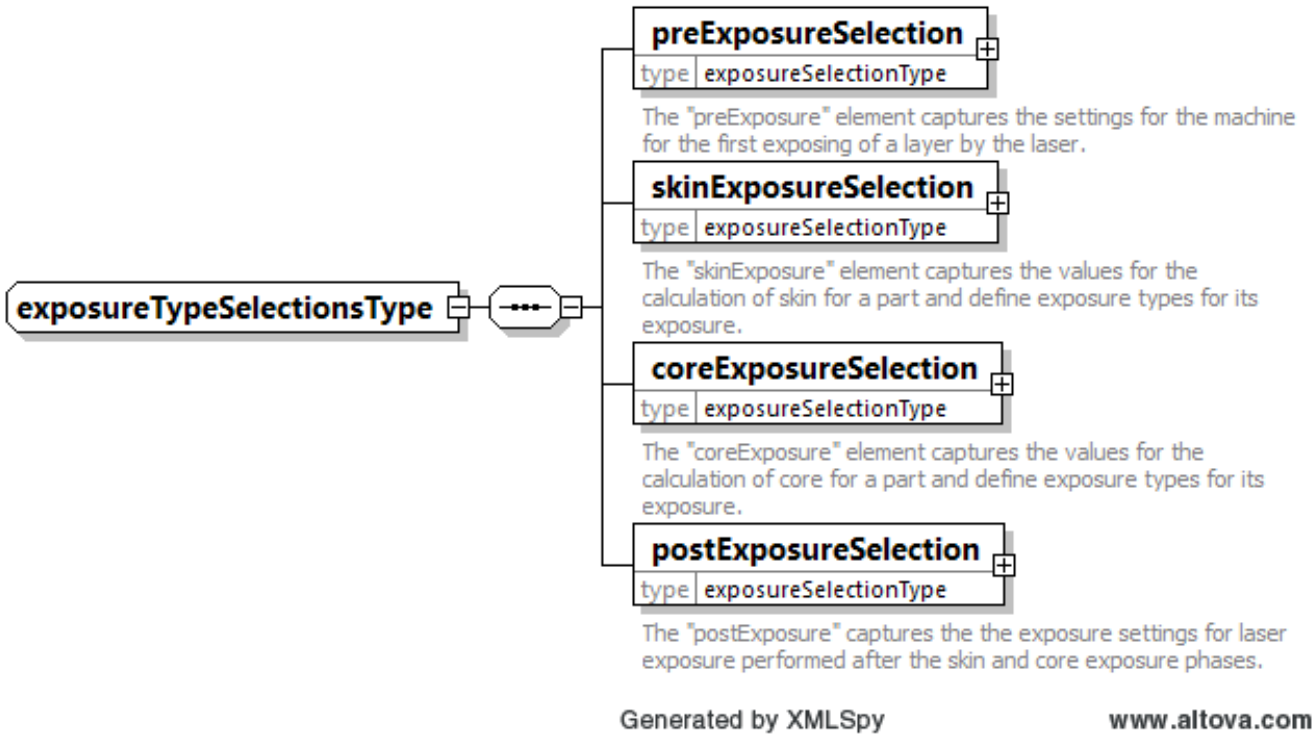

Figure 58: exposureTypeSelections element (Not to be confused with exposureSelectionType)

B.20 generalDownskinParameters

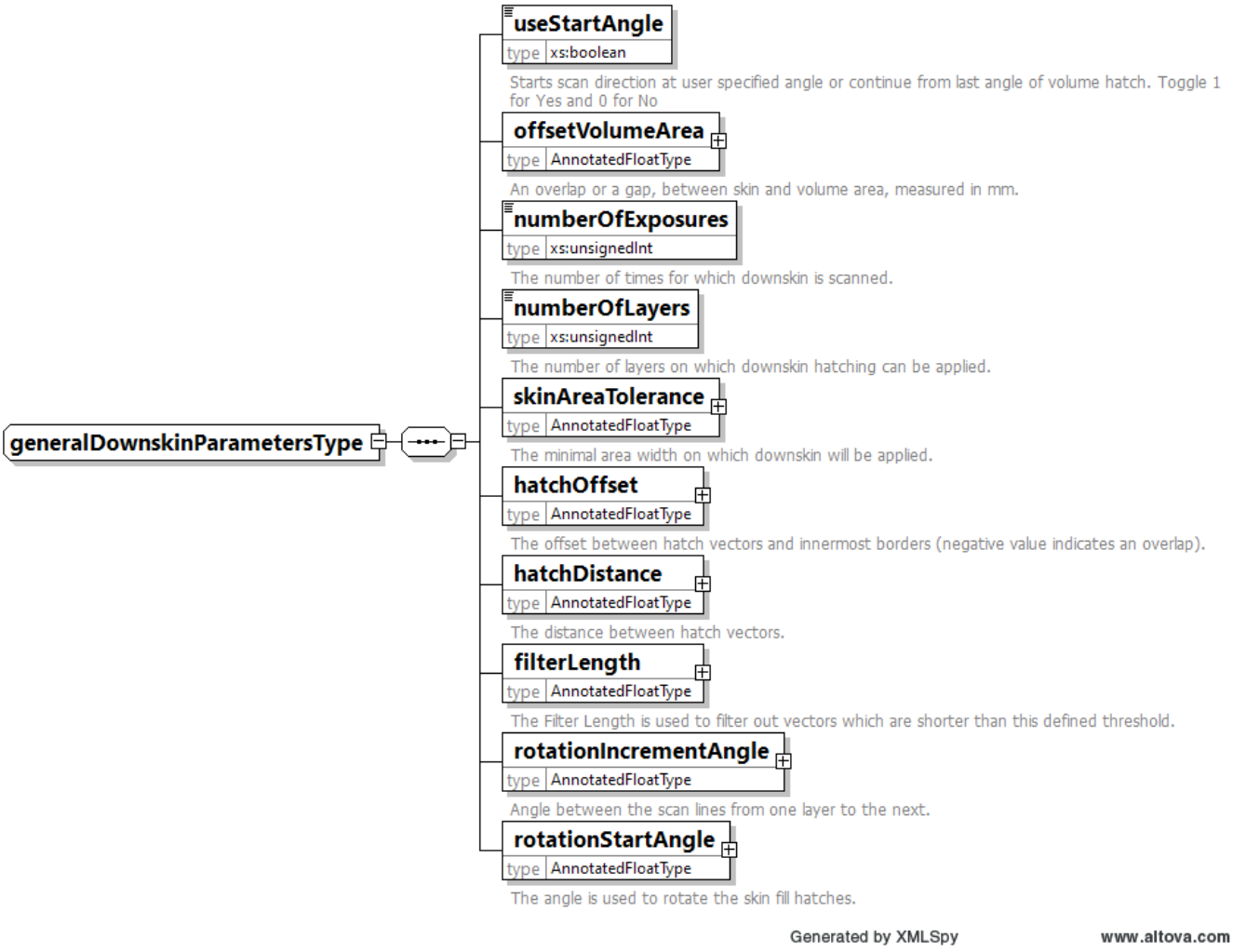

Figure 59: generalDownskinParameters element

B.21 generalUpskinParameters

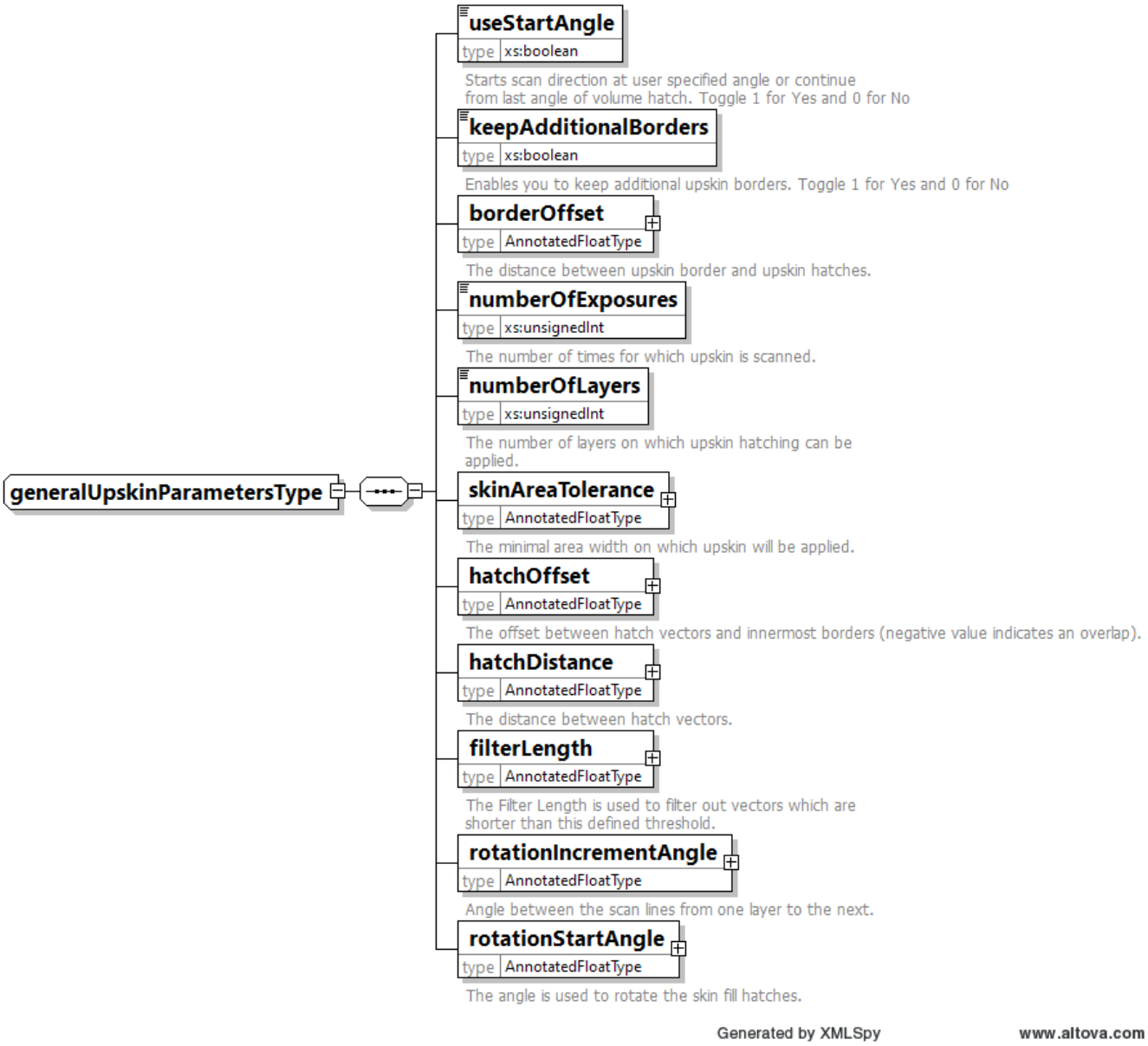

Figure 60: generalUpskinParameters element

B.22 generalVolumeParameters

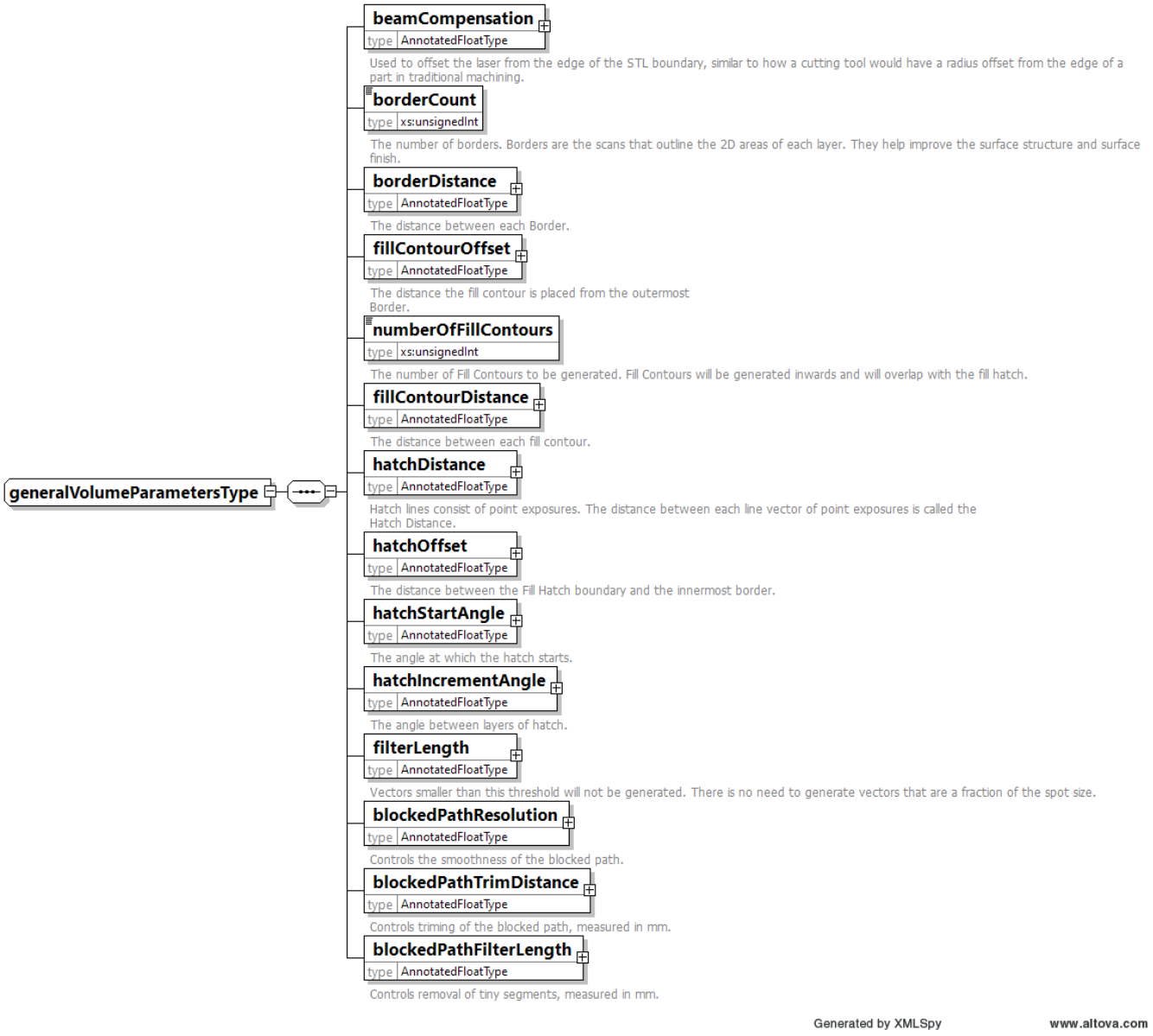

Figure 61: generalVolumeParameters element

B.23 healingParameters

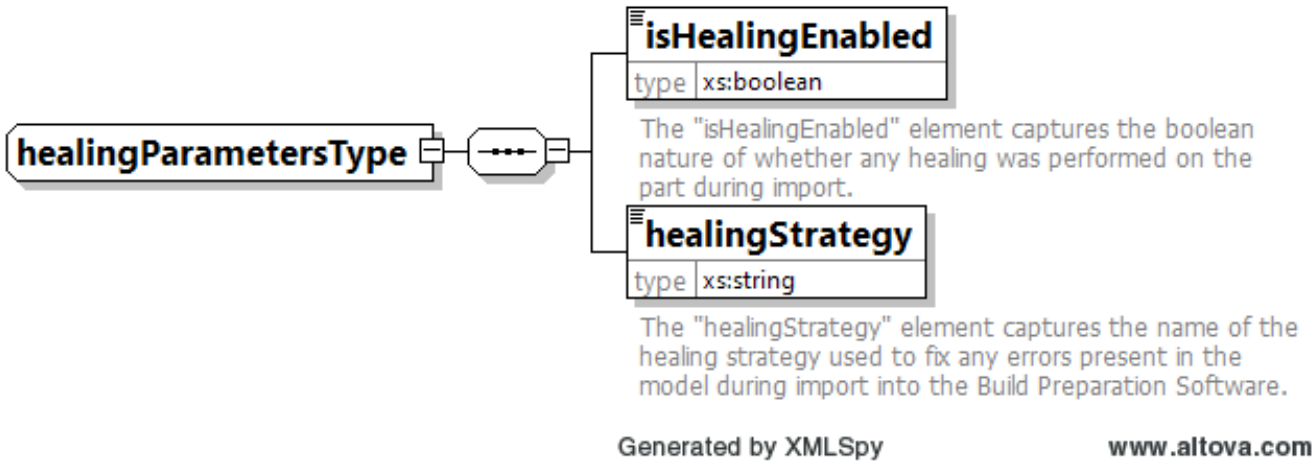

Figure 62: healingParameters element

B.24 instancingSettings

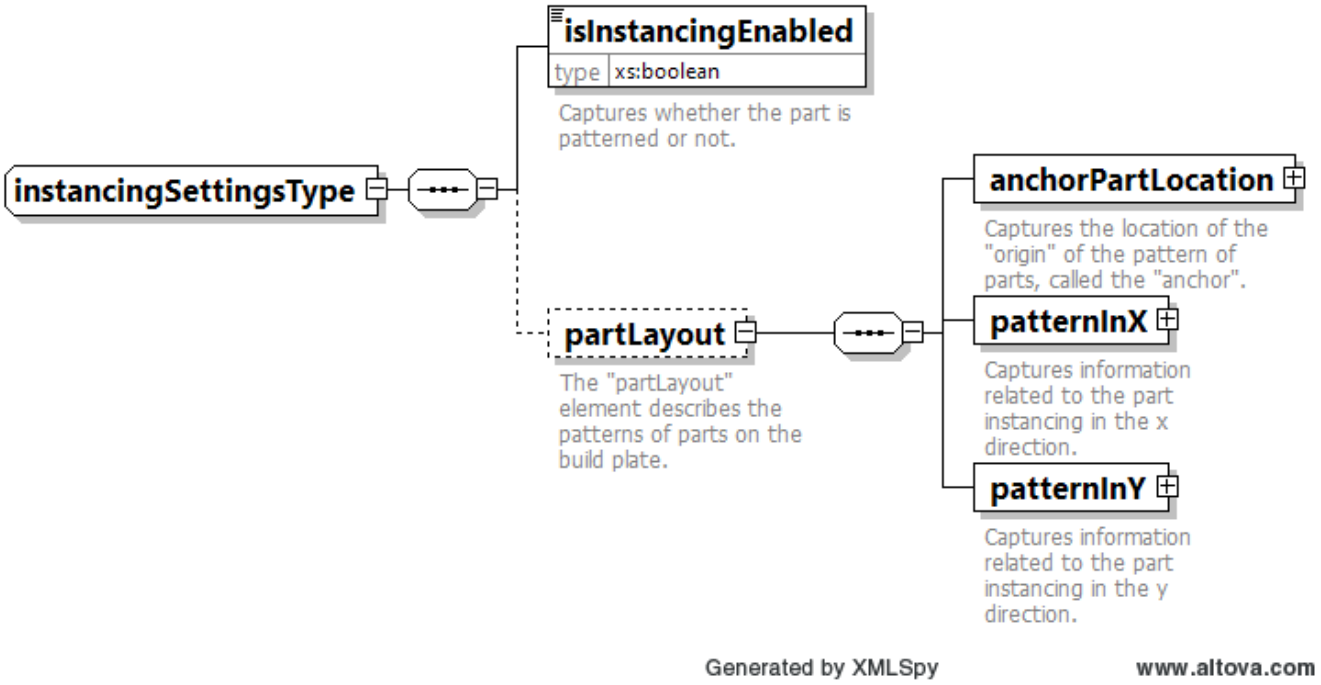

Figure 63: instancingSettings element

B.25 machineSpecsData

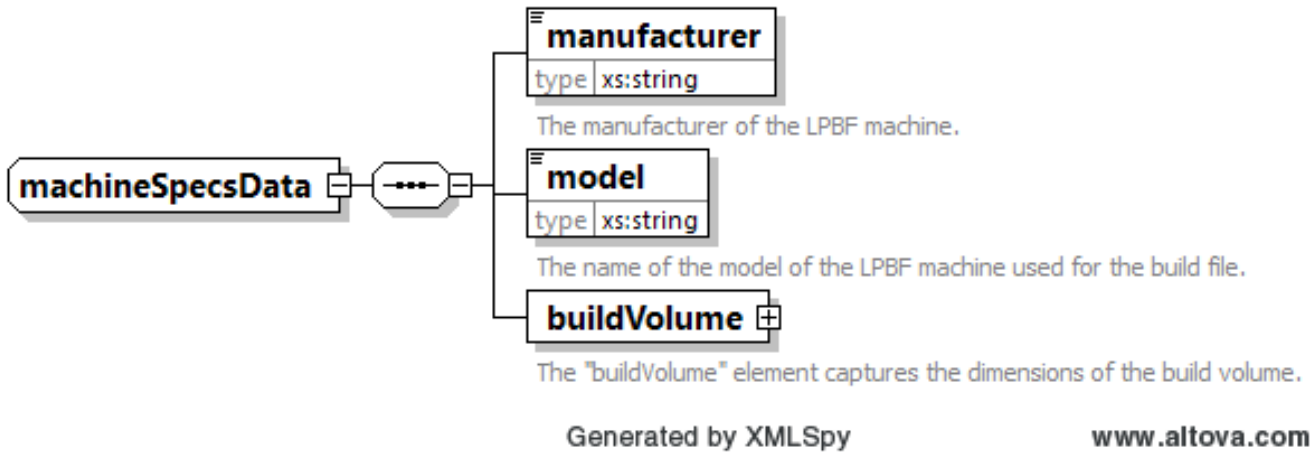

Figure 64: machineSpecsData element

B.26 materialSpecificSettings

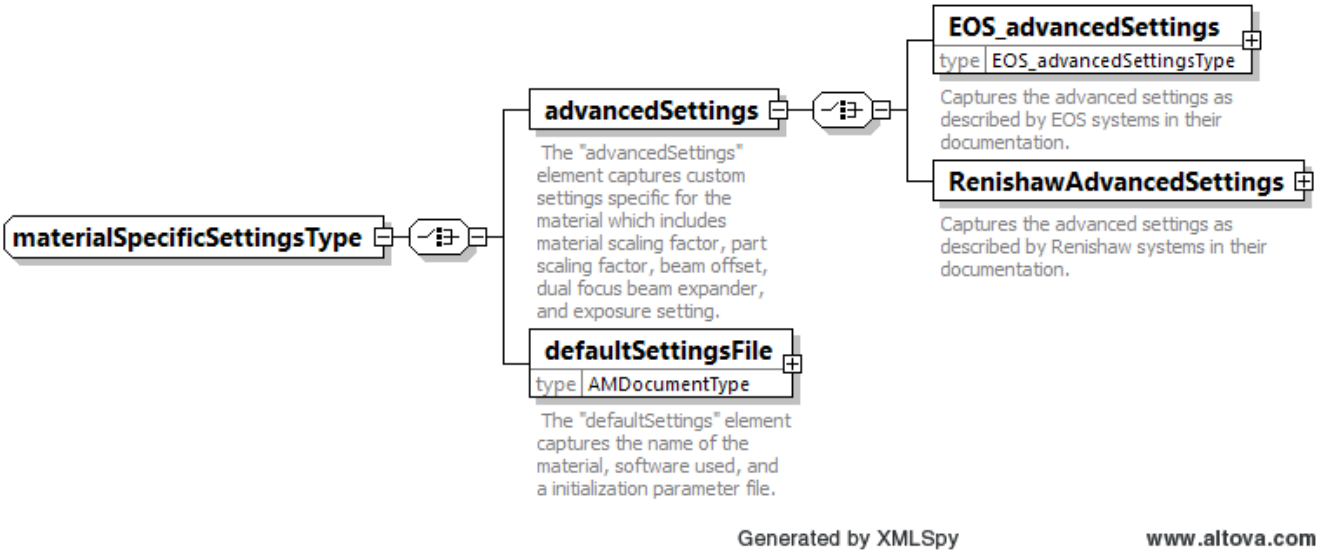

Figure 65: materialSpecificSettings element

B.27 miscellaneousSettings

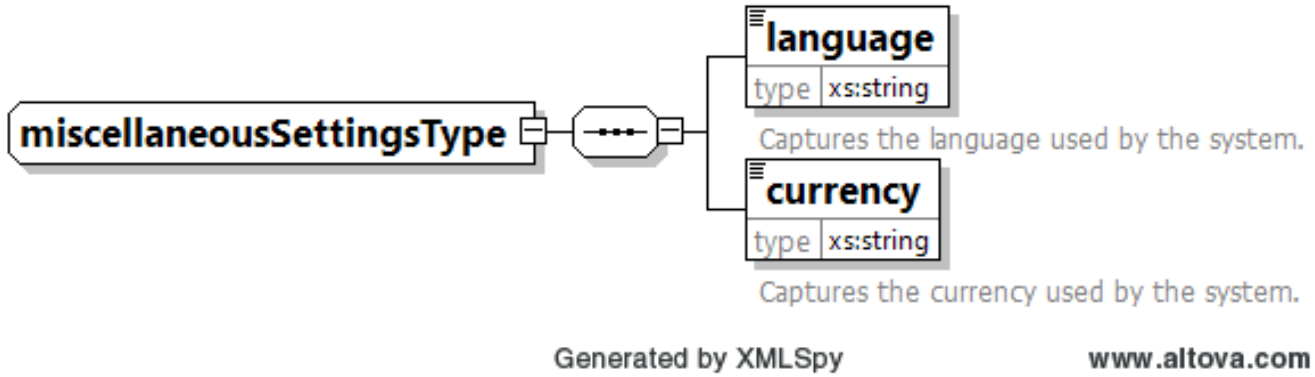

Figure 66: miscellaneousSettings element

B.28 orderParameters

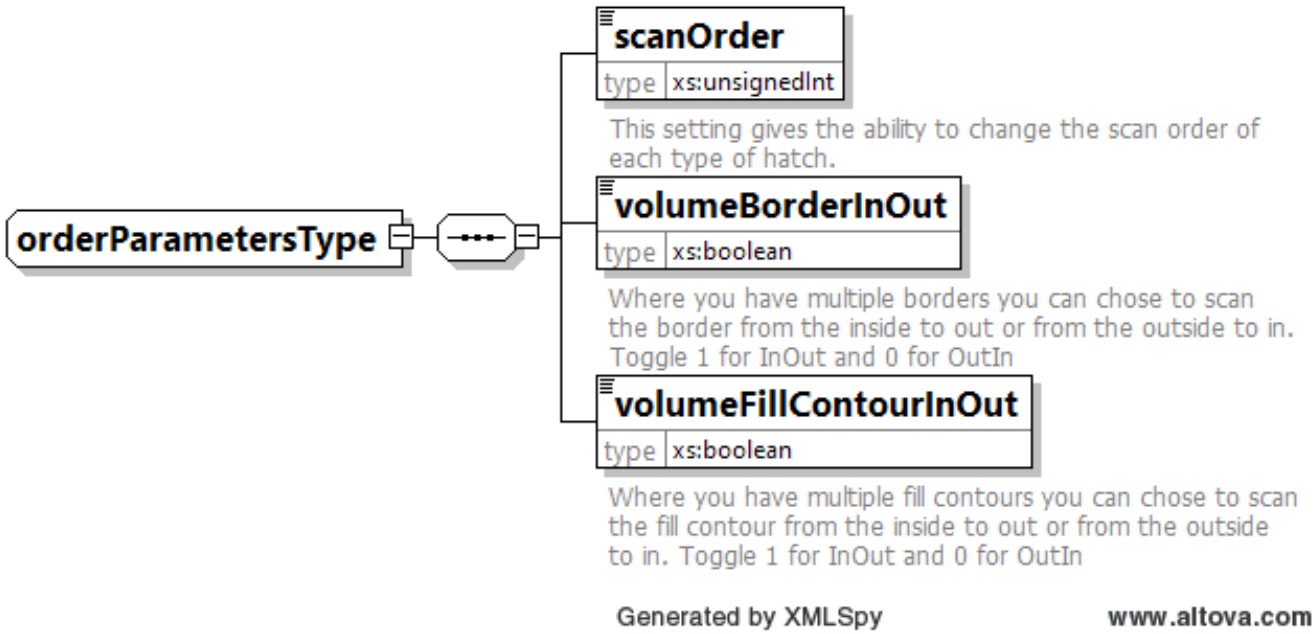

Figure 67: orderParameters element

B.29 partData

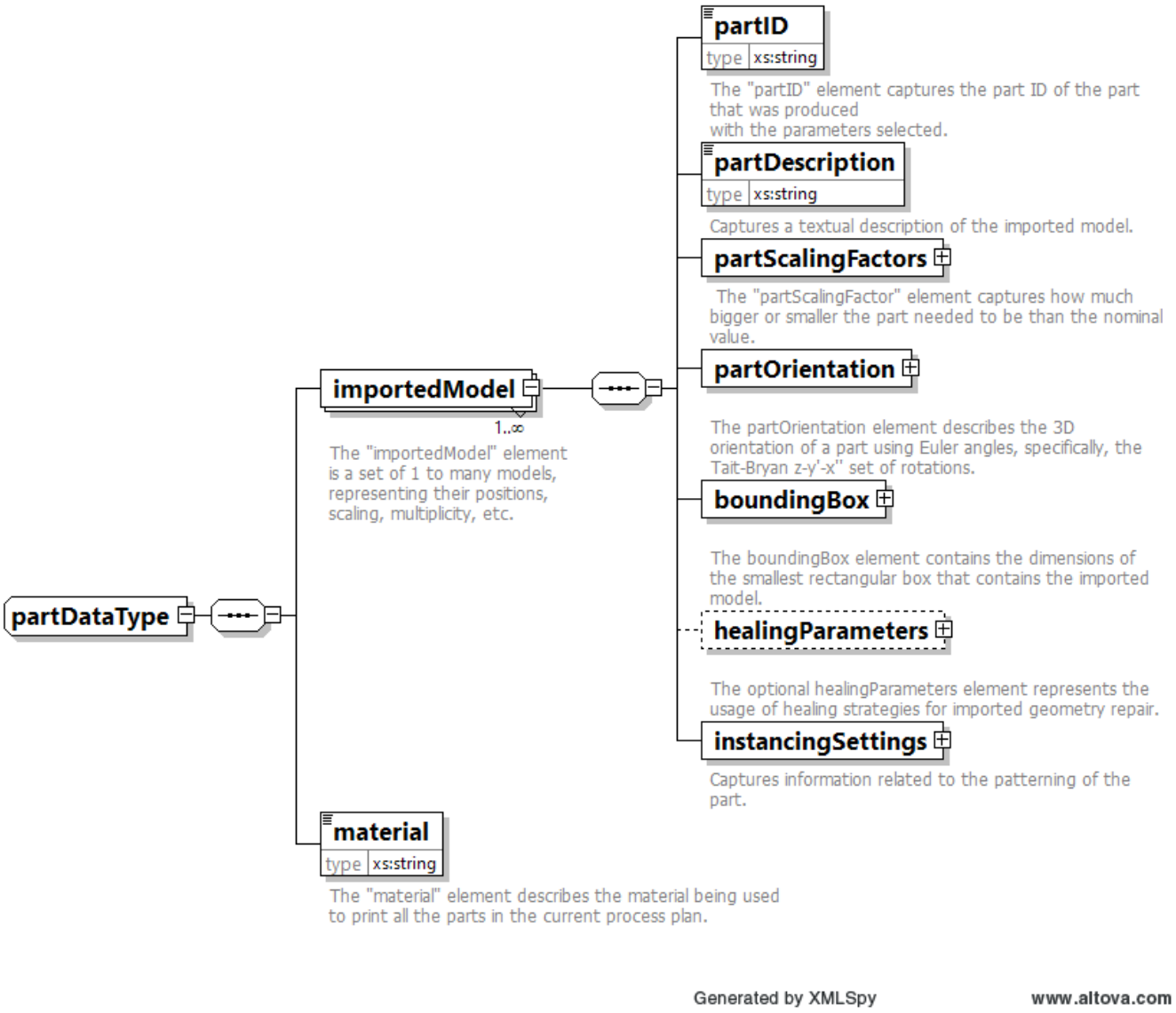

Figure 68: partData element

B.30 partOrientation

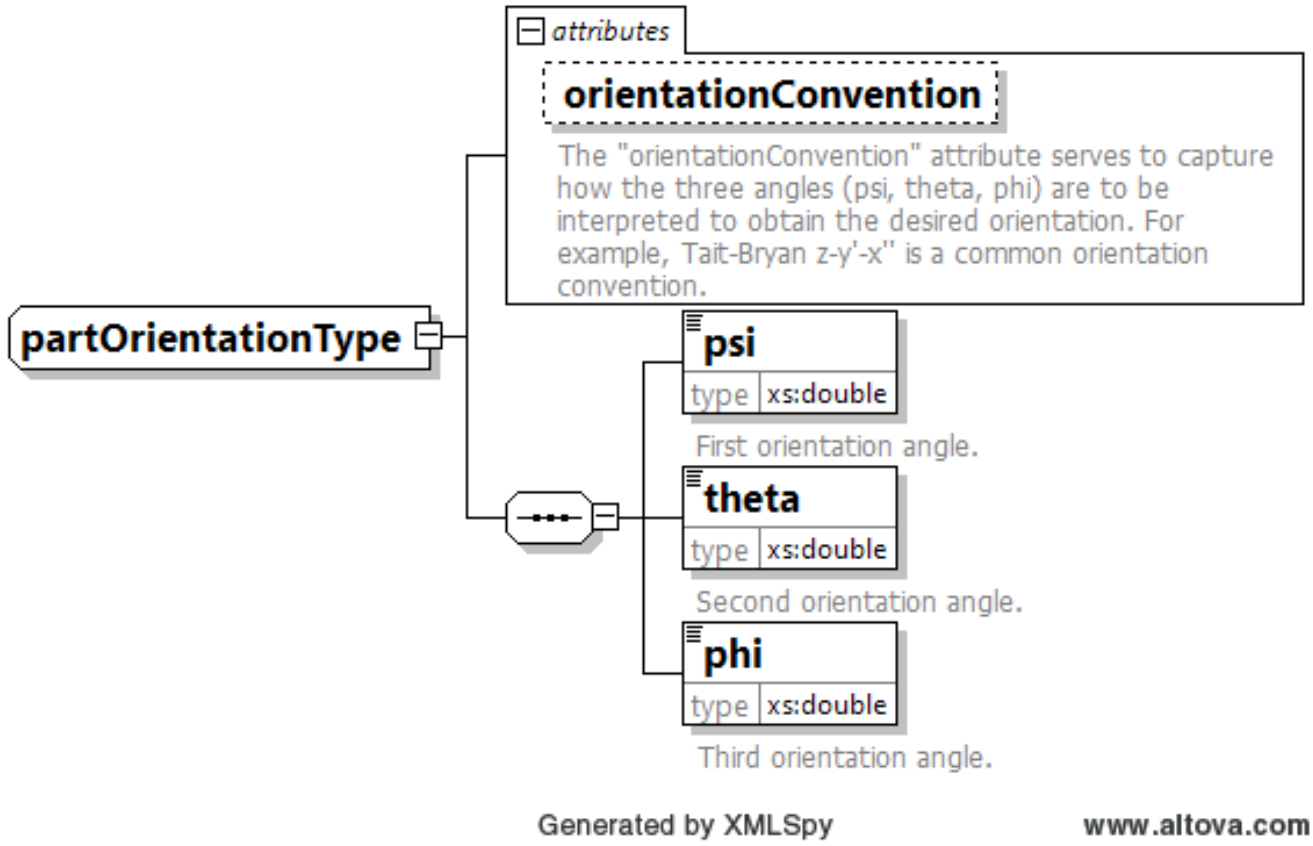

Figure 69: partOrientation element

B.31 patternInX

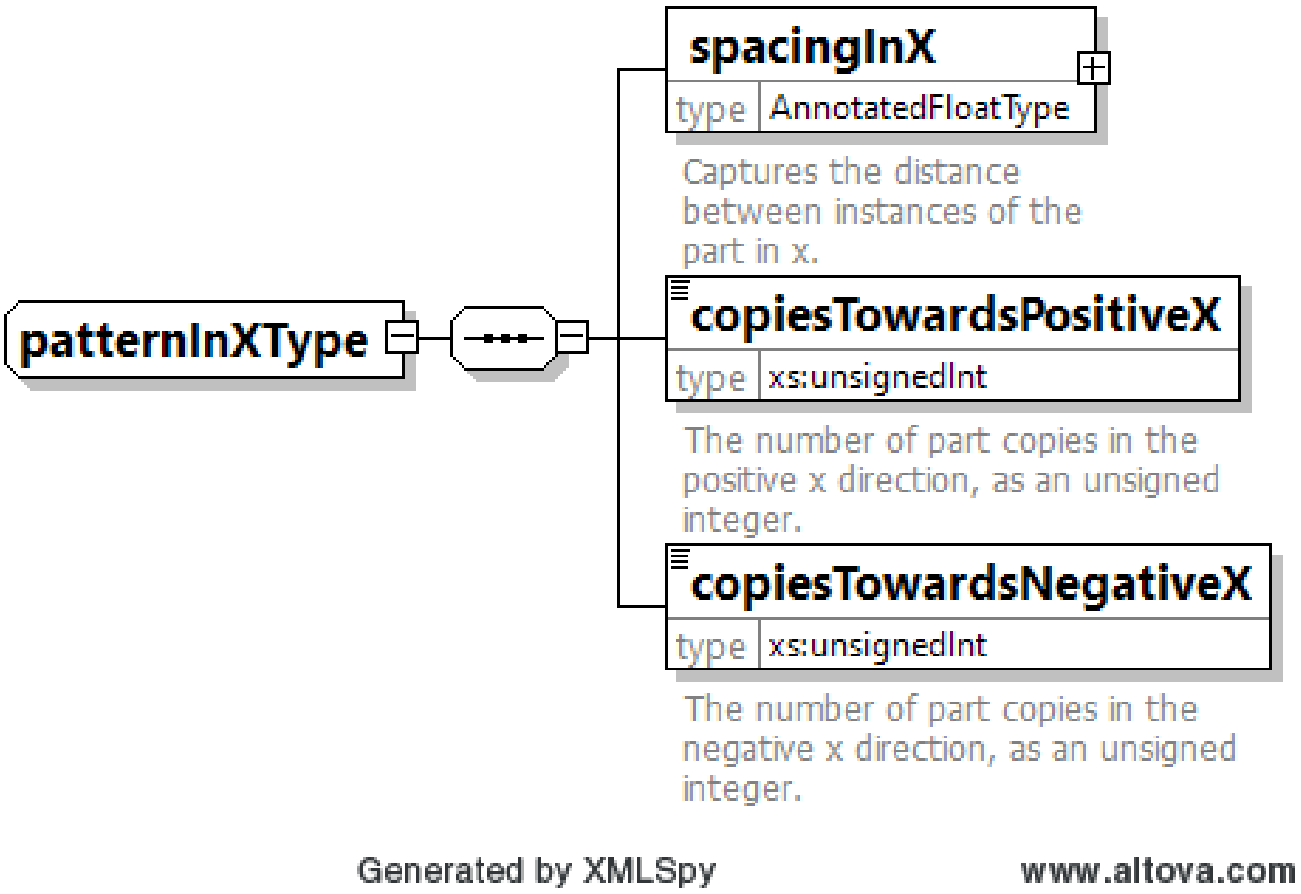

Figure 70: patternInX element

B.32 patternInY

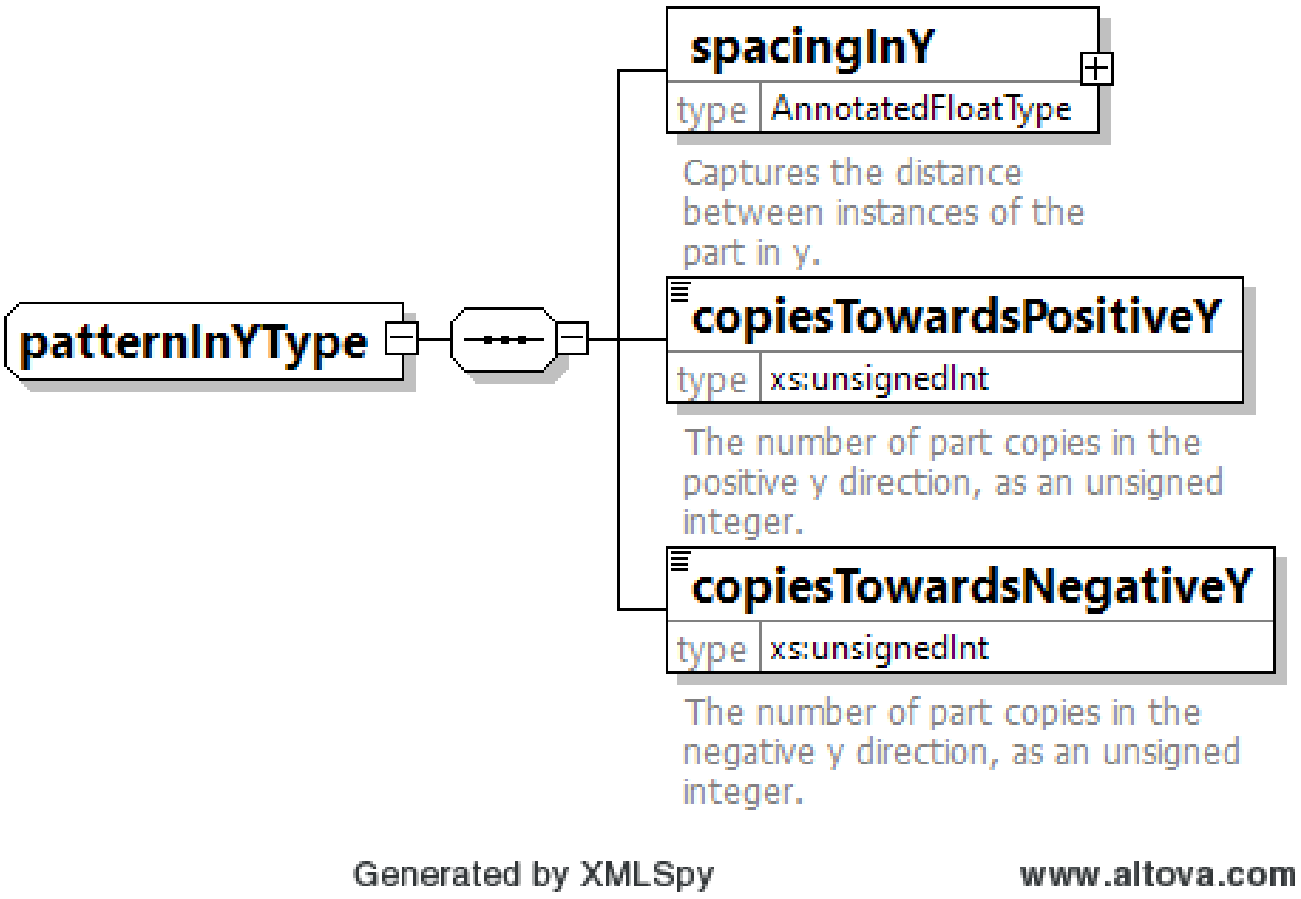

Figure 71: patternInY element

B.33 pointSupportSettings

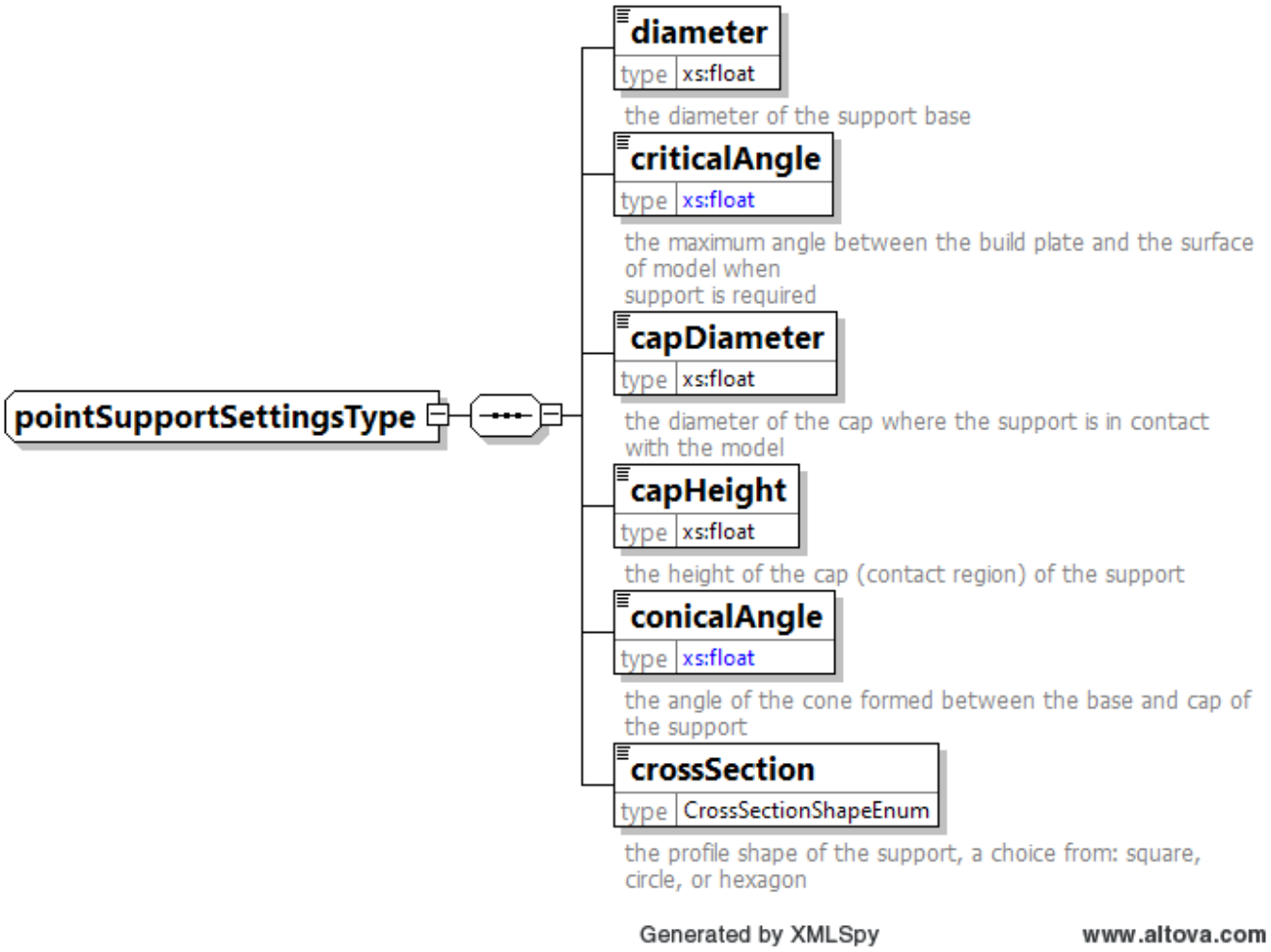

Figure 72: pointSupportSettings element

B.34 PolymorphicProcessPlan

Figure 73: PolymorphicProcessPlan element

B.35 recoaterSettings

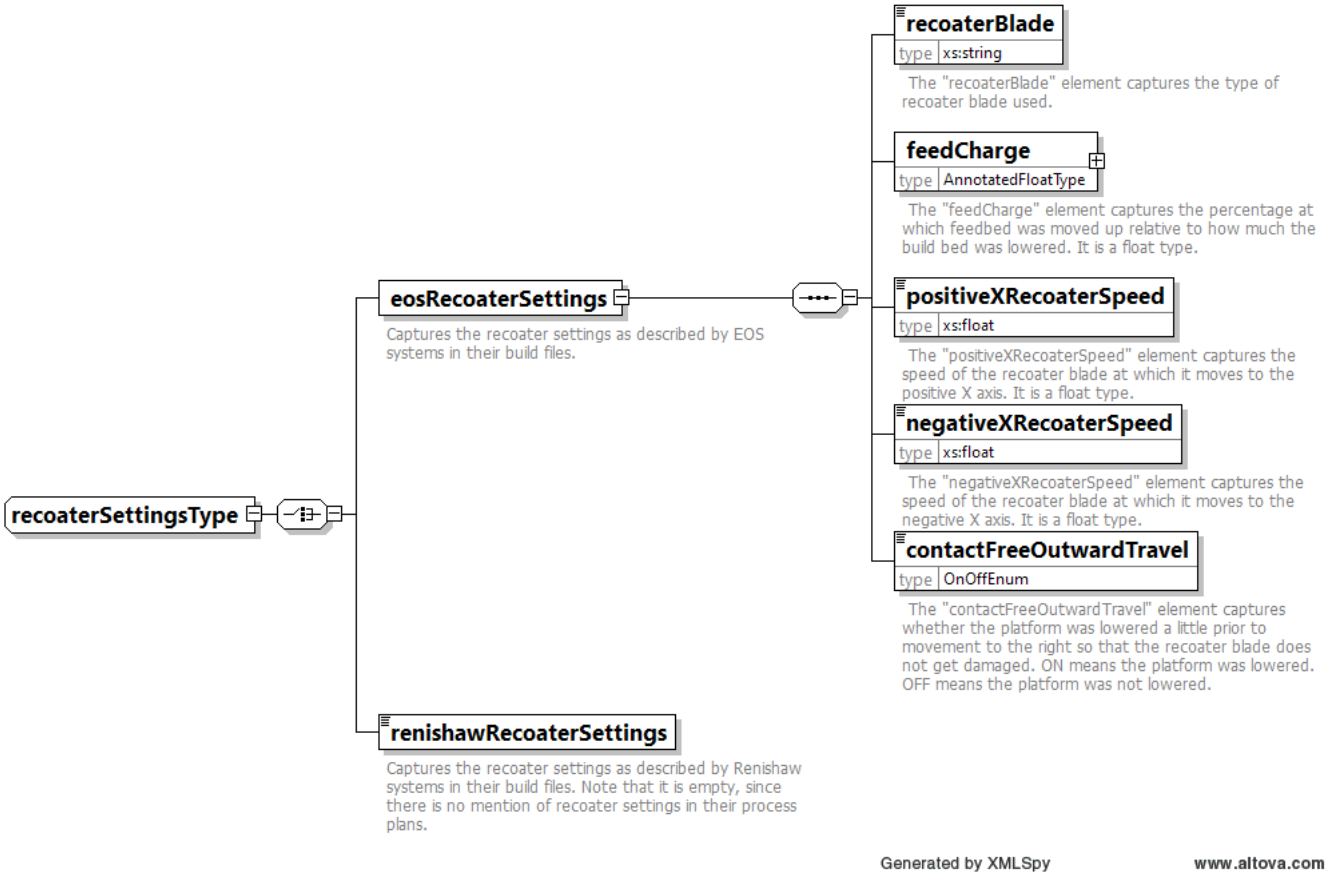

Figure 74: recoaterSettings element

B.36 RenishawAdvancedSettings

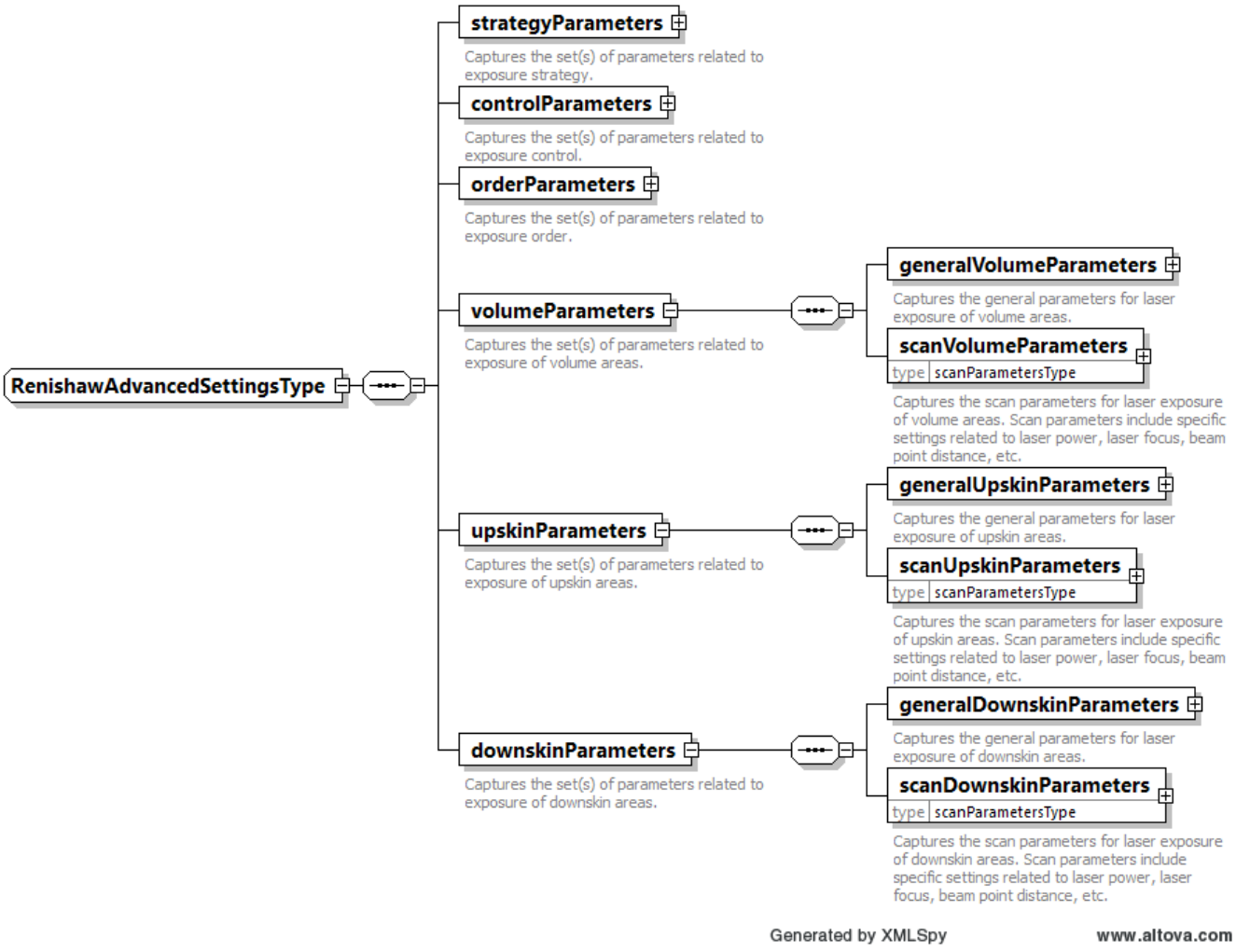

Figure 75: RenishawAdvancedSettings element

B.37 scanParametersType

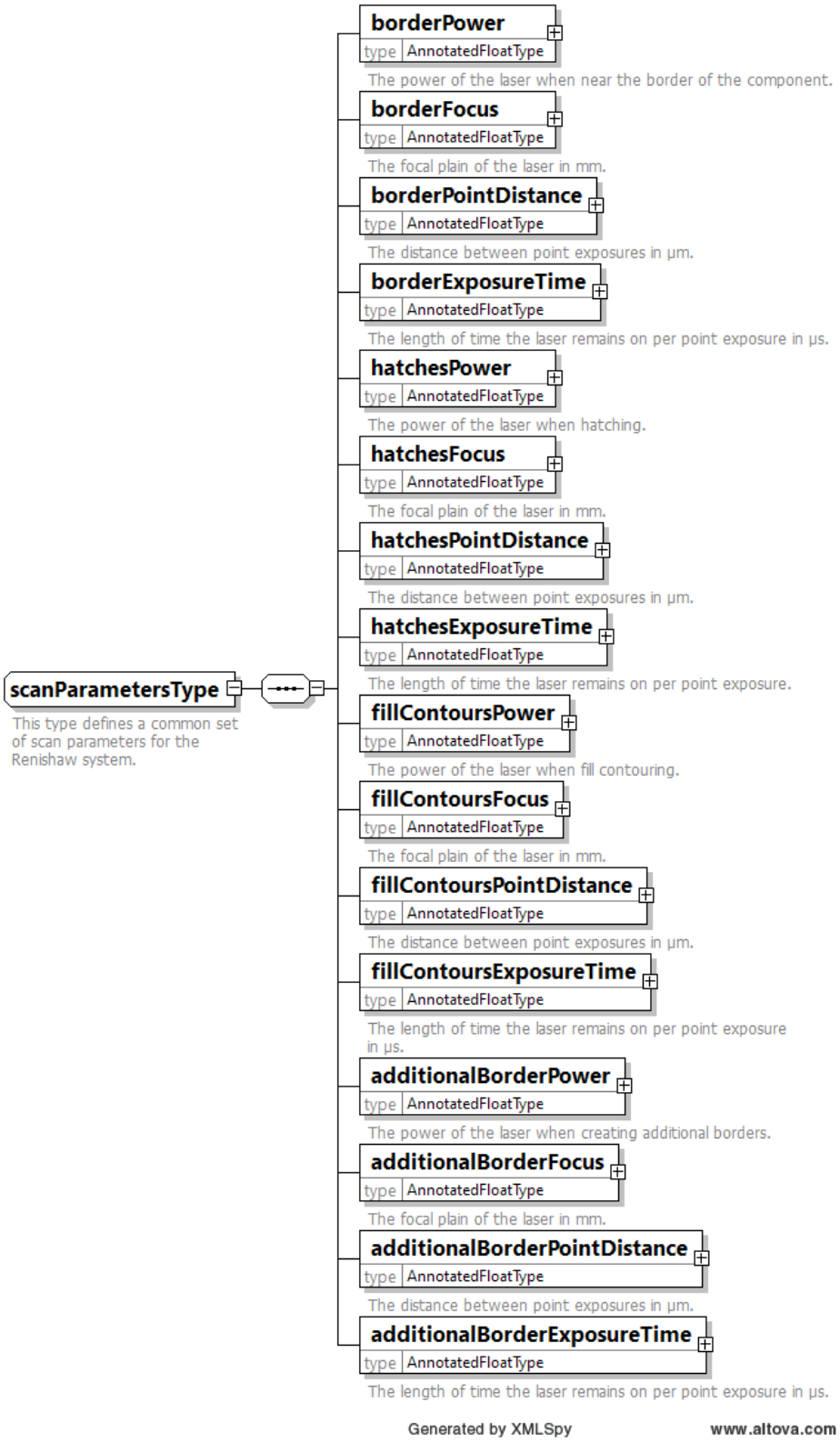

Figure 76: scanParametersType element

B.38 skinCoreSettings

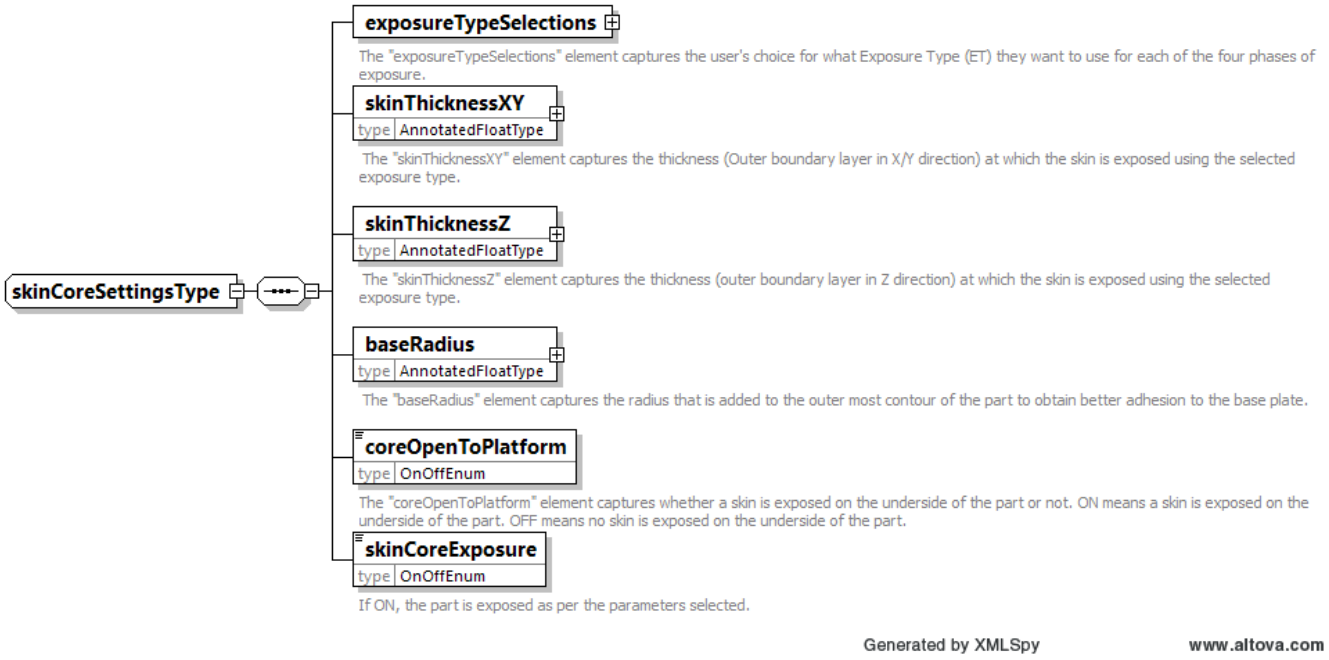

Figure 77: skinCoreSettings element

B.39 skipLayerSettings

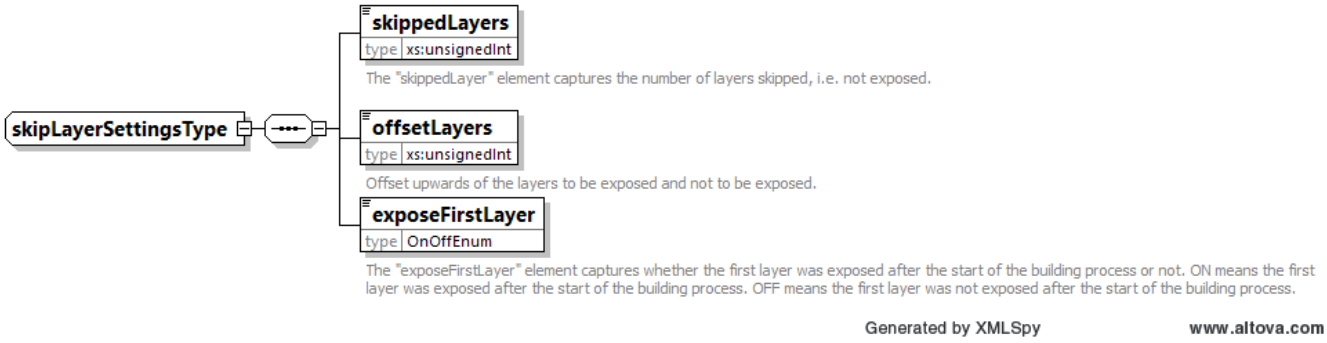

Figure 78: skipLayerSettings element

B.40 SLIHatchSettings

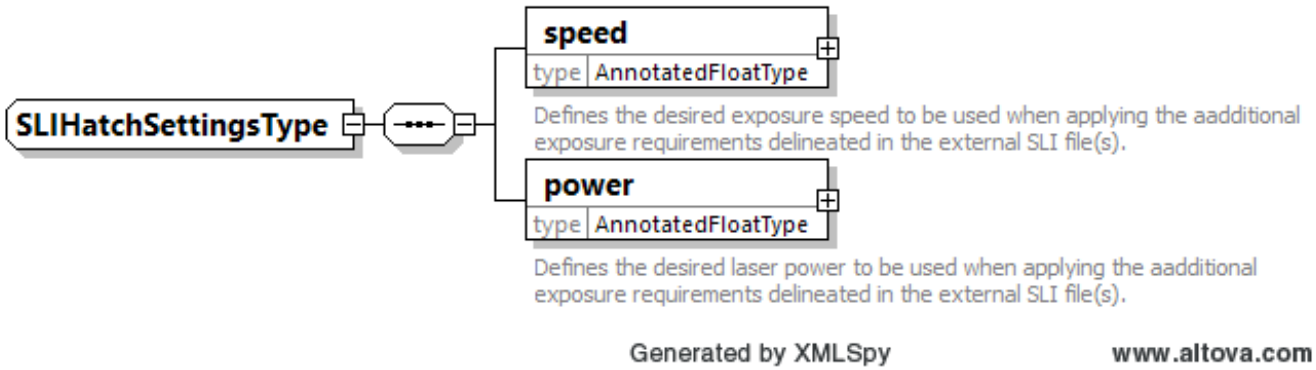

Figure 79: SLIHatchSettings element

B.41 strategyParameters

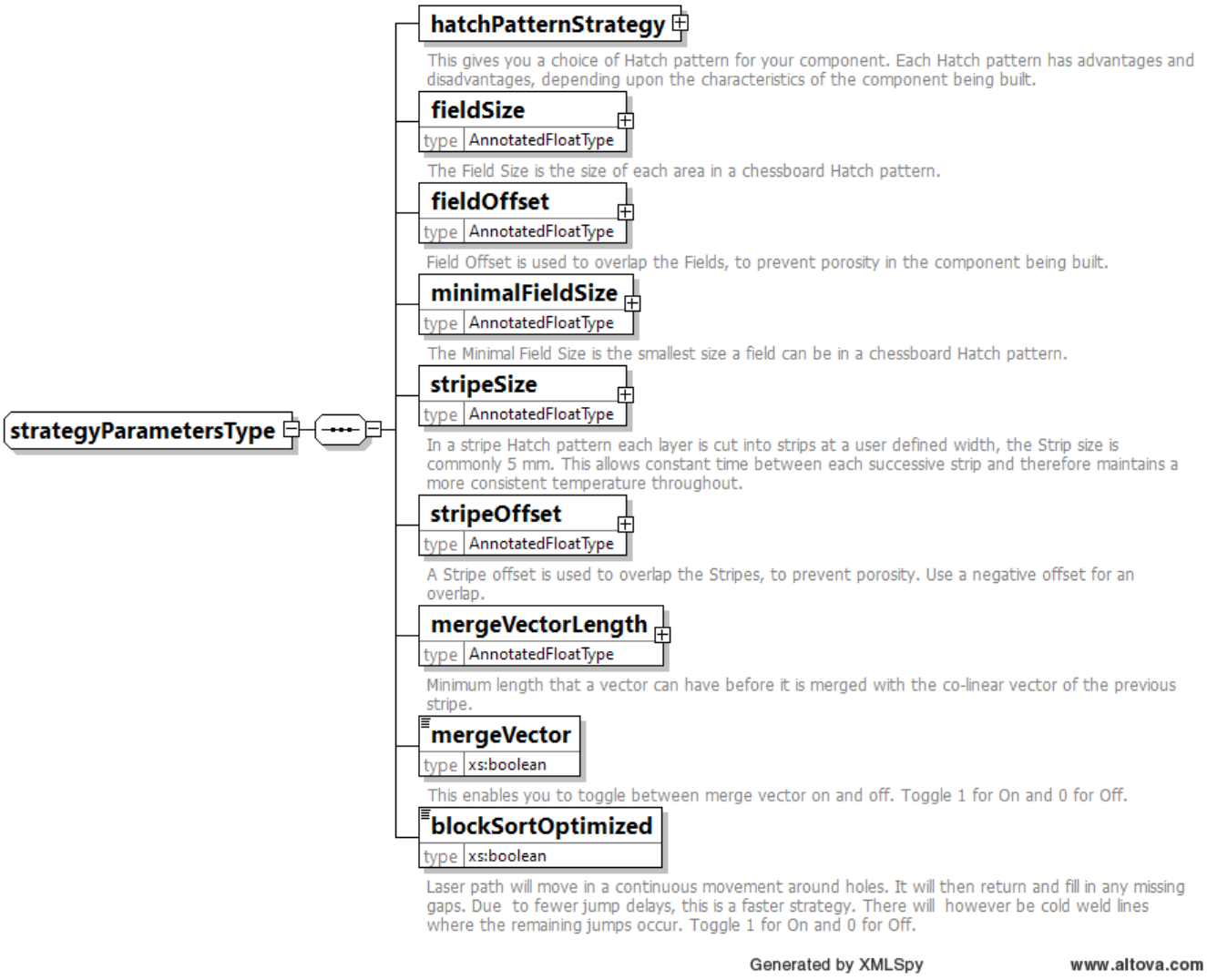

Figure 80: strategyParameters element

B.42 stripesCategorySettings

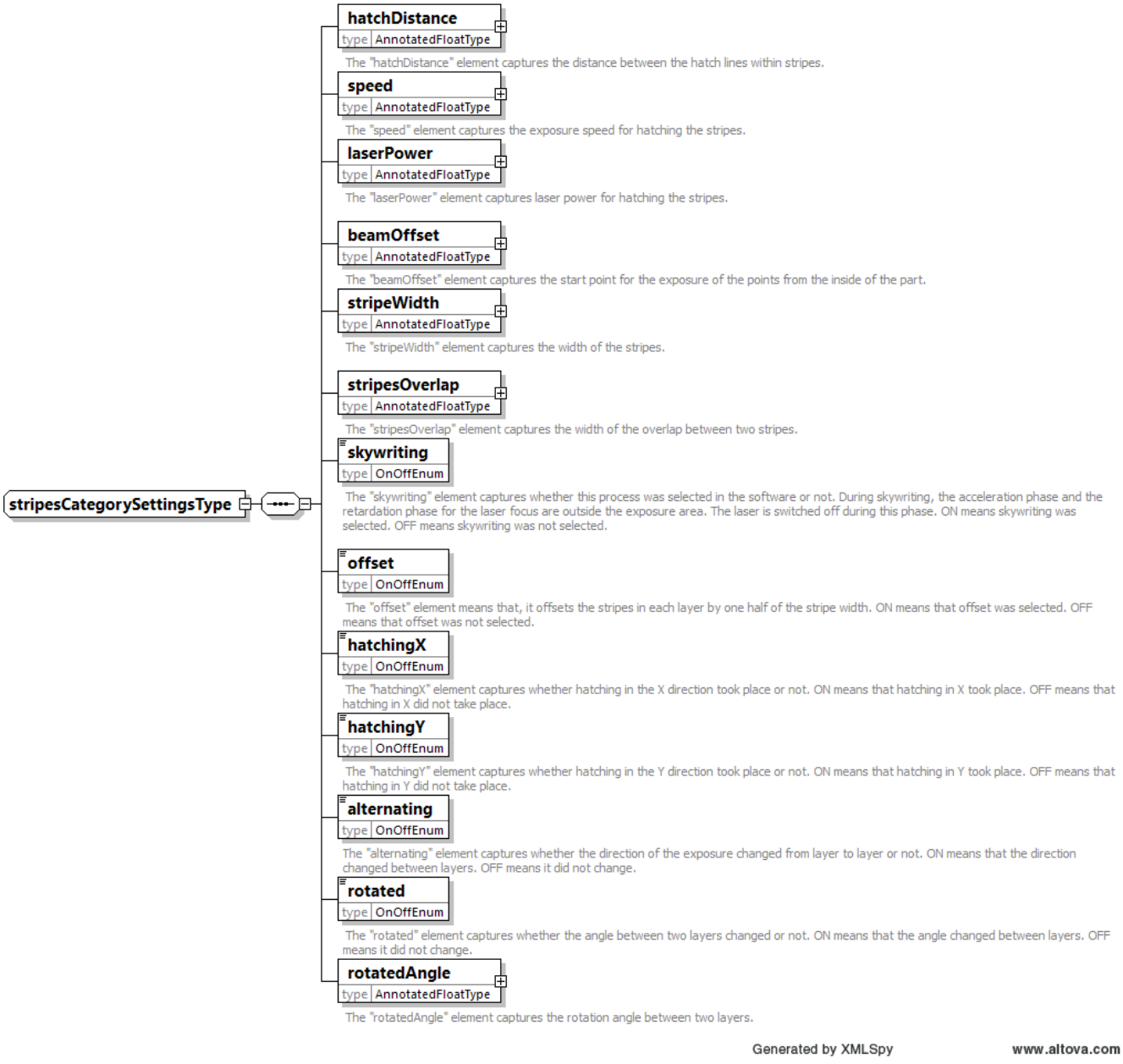

Figure 81: stripesCategorySettings element

B.43 supportStartLocation

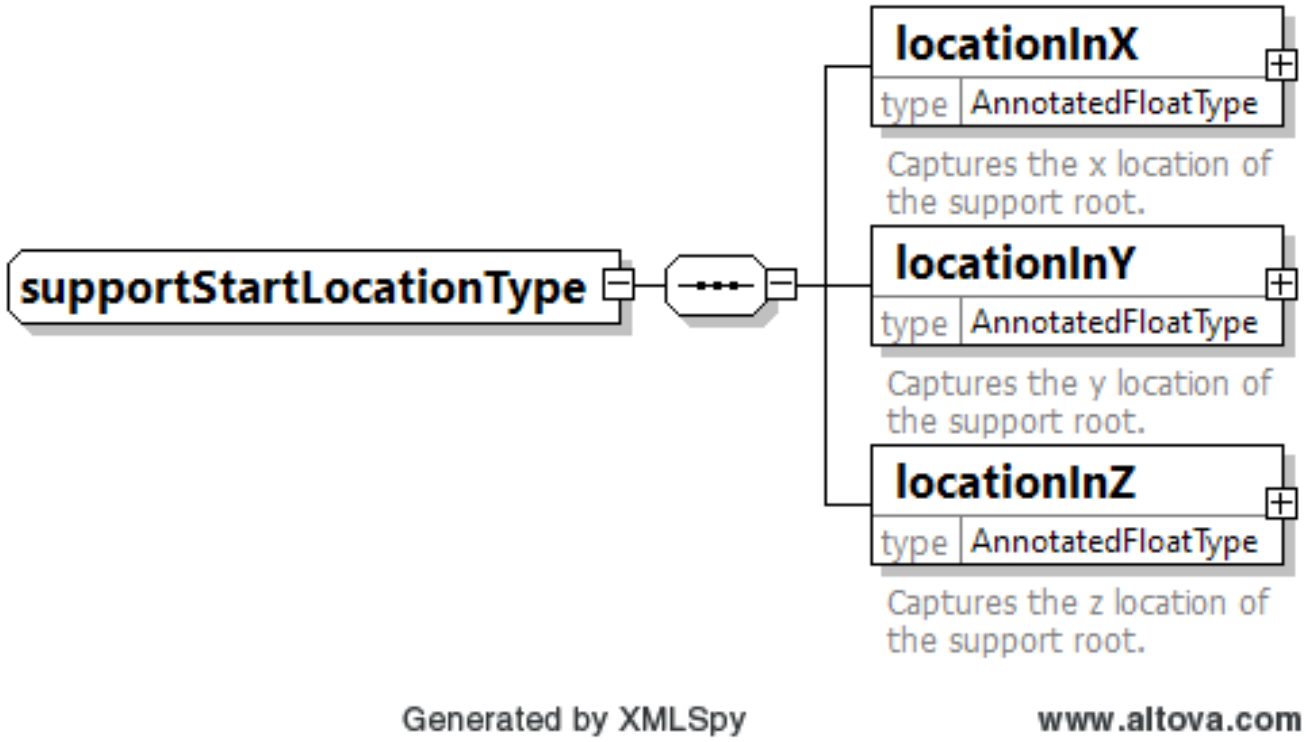

Figure 82: supportStartLocation element

B.44 supportStyle

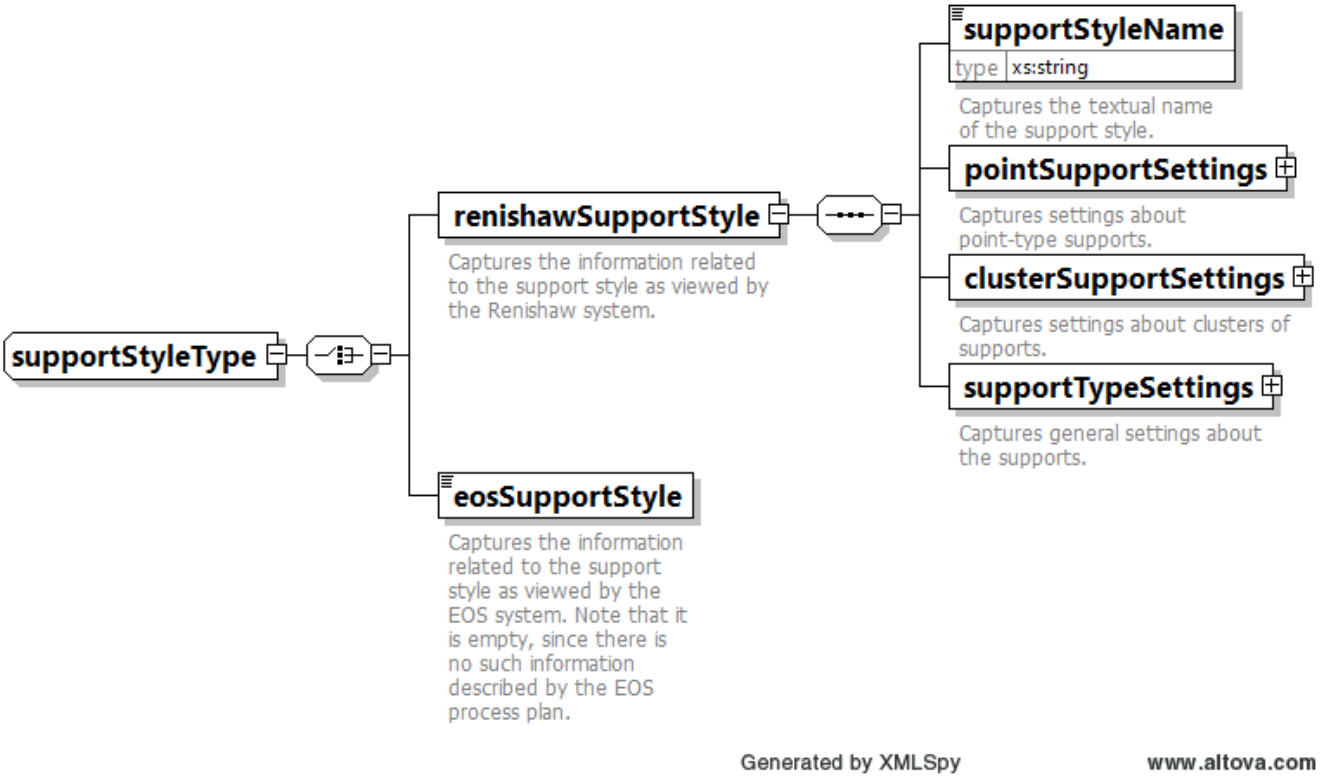

Figure 83: supportStyle element

B.45 supportTypeSettings

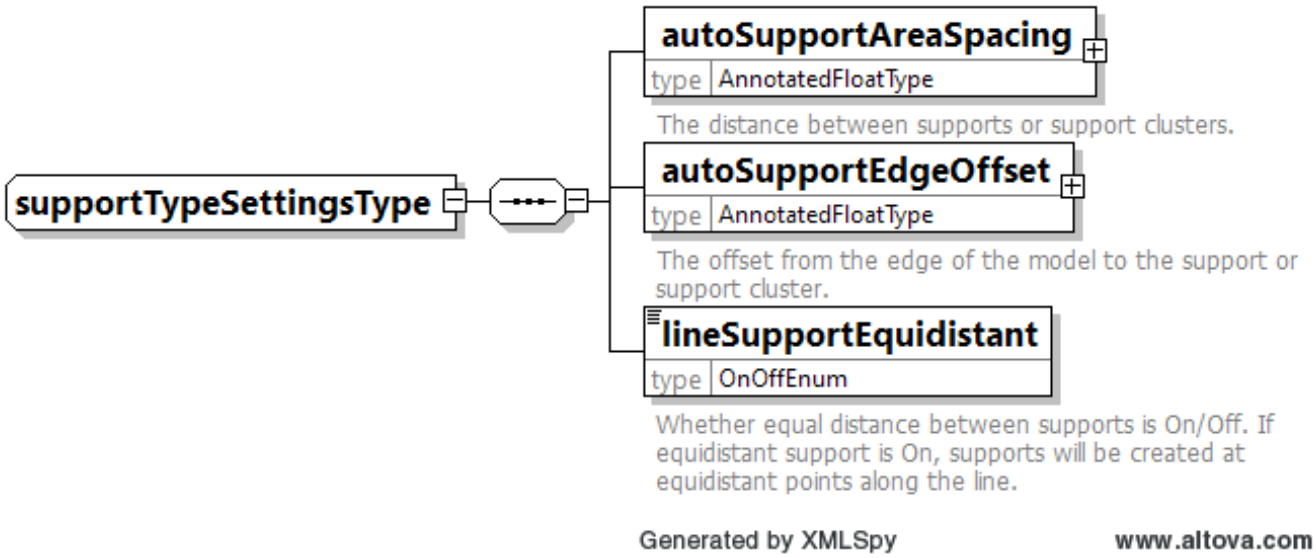

Figure 84: supportTypeSettings element

B.46 unitSettings

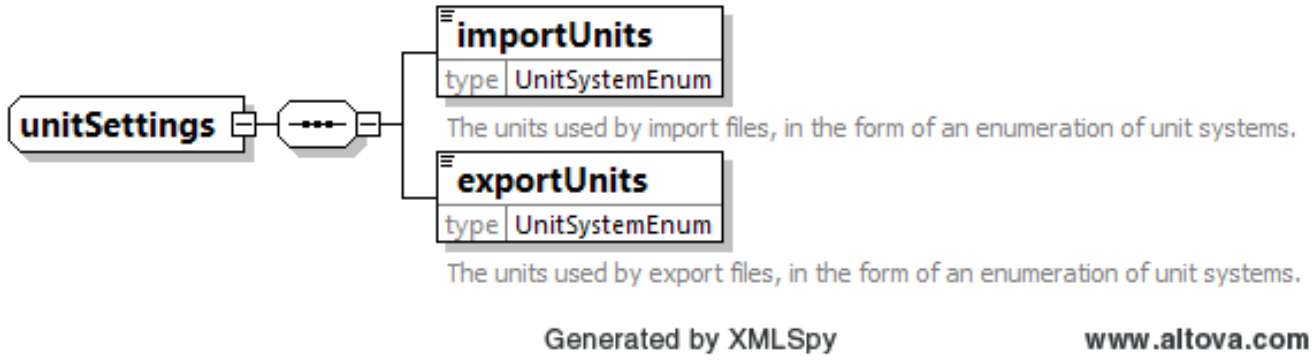

Figure 85: unitSettings element

B.47 upDownCategorySettings

Figure 86: upDownCategorySettings element

B.48 ZipFileType

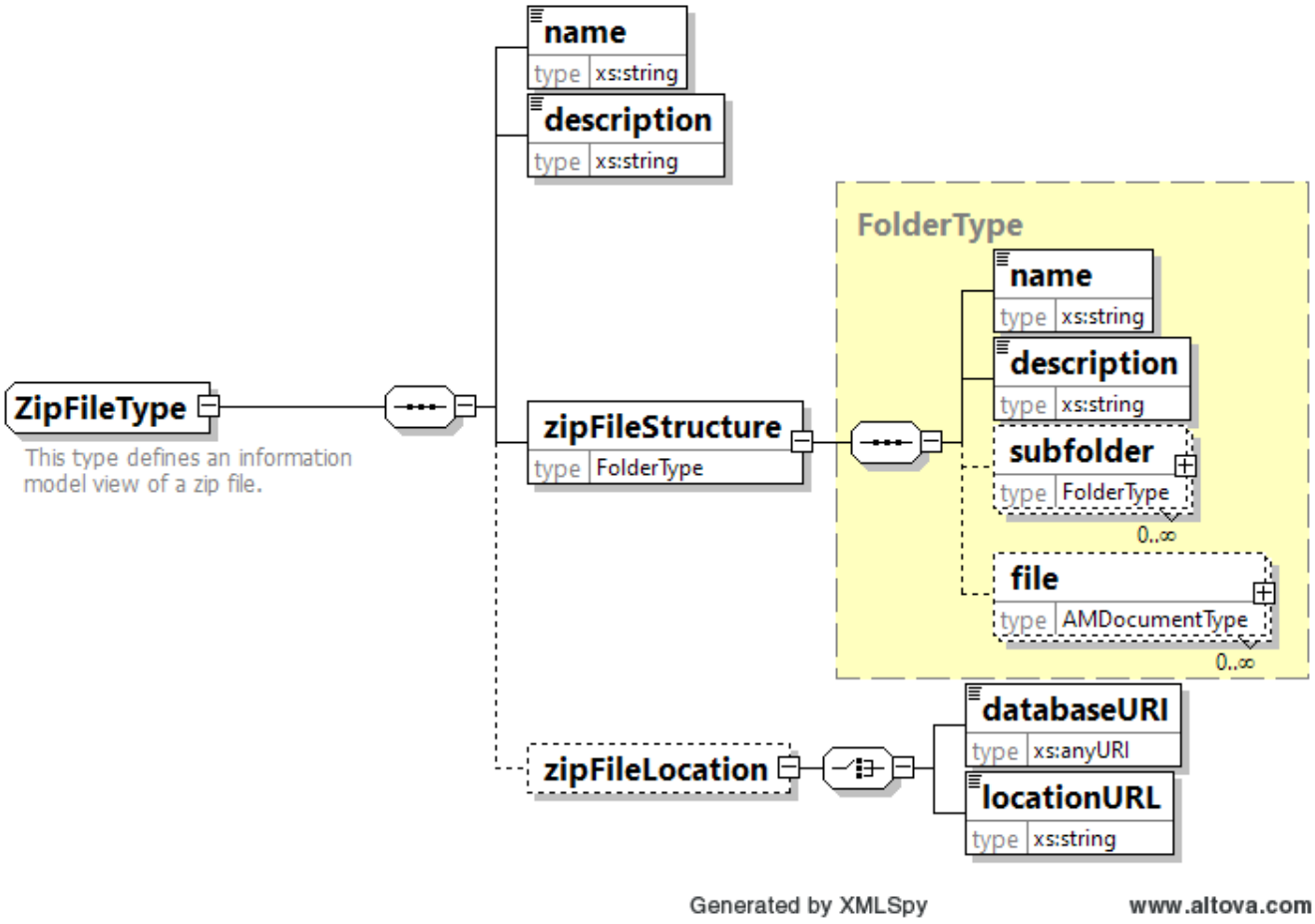

Figure 87: ZipFileType element

C Code for the XML Schema Information Model

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2022 rel. 2 (x64) (http://www.altova.com) by
   ↪ Sariel Coronado -->
<xs:schema xmlns="http://adml.lab.mcgill.ca/PPPSchema" xmlns:xs="http
   ↪ ://www.w3.org/2001/XMLSchema" targetNamespace="http://adml.lab.
   ↪ mcgill.ca/PPPSchema" elementFormDefault="qualified"
   ↪ attributeFormDefault="unqualified" version="1.0">
  <xs:element name="PolymorphicProcessPlan">
    <xs:annotation>
      <xs:documentation> Root element</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="machineSpecsData" type="machineSpecsDataType">
          <xs:annotation>
            <xs:documentation>The machineSpecsData element contains basic
               ↪  information about the LPBF machine that is doing the
               ↪ printing.
</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="unitSettings" type="unitSettingsType"
           ↪ minOccurs="0">
          <xs:annotation>
            <xs:documentation>The optional "unitSettings" element
               ↪ captures the information related to the system of units
               ↪  used by import or export files related to the process
               ↪ plan.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="exportSettings" type="exportSettingsType"
           ↪ minOccurs="0">
          <xs:annotation>
            <xs:documentation>The optional "exportSettings" element
               ↪ captures the filepaths of any export files related to
               ↪ the process plan.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="buildSettings" type="buildSettingsType">
          <xs:annotation>
            <xs:documentation>The "buildSettings" element captures the
               ↪ start height and the final height of the build, and the
               ↪  characteristics of pre-exposure settings.</xs:
               ↪ documentation>
          </xs:annotation>
```

```
      </xs:element>
      <xs:element name="materialSpecificSettings" type="
         ↪ materialSpecificSettingsType">
        <xs:annotation>
          <xs:documentation>The "materialSpecificSettings" element
             ↪ captures the settings (either advanced or default)
             ↪ specific to the material used.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="recoaterSettings" type="recoaterSettingsType">
        <xs:annotation>
          <xs:documentation>The "recoaterSettings" element captures the
             ↪  type of the recoater blade used, feed
charge, and blade's speed.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="miscellaneousSettings" type="
         ↪ miscellaneousSettingsType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>The "miscellaneousSettings" element
             ↪ contains the set of settings which does not necessarily
             ↪  correspond elsewhere, such as language or currency
             ↪ settings.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="partData" type="partDataType">
        <xs:annotation>
          <xs:documentation>The "partData" element captures information
             ↪  about the STL models that have been imported onto the
             ↪ build plate, their orientations, scalings, and if there
             ↪  is any instancing (copies).</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="buildSupportsData" type="buildSupportsDataType
         ↪ ">
        <xs:annotation>
          <xs:documentation>Captures the set of data regarding the
             ↪ support structures necessary for the generation of the
             ↪ part by the LPBF machine. It contains an abstracted set
             ↪  of 0 to many supports.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="buildCommandDataZipfile" type="ZipfileType"
         ↪ minOccurs="0">
        <xs:annotation>
          <xs:documentation>The optional "buildCommandDataZipfile"
             ↪ represents the possible reference to an export file
```

```
            ↪ that contains the machine language commands that
            ↪ contain the scanpaths used by the LBPF laser.</xs:
            ↪ documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="ZipfileType">
  <xs:annotation>
   <xs:documentation> This type defines an information
model view of a zip file.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="description" type="xs:string"/>
   <xs:element name="zipfileStructure" type="FolderType"/>
   <xs:element name="zipfileLocation" minOccurs="0">
    <xs:complexType>
     <xs:choice>
      <xs:element name="databaseURI" type="xs:anyURI">
       <xs:annotation>
        <xs:appinfo>
         <module>module-blob-host</module>
        </xs:appinfo>
       </xs:annotation>
      </xs:element>
      <xs:element name="locationURL" type="xs:string"/>
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="FolderType">
  <xs:annotation>
   <xs:documentation> This type defines an information model view of
      ↪  a folder.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="description" type="xs:string"/>
   <xs:element name="subfolder" type="FolderType" minOccurs="0"
      ↪ maxOccurs="unbounded"/>
   <xs:element name="file" type="AMDocumentType" minOccurs="0"
      ↪ maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
```

```
 <xs:complexType name="AMDocumentType">
  <xs:annotation>
   <xs:documentation> This type defines an electronic version of an
      ↪ AM document.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
   <xs:element name="modelName" type="xs:string" default="text"
      ↪ minOccurs="0">
    <xs:annotation>
     <xs:documentation> The "name" element captures the name of the
        ↪  model. </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="fileName" type="xs:string" default="text">
    <xs:annotation>
     <xs:documentation> The "fileName" element captures the name of
        ↪  the file.
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="software" type="xs:string" default="text"
      ↪ minOccurs="0">
    <xs:annotation>
     <xs:documentation> The optional "software" element captures
        ↪ the information about the
software application wherein the model was most recently edited. </xs:
  ↪ documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="author" type="xs:string" default="text"
      ↪ minOccurs="0">
    <xs:annotation>
     <xs:documentation>The optional "author" element is the author
        ↪ who created this file.
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="description" type="xs:string" default="text"
      ↪ minOccurs="0">
    <xs:annotation>
     <xs:documentation>The optional "description" element is a
        ↪ description of the model or any
additional information on the process used. </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="kind" type="xs:string" minOccurs="0">
    <xs:annotation>
```

```
      <xs:documentation>The type of the file</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="sizeInBytes" type="xs:unsignedInt" minOccurs
    ↪ ="0">
    <xs:annotation>
      <xs:documentation>The size of the file</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="createdDate" type="xs:dateTime" minOccurs="0">
    <xs:annotation>
      <xs:documentation>The date the file was created.</xs:
        ↪ documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="modifiedDate" type="xs:dateTime" minOccurs="0">
    <xs:annotation>
      <xs:documentation>The date the file was last modified.</xs:
        ↪ documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="fileLocation" type="xs:anyURI">
    <xs:annotation>
      <xs:appinfo>
        <module>module-blob-host</module>
      </xs:appinfo>
      <xs:documentation>The location of the file as an URI.</xs:
        ↪ documentation>
    </xs:annotation>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="AnnotatedFloatType">
 <xs:annotation>
  <xs:documentation> This type defines a data structure built
    ↪ around a floating point number, which includes the unit of
    ↪ measure, and optionally, the name and description.</xs:
    ↪ documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="name" type="xs:string" default="text" minOccurs
    ↪ ="0">
    <xs:annotation>
      <xs:documentation>The optional "name" element is the name of
        ↪ the variable.
      </xs:documentation>
    </xs:annotation>
```

```
    </xs:element>
    <xs:element name="value" type="xs:float">
      <xs:annotation>
        <xs:documentation>The "value" element captures the quantity
          ↪ being referred. It is a float type.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="unit" type="xs:string" default="text">
      <xs:annotation>
        <xs:documentation>The "unit" element captures the unit of
          ↪ measurement. For example, mm, cubic inches, Celsius, etc.
          ↪  </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="description" type="xs:string" default="text"
      ↪ minOccurs="0">
      <xs:annotation>
        <xs:documentation>The optional "description" element captures
          ↪ the description of the variable.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="exposureSelectionType">
  <xs:annotation>
    <xs:documentation> This type defines the choice of several
      ↪ exposure strategies in the EOS system. Each choice is an
      ↪ Exposure Type (ET) that the user can select. It will exhibit
      ↪  a certain kind of exposure behavior determined by what
      ↪ strategy it represents, and modified by the values of the
      ↪ parameters of the sets of settings that affect it. It is an
      ↪ "empty" element, in that it does not strictly contain a data
      ↪  structure. Rather, the presence of the named element is
      ↪ indication of it being selected.</xs:documentation>
  </xs:annotation>
  <xs:choice>
    <xs:element name="ContoursET">
      <xs:annotation>
        <xs:documentation>lf the path of the centre of the laser beam
          ↪ moves along the nominal contour of the part during
          ↪ exposure, the contour of the part is enlarged by the
          ↪ radius of the curing zone of the laser beam. The beam
          ↪ displacement compensates for this contour enlargement. It
          ↪  displaces the centre of the path of the laser beam
          ↪ inwards.</xs:documentation>
      </xs:annotation>
```

```
    </xs:element>
    <xs:element name="SortedET">
      <xs:annotation>
        <xs:documentation>Sequential exposure with Skywriting.
          ↪ Skywriting description: "During Skywriting the
          ↪ acceleration phase and the retardation phase for the
          ↪ laser beam are outside the exposure area. The laser is
          ↪ switched off during these phases."</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="UnsortedET">
      <xs:annotation>
        <xs:documentation>Continous exposure without Skywriting.
          ↪ Skywriting description: "During Skywriting the
          ↪ acceleration phase and the retardation phase for the
          ↪ laser beam are outside the exposure area. The laser is
          ↪ switched off during these phases." </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="UpDownskinET">
      <xs:annotation>
        <xs:documentation>With this exposure type the software checks
          ↪ during each layer whether there is an exposed area above
          ↪ or below the area exposed in the layer. The result is
          ↪ Upskin, Downskin or lnskin.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="StripesET">
      <xs:annotation>
        <xs:documentation>Exposure in stripes without Skywriting</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="StripesLxET">
      <xs:annotation>
        <xs:documentation>With this exposure type exposure is
          ↪ performed as for the exposure type Stripes, however not
          ↪ every layer is exposed, e.g., only every second or third
          ↪ layer.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="UpDownStripesET">
      <xs:annotation>
        <xs:documentation>This exposure type is a combination of the
          ↪ exposure types UpDownskin and Stripes. The upskin and
          ↪ downskin areas are exposed with parameters for the
          ↪ exposure type UnSorted without stripes.</xs:documentation
```

```
      ↪ >
    </xs:annotation>
  </xs:element>
  <xs:element name="ChessET">
    <xs:annotation>
      <xs:documentation>Exposure with the exposure type Chess is
        ↪ performed like the exposure type Squares in squares and
        ↪ gaps. Layout and parameters for the exposure type Chess
        ↪ and the exposure type Squares only differ in the
        ↪ direction of exposure and in the sequence of exposure of
        ↪ the squares. During the exposure, first all squares in
        ↪ one direction of exposure are exposed in the defined
        ↪ sequence, then all squares in the other exposure
        ↪ direction. The gaps are exposed last.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ChessLxET">
    <xs:annotation>
      <xs:documentation>With this exposure type, exposure is
        ↪ performned as for the exposure type Chess, howeyer not
        ↪ every layer is exposed, e.g., only every second or third
        ↪ layer.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SquaresET">
    <xs:annotation>
      <xs:documentation>Exposure with the exposure type Squares is
        ↪ performed like the exposure type Chess in squares and
        ↪ gaps. Layout and parameters for the exposure type Chess
        ↪ and the exposure type Squares only differ in the
        ↪ direction of exposure and in the sequence of exposure of
        ↪ the squares. During the exposure, first all squares in
        ↪ one direction of exposure are exposed in the defined
        ↪ sequence, then all squares in the other exposure
        ↪ direction. The gaps are exposed last.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SquaresLxET">
    <xs:annotation>
      <xs:documentation>With this exposure type, exposure is
        ↪ performned as for the exposure type Squares, howeyer not
        ↪ every layer is exposed, e.g., only every second or third
        ↪ layer.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="SLI_HatchET">
    <xs:annotation>
```

```
        <xs:documentation>With this exposure type, defined exposure
           ↪ requirements are included in the SLI file.</xs:
           ↪ documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="SLI_HatchLxET">
        <xs:annotation>
          <xs:documentation>With this exposure type, exposure is
             ↪ performed as for the exposure type SLI Hatch, however not
             ↪  every layer is exposed, e.g., only every second or third
             ↪  layer.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="No_ExposureET">
        <xs:annotation>
          <xs:documentation>No exposure is performed with this exposure
             ↪ type.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="scanParametersType">
    <xs:annotation>
      <xs:documentation>This type defines a common set
of scan parameters for the
Renishaw system.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="borderPower" type="AnnotatedFloatType">
        <xs:annotation>
          <xs:documentation>The power of the laser when near the border
             ↪ of the component.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="borderFocus" type="AnnotatedFloatType">
        <xs:annotation>
          <xs:documentation>The focal plain of the laser in mm.</xs:
             ↪ documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="borderPointDistance" type="AnnotatedFloatType">
        <xs:annotation>
          <xs:documentation>The distance between point exposures in
             ↪ microns.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="borderExposureTime" type="AnnotatedFloatType">
```

```
  <xs:annotation>
    <xs:documentation>The length of time the laser remains on per
      ↪ point exposure in microns.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="hatchesPower" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>The power of the laser when hatching.</xs:
      ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="hatchesFocus" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>The focal plain of the laser in mm.</xs:
      ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="hatchesPointDistance" type="AnnotatedFloatType
  ↪ ">
  <xs:annotation>
    <xs:documentation>The distance between point exposures in
      ↪ microns.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="hatchesExposureTime" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>The length of time the laser remains on per
      ↪ point exposure.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="fillContoursPower" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>The power of the laser when fill contouring
      ↪ .</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="fillContoursFocus" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>The focal plain of the laser in mm.</xs:
      ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="fillContoursPointDistance" type="
  ↪ AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>The distance between point exposures in
      ↪ microns.</xs:documentation>
```

```
      </xs:annotation>
    </xs:element>
    <xs:element name="fillContoursExposureTime" type="
      ↪ AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The length of time the laser remains on per
          ↪ point exposure
in microseconds.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="additionalBorderPower" type="AnnotatedFloatType
      ↪ ">
      <xs:annotation>
        <xs:documentation>The power of the laser when creating
          ↪ additional borders.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="additionalBorderFocus" type="AnnotatedFloatType
      ↪ ">
      <xs:annotation>
        <xs:documentation>The focal plain of the laser in mm.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="additionalBorderPointDistance" type="
      ↪ AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The distance between point exposures in
          ↪ microns.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="additionalBorderExposureTime" type="
      ↪ AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The length of time the laser remains on per
          ↪ point exposure in microseconds.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:simpleType name="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The OnOffEnum enumerates values that mean on
      ↪ and off.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
```

```
    <xs:enumeration value="ON"/>
    <xs:enumeration value="OFF"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="UnitSystemEnum">
  <xs:annotation>
    <xs:documentation> This type defines the system of units used --
       ↪ it is an enumeration of "millimeters" and "inches"</xs:
       ↪ documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="MILLIMETERS"/>
    <xs:enumeration value="INCHES"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SupportOriginEnum">
  <xs:annotation>
    <xs:documentation> This type defines the origin of each support
       ↪ -- it is an enumeration of "buildplate" and "part". It means
       ↪ , does the support grow from the build plate directly, or
       ↪ does it grow from some surface on the part itself?</xs:
       ↪ documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="BUILDPLATE"/>
    <xs:enumeration value="PART"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SupportAreaEnum">
  <xs:annotation>
    <xs:documentation> This type defines whether the support is an
       ↪ edge support or an area support -- it is an enumeration of "
       ↪ EDGE" and "AREA".</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="AREA"/>
    <xs:enumeration value="EDGE"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CrossSectionShapeEnum">
  <xs:annotation>
    <xs:documentation> This type defines whether the support has a
       ↪ profile shape of a square, circle, or hexagon -- it is an
       ↪ enumeration of "SQUARE", "CIRCLE", and "HEXAGON".</xs:
       ↪ documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
```

```
    <xs:enumeration value="SQUARE"/>
    <xs:enumeration value="CIRCLE"/>
    <xs:enumeration value="HEXAGON"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DFBE_SwitchingEnum">
  <xs:annotation>
    <xs:documentation> This type defines whether the 'dual focus beam
       ↪  expander' setting is to have the laser move in such a way
       ↪ that it sinters one part after another (PART) or whether it
       ↪ sinters similar features in parts one after another (LAYER)
       ↪ </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="PART"/>
    <xs:enumeration value="LAYER"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="EOS_advancedSettingsType">
  <xs:sequence>
    <xs:element name="exposureSettings" type="exposureSettingsType">
      <xs:annotation>
        <xs:documentation>The "exposureSettings" element captures the
           ↪ settings of the LPBF machine during the exposure
           ↪ processes.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="exposureCoordinatesCorrection">
      <xs:annotation>
        <xs:documentation>The "exposureCoordinatesCorrection" element
           ↪ captures the floating number values that allow the user
           ↪ to tweak the location and orientation of the exposure
           ↪ area if calibration is required.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="xDirection" type="xs:float">
            <xs:annotation>
              <xs:documentation>Moves the exposure area by the value
                 ↪ given in the X direction.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="yDirection" type="xs:float">
            <xs:annotation>
              <xs:documentation>Moves the exposure area by the value
                 ↪ given in the Y direction.</xs:documentation>
            </xs:annotation>
```

```
      </xs:element>
      <xs:element name="rotation" type="xs:float">
        <xs:annotation>
          <xs:documentation>Rotates the exposure field by the value
            ↪ given.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="HWI_Parameters">
  <xs:annotation>
    <xs:documentation>The "HWI_Parameters" element captures a set
      ↪ of parameters that the EOS manual groups under a window
      ↪ called "HWI Parameters". Unfortunately it neglects to
      ↪ explain what "HWI" is. However, the grouping was retained
      ↪  since it was useful.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="materialScalingFactors">
        <xs:annotation>
          <xs:documentation> The "materialScalingFactors" element
            ↪ captures how much bigger or smaller the powder
            ↪ needed to be than the nominal value. </xs:
            ↪ documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="x" type="xs:float" default="0">
              <xs:annotation>
                <xs:documentation> The "x" element captures the
                  ↪ material dependent scaling values for all
      parts in X direction </xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="y" type="xs:float" default="0">
              <xs:annotation>
                <xs:documentation> The "y" element captures the
                  ↪ material dependent scaling values for all
      parts in Y direction. </xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="z0" type="xs:float" default="0">
              <xs:annotation>
                <xs:documentation> The "z0" element captures the
                  ↪ material dependent scaling values
```

```
applicable to all parts at job height Z = 0 mm </xs:
   ↪ documentation>
      </xs:annotation>
     </xs:element>
     <xs:element name="z200" type="xs:float" default="0">
      <xs:annotation>
       <xs:documentation> The "z200" element captures the
          ↪ material dependent scaling values
applicable to all parts at job height Z = 7.87 in. </xs:
   ↪ documentation>
      </xs:annotation>
     </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="beamOffset" type="xs:float">
  <xs:annotation>
   <xs:documentation> The "beamOffset" element is a part (
      ↪ part group) specific beam offset parameter. </xs:
      ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="O2_startLimit" type="xs:float">
  <xs:annotation>
   <xs:documentation>Displays the maximum oxygen content in
      ↪ the process chamber at job start.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="buildPlatformNominalTemp" type="xs:float">
  <xs:annotation>
   <xs:documentation>Displays the nominal temperature of the
      ↪ building platform heating.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="dualFocusBeamExpanderSwitching" type="
   ↪ DFBE_SwitchingEnum">
  <xs:annotation>
   <xs:documentation> The "dualFocusBeamExpanderSwitching"
      ↪ element captures the choice between the laser
      ↪ exposing one part after the other, or exposing
      ↪ similar features in parts in sequence.</xs:
      ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="switchOffAirSupply" type="OnOffEnum">
  <xs:annotation>
```

```
            <xs:documentation>If ON, the compressed air supply is
               ↪ switched off after building is completed.</xs:
               ↪ documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="homeCollectorBeforeStart" type="OnOffEnum">
            <xs:annotation>
              <xs:documentation>If ON, when the job is started, the
                 ↪ collector platfomi is homed-</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="homeDispenserBeforeStart" type="OnOffEnum">
            <xs:annotation>
              <xs:documentation>If ON, when the job is started, the
                 ↪ dispenser platfonn is homed.</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="machineSpecsDataType">
  <xs:sequence>
    <xs:element name="manufacturer" type="xs:string">
      <xs:annotation>
        <xs:documentation>The manufacturer of the LPBF machine.</xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="model" type="xs:string">
      <xs:annotation>
        <xs:documentation>The name of the model of the LPBF machine
           ↪ used for the build file.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="buildVolume" type="buildVolumeType">
      <xs:annotation>
        <xs:documentation>The "buildVolume" element captures the
           ↪ dimensions of the build volume.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="buildVolumeType">
  <xs:sequence>
    <xs:element name="sizeInX" type="AnnotatedFloatType">
```

```
      <xs:annotation>
        <xs:documentation>Build volume dimensions in x.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sizeInY" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Build volume dimensions in y.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sizeInZ" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Build volume dimensions in z.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="unitSettingsType">
  <xs:sequence>
    <xs:element name="importUnits" type="UnitSystemEnum">
      <xs:annotation>
        <xs:documentation>The units used by import files, in the form
          ↪ of an enumeration of unit systems.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="exportUnits" type="UnitSystemEnum">
      <xs:annotation>
        <xs:documentation>The units used by export files, in the form
          ↪ of an enumeration of unit systems.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="exportSettingsType">
  <xs:sequence>
    <xs:element name="fileExportData" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The "fileExportData" element represents 1 to
          ↪  many instances of an exported file containing data,
          ↪ related to the process plan.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="fileExtension" type="xs:string">
            <xs:annotation>
```

```
            <xs:documentation>The "fileExtension" element defines what
               ↪  type of exported file the fileExportPath relates to
               ↪ .</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="exportPath" type="xs:anyURI">
            <xs:annotation>
              <xs:documentation>The "exportPath" element represents the
                 ↪ URI location of the exported file.</xs:documentation
                 ↪ >
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="buildSettingsType">
  <xs:sequence>
    <xs:element name="startHeight" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "startHeight" element captures the
           ↪ height at which the building process started.</xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="finalHeight" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "finalHeight" element captures the
           ↪ height at which the building process stopped.</xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="layerThickness" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "layerThickness" element captures the
           ↪ distance which the build platform is lowered prior to
           ↪ printing each new layer.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="layersCount" type="xs:int">
      <xs:annotation>
        <xs:documentation>The "layersCount" element is an integer that
           ↪  captures the number of layers in the current build file
           ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
```

```
    <xs:element name="posX" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "posX" element is a floating number that
          ↪  captures the position of the centroid of the set of all
          ↪ parts in X (left/right) as measured from the centre of
          ↪ the build plate.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="posY" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "posY" element is a floating number that
          ↪  captures the position of the centroid of the set of all
          ↪ parts in Y (wipe axis) as measured from the centre of the
          ↪  build plate.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="DMLS_settings" type="DMLS_settingsType"
      ↪ minOccurs="0">
      <xs:annotation>
        <xs:documentation> The "DMLSSetting" element is a build
          ↪ setting specific to EOS Machines. It stands for Direkt
          ↪ Metall Laser Schmelzen. If DMLS is not selected, each
          ↪ layer in the preexposure are will be exposed twice at the
          ↪  selected speed. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="buildProperties" type="buildPropertiesType"
      ↪ minOccurs="0">
      <xs:annotation>
        <xs:documentation>The "buildProperties" element is an optional
          ↪  description of a set of properties of the build file
          ↪ that may be of interest to operators, such as total
          ↪ simualted build time, build cost, part volume, etc.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DMLS_settingsType">
  <xs:sequence>
    <xs:element name="DMLS" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation> The "DMLS" element describes whether DMLS
          ↪ is switched on or off. ON means DMLS is on. OFF means
          ↪ DMLS is off. </xs:documentation>
      </xs:annotation>
    </xs:element>
```

```
    <xs:element name="DMLSRange" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "DMLSRange" element gives the thickness
          ↪  of the area within which building is performed with the
          ↪ exposure speed reduced in accordance with DMLS rules.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="preExposureDMLS_Settings">
      <xs:annotation>
        <xs:documentation>The "preExposureDMLS_Settings" element
          ↪ captures the boolean responsible for DMLS activation in
          ↪ the pre-exposure phase (as defined by EOS) and the
          ↪ thickness of the area within exposure is performed as per
          ↪  the DMLS protocol.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="preExposure" type="OnOffEnum">
            <xs:annotation>
              <xs:documentation> The "preExposure" element captures
                ↪ whether the laser was on or not. If on, the selected
                ↪  area is exposed at high speed and then at DMLS
                ↪ speed. ON means the laser was on. OFF means the
                ↪ laser was off. </xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="preExposureRange" type="AnnotatedFloatType
            ↪ ">
            <xs:annotation>
              <xs:documentation> The "preExposureRange" element captures
                ↪  the thickness of the area within which exposure is
                ↪ first performed at the selected speed then DMLS
                ↪ speed. It is a float type. </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="buildPropertiesType">
  <xs:sequence>
    <xs:element name="buildTime">
      <xs:annotation>
        <xs:documentation>Captures the time estimated for the build
          ↪ process to take.</xs:documentation>
```

```
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="hours" type="xs:unsignedInt"/>
        <xs:element name="minutes" type="xs:unsignedInt"/>
        <xs:element name="seconds" type="xs:unsignedInt"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="buildCosts">
    <xs:annotation>
      <xs:documentation>Captures the estimated cost of producing the
        ↪  parts in the LBPF machine.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="materialCosts" type="xs:float"/>
        <xs:element name="energyCosts" type="xs:float"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="buildHeight" type="AnnotatedFloatType">
    <xs:annotation>
      <xs:documentation>Captures the total calculated build height
        ↪ of the parts.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="totalPartNumber" type="AnnotatedFloatType">
    <xs:annotation>
      <xs:documentation>Captures the number of parts being printed
        ↪ at the same time on the build plate.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="totalLayerNumber" type="AnnotatedFloatType">
    <xs:annotation>
      <xs:documentation>Captures the total number of layers required
        ↪  to print the parts in the LBPF machine.</xs:
        ↪ documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="totalPartVolume" type="AnnotatedFloatType">
    <xs:annotation>
      <xs:documentation>Captures the total volume occupied by all
        ↪ parts on the build plate, excluding the supports.</xs:
        ↪ documentation>
    </xs:annotation>
  </xs:element>
```

```
    <xs:element name="totalSupportVolume" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the total volume occupied by the
          ↪ supports of all the parts on the build plate.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="materialSpecificSettingsType">
  <xs:choice>
    <xs:element name="advancedSettings">
      <xs:annotation>
        <xs:documentation> The "advancedSettings" element captures
          ↪ custom settings specific for the material which includes
          ↪ material scaling factor, part scaling factor, beam offset
          ↪ , dual focus beam expander, and exposure setting. </xs:
          ↪ documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:choice>
          <xs:element name="EOS_advancedSettings" type="
            ↪ EOS_advancedSettingsType">
            <xs:annotation>
              <xs:documentation>Captures the advanced settings as
                ↪ described by EOS systems in their documentation.</xs
                ↪ :documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="RenishawAdvancedSettings" type="
            ↪ RenishawAdvancedSettingsType">
            <xs:annotation>
              <xs:documentation>Captures the advanced settings as
                ↪ described by Renishaw systems in their documentation
                ↪ .</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="defaultSettingsfile" type="AMDocumentType">
      <xs:annotation>
        <xs:documentation> The "defaultSettings" element captures the
          ↪ name of the material, software used, and a initialization
          ↪  parameter file. </xs:documentation>
      </xs:annotation>
    </xs:element>
```

```
  </xs:choice>
</xs:complexType>
<xs:complexType name="RenishawAdvancedSettingsType">
  <xs:sequence>
    <xs:element name="strategyParameters" type="
       ↪ strategyParametersType">
      <xs:annotation>
        <xs:documentation>Captures the set(s) of parameters related to
           ↪  exposure strategy.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="controlParameters" type="controlParametersType
       ↪ ">
      <xs:annotation>
        <xs:documentation>Captures the set(s) of parameters related to
           ↪  exposure control.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="orderParameters" type="orderParametersType">
      <xs:annotation>
        <xs:documentation>Captures the set(s) of parameters related to
           ↪  exposure order.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="volumeParameters">
      <xs:annotation>
        <xs:documentation>Captures the set(s) of parameters related to
           ↪  exposure of volume areas.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="generalVolumeParameters" type="
             ↪ generalVolumeParametersType">
            <xs:annotation>
              <xs:documentation>Captures the general parameters for
                 ↪ laser exposure of volume areas.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="scanVolumeParameters" type="
             ↪ scanParametersType">
            <xs:annotation>
              <xs:documentation>Captures the scan parameters for laser
                 ↪ exposure of volume areas. Scan parameters include
                 ↪ specific settings related to laser power, laser
                 ↪ focus, beam point distance, etc.</xs:documentation>
            </xs:annotation>
          </xs:element>
```

```
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="upskinParameters">
  <xs:annotation>
    <xs:documentation>Captures the set(s) of parameters related to
      ↪ exposure of upskin areas.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="generalUpskinParameters" type="
        ↪ generalUpskinParametersType">
        <xs:annotation>
          <xs:documentation>Captures the general parameters for
            ↪ laser exposure of upskin areas.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="scanUpskinParameters" type="
        ↪ scanParametersType">
        <xs:annotation>
          <xs:documentation>Captures the scan parameters for laser
            ↪ exposure of upskin areas. Scan parameters include
            ↪ specific settings related to laser power, laser
            ↪ focus, beam point distance, etc.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="downskinParameters">
  <xs:annotation>
    <xs:documentation>Captures the set(s) of parameters related to
      ↪ exposure of downskin areas.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="generalDownskinParameters" type="
        ↪ generalDownskinParametersType">
        <xs:annotation>
          <xs:documentation>Captures the general parameters for
            ↪ laser exposure of downskin areas.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="scanDownskinParameters" type="
        ↪ scanParametersType">
        <xs:annotation>
```

```
            <xs:documentation>Captures the scan parameters for laser
               ↪ exposure of downskin areas. Scan parameters include
               ↪ specific settings related to laser power, laser
               ↪ focus, beam point distance, etc.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="exposureSettingsType">
 <xs:sequence>
  <xs:element name="contourSettings" type="contourSettingsType">
   <xs:annotation>
    <xs:documentation>The "contourSettings" element captures the
       ↪ definition of parameters which influence how the "
       ↪ ContoursExposureType" and "UpDownStripesExposureType"
       ↪ function</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="edgeSettings" type="edgeSettingsType">
   <xs:annotation>
    <xs:documentation>The "edgeSettings" element captures the set
       ↪ of parameters used to define how points and thin areas of
       ↪  parts are exposed. Important for creating fine features
       ↪ since beam is larger than a point or thin areas. It is an
       ↪  optional setting on EOS, activated by the "edges" sub-
       ↪ element.</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="skipLayerSettings" type="skipLayerSettingsType
     ↪ ">
   <xs:annotation>
    <xs:documentation>The "skipLayerSettings" element captures how
       ↪  many layers of the hatching are skipped before exposure
       ↪ is performed again. </xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="chessCategorySettings" type="
     ↪ chessCategorySettingsType">
   <xs:annotation>
    <xs:documentation>The "chessCategorySettings" element captures
       ↪  the parameters for the exposure types that use the chess
       ↪  pattern.</xs:documentation>
   </xs:annotation>
  </xs:element>
```

```
    <xs:element name="upDownCategorySettings" type="
      ↪ upDownCategorySettingsType">
     <xs:annotation>
      <xs:documentation>The "upDownCategorySettings" element
        ↪ captures the parameters for the exposure of areas that
        ↪ bound loose metal powder above or below. </xs:
        ↪ documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="stripesCategorySettings" type="
      ↪ stripesCategorySettingsType">
     <xs:annotation>
      <xs:documentation>The "stripesCategorySettings" element
        ↪ captures the parameters for the exposure types that use
        ↪ the stripes pattern. </xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="skinCoreSettings" type="skinCoreSettingsType">
     <xs:annotation>
      <xs:documentation>The "skinCoreSettings" element captures the
        ↪ values used to define the 'skin' and 'core' regions for a
        ↪  part, and to select which exposure types are used in the
        ↪  pre, skin, core, and post exposure phases.</xs:
        ↪ documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="SLIHatchSettings" type="SLIHatchSettingsType">
     <xs:annotation>
      <xs:documentation> The "SLIHatchSetting" element defines the
        ↪ set of parameters that the machine will use when applying
        ↪  exposure types defined in external SLI files.</xs:
        ↪ documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="SLI_file" type="AMDocumentType" minOccurs="0"
      ↪ maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>An optional element, it denotes an external
        ↪ file that would contain exposure requirements as a way to
        ↪  customize exposure settings.</xs:documentation>
     </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="contourSettingsType">
  <xs:sequence>
    <xs:element name="contourEnabled" type="OnOffEnum">
```

```
      <xs:annotation>
        <xs:documentation>The "contourEnabled" element captures
           ↪ whether the laser beam exposes the
contour in the pre-exposure phase or not. ON means the laser beam
   ↪ exposes the contour. OFF means it does not. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="postContourEnabled" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation>The "postContour" element captures whether
           ↪ the laser beam exposes the contour in the post-exposure
           ↪ phase or not. ON means the laser beam exposes the contour
           ↪ . OFF means it does not. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="beamOffset" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>On the exposure of the contour, offsets the
           ↪ path of the centre of the laser beam inwards by the value
           ↪  entered.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="standardSpeed" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "standardSpeed" element captures the
           ↪ speed at which the laser moves across the part when the
           ↪ laser is above the corridor defined by the comparison
           ↪ contour.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="onPartSpeed" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "onPartSpeed" element captures the speed
           ↪  at which the laser moves across the part when the laser
           ↪ is inside the corridor defined by the comparison contour
           ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="downskinSpeed" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "downskinSpeed" element captures the
           ↪ speed at which the laser moves across the part when the
           ↪ laser is outside the corridor defined by the comparison
           ↪ contour.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="standardPower" type="AnnotatedFloatType">
```

```
      <xs:annotation>
        <xs:documentation>The "standardPower" element captures the
           ↪ power of the laser during the contour exposure, when the
           ↪ laser moves above the corridor defined by the comparison
           ↪ contour.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="onPartPower" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "onPartPower" element TBD </xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="downskinPower" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "downskinPower" element TBD </xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="thickness" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Defines where the comparison contour is to
           ↪ be searched for under the current layer. In this way the
           ↪ onPart area or the Downskin area can be expanded to cover
           ↪  several layers.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="corridor" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Width of the corridor around a comparison
           ↪ contour. The comparison contour is the contour around the
           ↪  layer that lies below the current contour by the value
           ↪ given in "thickness". Dependng on whether the current
           ↪ contour line is inside, above or outside the corridor, it
           ↪  is exposed using the
selected OnPart, Standard, or Downskin parameters.</xs:documentation>
      </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="edgeSettingsType">
    <xs:sequence>
      <xs:element name="edges" type="OnOffEnum">
        <xs:annotation>
          <xs:documentation>The "edges" element captures whether points
             ↪ are exposed or not.</xs:documentation>
        </xs:annotation>
```

```
    </xs:element>
    <xs:element name="postEdge" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation>The "postEdge" element captures whether the
           ↪ points are exposed after the exposure of the inner
           ↪ hatching.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="edgeFactor" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation> The "edgeFactor" element captures the level
           ↪  of exposure for points starting from the outermost
           ↪ points on the nominal contour to create fine
features. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="threshold" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The "threshold" element captures the
           ↪ threshold value. If the distance from the actual contour
           ↪ to the nominal contour at a point exceeds the value
           ↪ threshold x beam offset, this point is exposed with an
           ↪ edge factor of 1.45.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="minimumRadiusFactor" type="xs:float" default
       ↪ ="0">
      <xs:annotation>
        <xs:documentation>The "minimumRadiusFactor" element captures
           ↪ the level of exposure of the points as a function of the
           ↪ radius of the laser beam.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="beamOffset" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "beamOffset" element captures the offset
           ↪  of the start point for the exposure of the points from
           ↪ the inside of the part. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="speed" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "speed" element captures the speed at
           ↪ which the laser
moves across the part. </xs:documentation>
```

```
      </xs:annotation>
    </xs:element>
    <xs:element name="laserPower" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The "laserPower" element captures the power
           ↪ of the laser
during this part of the exposure. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="skipLayerSettingsType">
  <xs:sequence>
    <xs:element name="skippedLayers" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The "skippedLayer" element captures the
           ↪ number of layers skipped, i.e. not exposed.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="offsetLayers" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>Offset upwards of the layers to be exposed
           ↪ and not to be exposed.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="exposefirstLayer" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation>The "exposefirstLayer" element captures
           ↪ whether the first layer was exposed after the start of
           ↪ the building process or not. ON means the first layer was
           ↪  exposed after the start of the building process. OFF
           ↪ means the first layer was not exposed after the start of
           ↪ the building process. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="chessCategorySettingsType">
  <xs:sequence>
    <xs:element name="squareDistance" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "squareDistance" element captures the
           ↪ distance between the hatch lines within the squares. </xs
           ↪ :documentation>
      </xs:annotation>
    </xs:element>
```

```
<xs:element name="squareSpeed" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "squareSpeed" element captures the
      ↪ speed of the laser in the squares. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="squarePower" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "squarePower" element captures the
      ↪ laser power in the squares.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="squareWidth" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "squareWidth" element captures the
      ↪ dimension of the squares.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="gapDistance" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "gapDistance" element captures the
      ↪ distance between the hatch lines within the squares. </xs
      ↪ :documentation>
  </xs:annotation>
</xs:element>
<xs:element name="gapSpeed" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "gapSpeed" element captures the speed
      ↪ of the laser in the gaps.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="gapPower" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "gapPower" element captures the laser
      ↪ power in the gaps.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="gapWidth" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "gapWidth" element captures the width
      ↪ of the gaps between the squares. </xs:documentation>
  </xs:annotation>
</xs:element>
```

```
<xs:element name="overlap" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "overlap" element captures the
      ↪ overlapping of the squares with the gaps. </xs:
      ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="beamOffset" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "beamOffset" element captures the start
      ↪  point for the exposure of the points from the inside of
      ↪ the part. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="hatchingX" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "hatchingX" element captures whether
      ↪ there was hatching in the X direction or not. ON means
      ↪ that hatching in X took place. OFF means that hatching in
      ↪  X did not take place. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="hatchingY" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "hatchingY" element captures whether
      ↪ there was hatching in the Y direction or not. ON means
      ↪ that hatching in Y took place. OFF means that hatching in
      ↪  Y did not take place. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="alternating" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "alternating" element captures whether
      ↪ it changed the direction of the exposure from layer to
      ↪ layer or not. ON means that the direction changed between
      ↪  layers. OFF means it did not change. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="rotated" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "rotated" element captures whether the
      ↪ angle between two layers changed or not. ON means that
      ↪ the angle changed between layers. OFF means it did not
      ↪ change. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="rotatedAngle" type="AnnotatedFloatType">
```

```
      <xs:annotation>
        <xs:documentation> The "rotatedAngle" element captures the
           ↪ rotation angle.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="skywriting" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation> The "skywriting" element captures whether
           ↪ this process was selected in the software or not. During
           ↪ skywriting, the acceleration phase and the retardation
           ↪ phase for the laser focus are outside the exposure area.
           ↪ The laser is switched off during this phase. ON means
           ↪ skywriting was selected. OFF means skywriting was not
           ↪ selected.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="offset" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation> The "offset" element means that, it offsets
           ↪  the stripes in each layer by one half of the stripe
           ↪ width. ON means that offset was selected. OFF means that
           ↪ offset was not selected. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="upDownCategorySettingsType">
  <xs:sequence>
    <xs:element name="distanceUp" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "distanceUp" element captures the
           ↪ distance between hatchlines in the upSkin. </xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="distanceDown" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "distanceDown" element captures the
           ↪ distance between hatchlines in the downSkin. </xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="speedUp" type="AnnotatedFloatType">
      <xs:annotation>
```

```
    <xs:documentation> The "speedUp" element captures the laser
       ↪ speed in the upSkin.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="speedDown" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "speedDown" element captures the laser
       ↪ speed in the downSkin.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="powerUp" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "powerUp" element captures the power of
       ↪  the laser in the upSkin.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="powerDown" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "powerDown" element captures the power
       ↪ of the laser in the downSkin. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="thicknessUp" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "thicknessUp" element captures the
       ↪ thickness of the upSkin areas.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="thicknessDown" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "thicknessDown" element captures how
       ↪ thick the downSkin areas are.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="overlapWithInSkin" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "overlapWithInSkin" element captures
       ↪ the overlapping of upSkin/downSkin with inSkin. </xs:
       ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="minLength" type="AnnotatedFloatType">
```

```
  <xs:annotation>
    <xs:documentation> The "minLength" element captures the
       ↪ minimum length of the upSkin/downSkin hatch lines. </xs:
       ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="xUp" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation>!The "xUp" element captures whether there
       ↪ hatching in positive X direction or not. ON means there
       ↪ was hatching in the positive X direction. OFF means there
       ↪  was not. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="xDown" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "xDown" element captures whether there
       ↪ hatching in the negative X direction or not. ON means
       ↪ there was hatching in the negative X direction. OFF means
       ↪  there was not. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="yUp" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "yUp" element captures whether there
       ↪ was hatching in the positive Y direction. ON means there
       ↪ was hatching in the positive Y direction. OFF means there
       ↪  was not. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="yDown" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "yDown" element captures whether there
       ↪ was hatching in the negative Y direction. ON means there
       ↪ was hatching in the negative Y direction. OFF means there
       ↪  was not. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="alternateUp" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "alternateUp" element captures whether
       ↪ there were changes in the direction of the exposure from
       ↪ layer to layer in upSkin. ON means there were changes in
       ↪ direction between layers. OFF means the direction did not
       ↪  change between layers.
    </xs:documentation>
  </xs:annotation>
```

```
    </xs:element>
    <xs:element name="alternateDown" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation> The "alternateDown" element captures
          ↪ whether there were changes in the direction of the
          ↪ exposure from layer to layer in downSkin. ON means there
          ↪ were changes in direction between layers. OFF means the
          ↪ direction did not change between layers.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="skywriting" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation> The "skywriting" element captures whether
          ↪ this process was selected in the software or not. During
          ↪ skywriting, the acceleration phase and the retardation
          ↪ phase for the laser focus are outside the exposure area.
          ↪ The laser is switched off during this phase. ON means
          ↪ skywriting was selected. OFF means skywriting was not
          ↪ selected.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="stripesCategorySettingsType">
  <xs:sequence>
    <xs:element name="hatchDistance" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "hatchDistance" element captures the
          ↪ distance between the hatch lines within stripes. </xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="speed" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "speed" element captures the exposure
          ↪ speed for hatching the stripes. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="laserPower" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "laserPower" element captures laser
          ↪ power for hatching the stripes.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
```

```
<xs:element name="beamOffset" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "beamOffset" element captures the start
      ↪  point for the exposure of the points from the inside of
      ↪ the part. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="stripeWidth" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation> The "stripeWidth" element captures the
      ↪ width of the stripes.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="stripesOverlap" type="AnnotatedFloatType" form
   ↪ ="qualified">
  <xs:annotation>
    <xs:documentation> The "stripesOverlap" element captures the
      ↪ width of the overlap between two stripes. </xs:
      ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="skywriting" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "skywriting" element captures whether
      ↪ this process was selected in the software or not. During
      ↪ skywriting, the acceleration phase and the retardation
      ↪ phase for the laser focus are outside the exposure area.
      ↪ The laser is switched off during this phase. ON means
      ↪ skywriting was selected. OFF means skywriting was not
      ↪ selected.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="offset" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "offset" element means that, it offsets
      ↪  the stripes in each layer by one half of the stripe
      ↪ width. ON means that offset was selected. OFF means that
      ↪ offset was not selected. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="hatchingX" type="OnOffEnum">
  <xs:annotation>
    <xs:documentation> The "hatchingX" element captures whether
      ↪ hatching in the X direction took place or not. ON means
      ↪ that hatching in X took place. OFF means that hatching in
```

```
        ↪  X did not take place. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="hatchingY" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation> The "hatchingY" element captures whether
          ↪ hatching in the Y direction took place or not. ON means
          ↪ that hatching in Y took place. OFF means that hatching in
          ↪  Y did not take place. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="alternating" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation>The "alternating" element captures whether
          ↪ the direction of the exposure changed from layer to layer
          ↪  or not. ON means that the direction changed between
          ↪ layers. OFF means it did not change. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="rotated" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation> The "rotated" element captures whether the
          ↪ angle between two layers changed or not. ON means that
          ↪ the angle changed between layers. OFF means it did not
          ↪ change. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="rotatedAngle" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "rotatedAngle" element captures the
          ↪ rotation angle between two layers. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="skinCoreSettingsType">
  <xs:sequence>
    <xs:element name="exposureTypeSelections" type="
      ↪ exposureTypeSelectionsType">
      <xs:annotation>
        <xs:documentation>The "exposureTypeSelections" element
          ↪ captures the user's choice for what Exposure Type (ET)
          ↪ they want to use for each of the four phases of exposure
          ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="skinThicknessXY" type="AnnotatedFloatType">
```

```
      <xs:annotation>
        <xs:documentation> The "skinThicknessXY" element captures the
           ↪ thickness (Outer boundary layer in X/Y direction) at
           ↪ which the skin is exposed using the selected exposure
           ↪ type.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="skinThicknessZ" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "skinThicknessZ" element captures the
           ↪ thickness (outer boundary layer in Z direction) at which
           ↪ the skin is exposed using the selected exposure type.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="baseRadius" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation> The "baseRadius" element captures the
           ↪ radius that is added to the outer most contour of the
           ↪ part to obtain better adhesion to the base plate.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="coreOpenToPlatform" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation>The "coreOpenToPlatform" element captures
           ↪ whether a skin is exposed on the underside of the part or
           ↪  not. ON means a skin is exposed on the underside of the
           ↪ part. OFF means no skin is exposed on the underside of
           ↪ the part. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="skinCoreExposure" type="OnOffEnum">
      <xs:annotation>
        <xs:documentation>If ON, the part is exposed as per the
           ↪ parameters selected.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SLIHatchSettingsType">
  <xs:sequence>
    <xs:element name="speed" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Defines the desired exposure speed to be
           ↪ used when applying the aadditional exposure requirements
```

```
        ↪ delineated in the external SLI file(s).</xs:documentation
        ↪ >
      </xs:annotation>
    </xs:element>
    <xs:element name="power" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Defines the desired laser power to be used
        ↪ when applying the aadditional exposure requirements
        ↪ delineated in the external SLI file(s).</xs:documentation
        ↪ >
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="exposureTypeSelectionsType">
  <xs:sequence>
    <xs:element name="preExposureSelection" type="
      ↪ exposureSelectionType">
      <xs:annotation>
        <xs:documentation>The "preExposure" element captures the
        ↪ settings for the machine for the first exposing of a
        ↪ layer by the laser.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="skinExposureSelection" type="
      ↪ exposureSelectionType">
      <xs:annotation>
        <xs:documentation>The "skinExposure" element captures the
        ↪ values for the calculation of skin for a part and define
        ↪ exposure types for its exposure.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="coreExposureSelection" type="
      ↪ exposureSelectionType">
      <xs:annotation>
        <xs:documentation>The "coreExposure" element captures the
        ↪ values for the calculation of core for a part and define
        ↪ exposure types for its exposure.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="postExposureSelection" type="
      ↪ exposureSelectionType">
      <xs:annotation>
        <xs:documentation>The "postExposure" captures the the exposure
        ↪  settings for laser exposure performed after the skin and
        ↪  core exposure phases.</xs:documentation>
      </xs:annotation>
```

```
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="strategyParametersType">
  <xs:sequence>
    <xs:element name="hatchPatternStrategy">
      <xs:annotation>
        <xs:documentation>This gives you a choice of Hatch pattern for
          ↪  your component. Each Hatch pattern has advantages and
          ↪ disadvantages, depending upon the characteristics of the
          ↪ component being built.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:choice>
          <xs:element name="Meander">
            <xs:annotation>
              <xs:documentation>Meander - the single pass meander
                ↪ pattern is a zigzag which breaks on a hole. The
                ↪ hatch is bordered by 2 offset passes.</xs:
                ↪ documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="Chessboard">
            <xs:annotation>
              <xs:documentation>Chessboard - the chessboard pattern is a
                ↪  square block based strategy where scan lines in one
                ↪  block are perpendicular to scan lines in adjacent
                ↪ blocks.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="Stripe">
            <xs:annotation>
              <xs:documentation>Stripe - the stripe strategy is similar
                ↪ to the single pass meander. Scanning is done based
                ↪ on fixed length broken stripes.</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="fieldSize" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The field Size is the size of each area in a
          ↪  chessboard Hatch pattern.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="fieldOffset" type="AnnotatedFloatType">
```

```
  <xs:annotation>
    <xs:documentation>field Offset is used to overlap the fields,
       ↪ to prevent porosity in the component being built.</xs:
       ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="minimalfieldSize" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>The Minimal field Size is the smallest size
       ↪ a field can be in a chessboard Hatch pattern.</xs:
       ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="stripeSize" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>In a stripe Hatch pattern each layer is cut
       ↪ into strips at a user defined width, the Strip size is
       ↪ commonly 5 mm. This allows constant time between each
       ↪ successive strip and therefore maintains a more
       ↪ consistent temperature throughout.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="stripeOffset" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>A Stripe offset is used to overlap the
       ↪ Stripes, to prevent porosity. Use a negative offset for
       ↪ an overlap.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="mergeVectorLength" type="AnnotatedFloatType">
  <xs:annotation>
    <xs:documentation>Minimum length that a vector can have before
       ↪  it is merged with the co-linear vector of the previous
       ↪ stripe.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="mergeVector" type="xs:boolean">
  <xs:annotation>
    <xs:documentation>This enables you to toggle between merge
       ↪ vector on and off. Toggle 1 for On and 0 for Off.</xs:
       ↪ documentation>
  </xs:annotation>
</xs:element>
<xs:element name="blockSortOptimized" type="xs:boolean">
  <xs:annotation>
    <xs:documentation>Laser path will move in a continuous
       ↪ movement around holes. It will then return and fill in
```

```
        ↪ any missing gaps. Due to fewer jump delays, this is a
        ↪ faster strategy. There will however be cold weld lines
        ↪ where the remaining jumps occur. Toggle 1 for On and 0
        ↪ for Off.</xs:documentation>
     </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="controlParametersType">
  <xs:sequence>
    <xs:element name="totalfill" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>fills the layer with only Additional border
        ↪ lines and no Hatch fill is used. Toggle 1 for On and 0
        ↪ for Off.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="volumefillContours" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>fill contours are similar to borders. The
        ↪ purpose of fill contours are to reduce porosity and
        ↪ strengthen the bond between the fill hatching and the
        ↪ borders Toggle 1 for On and 0 for Off.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="volumeHatch" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Volume hatch is the hatching that forms the
        ↪ vast majority of a layers area within a component. Toggle
        ↪  1 for On and 0 for Off.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="volumeJumpOptimization" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Laser path will move in a continuous
        ↪ movement around holes. It will then return and fill in
        ↪ any missing gaps. Due to fewer jump delays, this is a
        ↪ faster strategy. There will however be cold weld lines
        ↪ where the remaining jumps occur. Toggle 1 for On and 0
        ↪ for Off.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="upskinEnable" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Without Upskin, track patterns are visible
        ↪ on the top surface of each part. Upskin re-scans the top
        ↪ surface of a build in such a way as to remove the marks
```

```
        ↪ left by scan strategies. Toggle 1 for On and 0 for Off.</
        ↪ xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="downskinEnable" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Without Downskin energy penetrates into
        ↪ layers below and can create a poor surface finish.
        ↪ Downskin reduces the energy, or increases speed at
        ↪ overhangs to reduce penetration and give a better surface
        ↪  finish. Toggle 1 for On and 0 for Off.</xs:documentation
        ↪ >
      </xs:annotation>
    </xs:element>
    <xs:element name="upskinBorderEnable" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>This enables you to chose whether you have a
        ↪  border or not on your Upskin layers. Toggle 1 for On and
        ↪  0 for Off.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="blockedPath" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Blocked Paths is used to reduce the number
        ↪ of scans in thin areas to a single scan Toggle 1 for On
        ↪ and 0 for Off.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="orderParametersType">
  <xs:sequence>
    <xs:element name="scanOrder" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>This setting gives the ability to change the
        ↪  scan order of each type of hatch.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="volumeBorderInOut" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Where you have multiple borders you can
        ↪ chose to scan the border from the inside to out or from
        ↪ the outside to in. Toggle 1 for InOut and 0 for OutIn</xs
        ↪ :documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="volumefillContourInOut" type="xs:boolean">
```

```
      <xs:annotation>
        <xs:documentation>Where you have multiple fill contours you
           ↪ can chose to scan the fill contour from the inside to out
           ↪  or from the outside to in. Toggle 1 for InOut and 0 for
           ↪ OutIn</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="generalVolumeParametersType">
  <xs:sequence>
    <xs:element name="beamCompensation" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Used to offset the laser from the edge of
           ↪ the STL boundary, similar to how a cutting tool would
           ↪ have a radius offset from the edge of a part in
           ↪ traditional machining.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="borderCount" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The number of borders. Borders are the scans
           ↪  that outline the 2D areas of each layer. They help
           ↪ improve the surface structure and surface finish.</xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="borderDistance" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The distance between each Border.</xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="fillContourOffset" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The distance the fill contour is placed from
           ↪  the outermost
Border.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="numberOffillContours" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The number of fill Contours to be generated.
           ↪  fill Contours will be generated inwards and will overlap
           ↪  with the fill hatch.</xs:documentation>
      </xs:annotation>
    </xs:element>
```

```
    <xs:element name="fillContourDistance" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The distance between each fill contour.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="hatchDistance" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Hatch lines consist of point exposures. The
          ↪ distance between each line vector of point exposures is
          ↪ called the
Hatch Distance.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="hatchOffset" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The distance between the fill Hatch boundary
          ↪  and the innermost border.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="hatchStartAngle" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The angle at which the hatch starts.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="hatchIncrementAngle" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The angle between layers of hatch.</xs:
          ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="filterLength" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Vectors smaller than this threshold will not
          ↪  be generated. There is no need to generate vectors that
          ↪ are a fraction of the spot size.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="blockedPathResolution" type="AnnotatedFloatType
      ↪ ">
      <xs:annotation>
        <xs:documentation>Controls the smoothness of the blocked path
          ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
```

```
    <xs:element name="blockedPathTrimDistance" type="
       ↪ AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Controls triming of the blocked path,
           ↪ measured in mm.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="blockedPathfilterLength" type="
       ↪ AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Controls removal of tiny segments, measured
           ↪ in mm.</xs:documentation>
      </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="generalUpskinParametersType">
   <xs:sequence>
    <xs:element name="useStartAngle" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Starts scan direction at user specified
           ↪ angle or continue
from last angle of volume hatch. Toggle 1 for Yes and 0 for No</xs:
   ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="keepAdditionalBorders" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>Enables you to keep additional upskin
           ↪ borders. Toggle 1 for Yes and 0 for No</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="borderOffset" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The distance between upskin border and
           ↪ upskin hatches.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="numberOfExposures" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The number of times for which upskin is
           ↪ scanned.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="numberOfLayers" type="xs:unsignedInt">
      <xs:annotation>
```

```
        <xs:documentation>The number of layers on which upskin
           ↪ hatching can be
applied.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="skinAreaTolerance" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The minimal area width on which upskin will
           ↪ be applied.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="hatchOffset" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The offset between hatch vectors and
           ↪ innermost borders (negative value indicates an overlap)
           ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="hatchDistance" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The distance between hatch vectors.</xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="filterLength" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The filter Length is used to filter out
           ↪ vectors which are
shorter than this defined threshold.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="rotationIncrementAngle" type="
       ↪ AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Angle between the scan lines from one layer
           ↪ to the next.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="rotationStartAngle" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>The angle is used to rotate the skin fill
           ↪ hatches.</xs:documentation>
      </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="generalDownskinParametersType">
```

```
<xs:sequence>
 <xs:element name="useStartAngle" type="xs:boolean">
  <xs:annotation>
   <xs:documentation>Starts scan direction at user specified
    ↪ angle or continue from last angle of volume hatch. Toggle
    ↪  1 for Yes and 0 for No</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="offsetVolumeArea" type="AnnotatedFloatType">
  <xs:annotation>
   <xs:documentation>An overlap or a gap, between skin and volume
    ↪  area, measured in mm.</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="numberOfExposures" type="xs:unsignedInt">
  <xs:annotation>
   <xs:documentation>The number of times for which downskin is
    ↪ scanned.</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="numberOfLayers" type="xs:unsignedInt">
  <xs:annotation>
   <xs:documentation>The number of layers on which downskin
    ↪ hatching can be applied.</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="skinAreaTolerance" type="AnnotatedFloatType">
  <xs:annotation>
   <xs:documentation>The minimal area width on which downskin
    ↪ will be applied.</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="hatchOffset" type="AnnotatedFloatType">
  <xs:annotation>
   <xs:documentation>The offset between hatch vectors and
    ↪ innermost borders (negative value indicates an overlap)
    ↪ .</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="hatchDistance" type="AnnotatedFloatType">
  <xs:annotation>
   <xs:documentation>The distance between hatch vectors.</xs:
    ↪ documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="filterLength" type="AnnotatedFloatType">
  <xs:annotation>
```

```
      <xs:documentation>The filter Length is used to filter out
         ↪ vectors which are shorter than this defined threshold.</
         ↪ xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="rotationIncrementAngle" type="
       ↪ AnnotatedFloatType">
     <xs:annotation>
      <xs:documentation>Angle between the scan lines from one layer
         ↪ to the next.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="rotationStartAngle" type="AnnotatedFloatType">
     <xs:annotation>
      <xs:documentation>The angle is used to rotate the skin fill
         ↪ hatches.</xs:documentation>
     </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="recoaterSettingsType">
  <xs:choice>
    <xs:element name="eosRecoaterSettings">
     <xs:annotation>
      <xs:documentation>Captures the recoater settings as described
         ↪ by EOS systems in their build files.</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="recoaterBlade" type="xs:string" default="
          ↪ text">
        <xs:annotation>
         <xs:documentation> The "recoaterBlade" element captures
            ↪ the type of recoater blade used.
      </xs:documentation>
         </xs:annotation>
       </xs:element>
       <xs:element name="feedCharge" type="AnnotatedFloatType">
        <xs:annotation>
         <xs:documentation> The "feedCharge" element captures the
            ↪ percentage at which feedbed was moved up relative to
            ↪  how much the build bed was lowered. It is a float
            ↪ type.
      </xs:documentation>
         </xs:annotation>
       </xs:element>
```

```
        <xs:element name="positiveXRecoaterSpeed" type="xs:float"
          ↪ default="0">
          <xs:annotation>
            <xs:documentation> The "positiveXRecoaterSpeed" element
              ↪ captures the speed of the recoater blade at which it
              ↪  moves to the positive X axis. It is a float type.
              ↪ </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="negativeXRecoaterSpeed" type="xs:float"
          ↪ default="0">
          <xs:annotation>
            <xs:documentation> The "negativeXRecoaterSpeed" element
              ↪ captures the speed of the recoater blade at which it
              ↪  moves to the negative X axis. It is a float type.
              ↪ </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="contactFreeOutwardTravel" type="OnOffEnum">
          <xs:annotation>
            <xs:documentation> The "contactFreeOutwardTravel" element
              ↪ captures whether the platform was lowered a little
              ↪ prior to movement to the right so that the recoater
              ↪ blade does not get damaged. ON means the platform
              ↪ was lowered. OFF means the platform was not lowered.
      </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="renishawRecoaterSettings">
      <xs:annotation>
        <xs:documentation>Captures the recoater settings as described
          ↪ by Renishaw systems in their build files. Note that it is
          ↪  empty, since there is no mention of recoater settings in
          ↪  their process plans.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:choice>
</xs:complexType>
<xs:complexType name="miscellaneousSettingsType">
  <xs:sequence>
    <xs:element name="language" type="xs:string">
      <xs:annotation>
        <xs:documentation>Captures the language used by the system.</
          ↪ xs:documentation>
```

```
      </xs:annotation>
    </xs:element>
    <xs:element name="currency" type="xs:string">
      <xs:annotation>
        <xs:documentation>Captures the currency used by the system.</
          ↪ xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="partDataType">
  <xs:sequence>
    <xs:element name="importedModel" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The "importedModel" element
is a set of 1 to many models,
representing their positions,
scaling, multiplicity, etc.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="partID" type="xs:string">
            <xs:annotation>
              <xs:documentation>The "partID" element captures the part
                ↪ ID of the part that was produced
with the parameters selected.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="partDescription" type="xs:string">
            <xs:annotation>
              <xs:documentation>Captures a textual description of the
                ↪ imported model.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="partScalingFactors" type="
            ↪ partScalingFactorsType">
            <xs:annotation>
              <xs:documentation> The "partScalingFactor" element
                ↪ captures how much bigger or smaller the part needed
                ↪ to be than the nominal value. </xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="partOrientation" type="partOrientationType
            ↪ ">
            <xs:annotation>
              <xs:documentation>The partOrientation element describes
                ↪ the 3D orientation of a part using Euler angles,
```

```
                ↪ specifically, the Tait-Bryan z-y'-x'' set of
                ↪ rotations.
</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="boundingBox" type="boundingBoxType">
          <xs:annotation>
            <xs:documentation>The boundingBox element contains the
               ↪ dimensions of the smallest rectangular box that
               ↪ contains the imported model.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="healingParameters" type="
           ↪ healingParametersType" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>The optional healingParameters element
               ↪ represents the usage of healing strategies for
               ↪ imported geometry repair.
</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="instancingSettings" type="
           ↪ instancingSettingsType">
          <xs:annotation>
            <xs:documentation>Captures information related to the
               ↪ patterning of the part.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="material" type="xs:string">
    <xs:annotation>
      <xs:documentation>The "material" element describes the
         ↪ material being used to print all the parts in the current
         ↪  process plan.</xs:documentation>
    </xs:annotation>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="partScalingFactorsType">
 <xs:sequence>
  <xs:element name="scaleInX" type="xs:float" default="0">
    <xs:annotation>
      <xs:documentation>The "scaleInX" element captures the part
         ↪ dependent scaling values for all instances of the
         ↪ imported model in the X direction.</xs:documentation>
```

```
      </xs:annotation>
    </xs:element>
    <xs:element name="scaleInY" type="xs:float" default="0">
      <xs:annotation>
        <xs:documentation>The "scaleInY" element captures the part
          ↪ dependent scaling values for all instances of the
          ↪ imported model in the Y direction.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="scaleInZ" type="xs:float" default="0">
      <xs:annotation>
        <xs:documentation>The "scaleInZ" element captures the part
          ↪ dependent scaling values for all instances of the
          ↪ imported model in the Z direction.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="partOrientationType">
  <xs:sequence>
    <xs:element name="psi" type="xs:double">
      <xs:annotation>
        <xs:documentation>first orientation angle.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="theta" type="xs:double">
      <xs:annotation>
        <xs:documentation>Second orientation angle.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="phi" type="xs:double">
      <xs:annotation>
        <xs:documentation>Third orientation angle.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="orientationConvention" type="xs:token" default
    ↪ ="Tait-Bryan z-y'-x''">
    <xs:annotation>
      <xs:documentation>The "orientationConvention" attribute serves
        ↪ to capture how the three angles (psi, theta, phi) are to
        ↪ be interpreted to obtain the desired orientation. For
        ↪ example, Tait-Bryan z-y'-x'' is a common orientation
        ↪ convention.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
```

```
<xs:complexType name="boundingBoxType">
  <xs:sequence>
    <xs:element name="sizeInX" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Size of the part bounding box in the x
          ↪ direction, expressed as an annotated floating point
          ↪ number.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sizeInY" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Size of the part bounding box in the y
          ↪ direction, expressed as an annotated floating point
          ↪ number.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sizeInZ" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Size of the part bounding box in the z
          ↪ direction, expressed as an annotated floating point
          ↪ number.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="healingParametersType">
  <xs:sequence>
    <xs:element name="isHealingEnabled" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>The "isHealingEnabled" element captures the
          ↪ boolean nature of whether any healing was performed on
          ↪ the part during import.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="healingStrategy" type="xs:string">
      <xs:annotation>
        <xs:documentation>The "healingStrategy" element captures the
          ↪ name of the healing strategy used to fix any errors
          ↪ present in the model during import into the Build
          ↪ Preparation Software.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="instancingSettingsType">
  <xs:sequence>
    <xs:element name="isInstancingEnabled" type="xs:boolean">
```

```
      <xs:annotation>
        <xs:documentation>Captures whether the part is patterned or
          ↪ not.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="partLayout" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The "partLayout" element describes the
          ↪ patterns of parts on the build plate.
</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="anchorPartLocation" type="
            ↪ anchorPartLocationType">
            <xs:annotation>
              <xs:documentation>Captures the location of the "origin" of
                ↪  the pattern of parts, called the "anchor".</xs:
                ↪ documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="patternInX" type="patternInXType">
            <xs:annotation>
              <xs:documentation>Captures information related to the part
                ↪  instancing in the x direction.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="patternInY" type="patternInYType">
            <xs:annotation>
              <xs:documentation>Captures information related to the part
                ↪  instancing in the y direction.</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="anchorPartLocationType">
  <xs:sequence>
    <xs:element name="locationInX" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the location of the anchor in the x
          ↪  direction.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="locationInY" type="AnnotatedFloatType">
```

```
      <xs:annotation>
        <xs:documentation>Captures the location of the anchor in the y
          ↪ direction.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="locationInZ" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the location of the anchor in the z
          ↪ direction.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="patternInXType">
  <xs:sequence>
    <xs:element name="spacingInX" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the distance between instances of
          ↪ the part in x.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="copiesTowardsPositiveX" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The number of part copies in the positive x
          ↪ direction, as an unsigned integer.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="copiesTowardsNegativeX" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The number of part copies in the negative x
          ↪ direction, as an unsigned integer.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="patternInYType">
  <xs:sequence>
    <xs:element name="spacingInY" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the distance between instances of
          ↪ the part in y.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="copiesTowardsPositiveY" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The number of part copies in the positive y
          ↪ direction, as an unsigned integer.</xs:documentation>
```

```
      </xs:annotation>
    </xs:element>
    <xs:element name="copiesTowardsNegativeY" type="xs:unsignedInt">
      <xs:annotation>
        <xs:documentation>The number of part copies in the negative y
          ↪ direction, as an unsigned integer.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="buildSupportsDataType">
  <xs:sequence>
    <xs:element name="support" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The conceptual support element represents an
          ↪  instance of a support. There can be 0 to many of these
          ↪ supports.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="supportStartLocation" type="
            ↪ supportStartLocationType">
            <xs:annotation>
              <xs:documentation>Captures the location of the root of the
                ↪  support. The z location can be greater than zero (
                ↪ above the build plate) if the support is on the part
                ↪  itself and supports another section of tthe part.</
                ↪ xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="height" type="AnnotatedFloatType">
            <xs:annotation>
              <xs:documentation>Captures the total height of the support
                ↪ . If the support is a simple vertical 'stick', then
                ↪ this would be the length.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="supportStyle" type="supportStyleType">
            <xs:annotation>
              <xs:documentation>Captures the style of support, that is,
                ↪ the information related to its geometry and
                ↪ generation.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="supportOrigin" type="SupportOriginEnum">
            <xs:annotation>
```

```
            <xs:documentation>The "supportOrigin" element captures
               ↪ whether the support originates on the buildplate or
               ↪ on the part with the aid of an enumeration.</xs:
               ↪ documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="supportArea" type="SupportAreaEnum">
            <xs:annotation>
              <xs:documentation>The "supportArea" element captures
                 ↪ whether the support originates on an open area (such
                 ↪  as the buildplate or a face of the part) or an edge
                 ↪  of the part, with the aid of an enumeration.</xs:
                 ↪ documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="supportStartLocationType">
  <xs:sequence>
    <xs:element name="locationInX" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the x location of the support root
           ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="locationInY" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the y location of the support root
           ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="locationInZ" type="AnnotatedFloatType">
      <xs:annotation>
        <xs:documentation>Captures the z location of the support root
           ↪ .</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="supportStyleType">
  <xs:choice>
    <xs:element name="renishawSupportStyle">
      <xs:annotation>
```

```
      <xs:documentation>Captures the information related to the
         ↪ support style as viewed by the Renishaw system.</xs:
         ↪ documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="supportStyleName" type="xs:string">
            <xs:annotation>
              <xs:documentation>Captures the textual name of the support
                 ↪  style.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="pointSupportSettings" type="
             ↪ pointSupportSettingsType">
            <xs:annotation>
              <xs:documentation>Captures settings about point-type
                 ↪ supports.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="clusterSupportSettings" type="
             ↪ clusterSupportSettingsType">
            <xs:annotation>
              <xs:documentation>Captures settings about clusters of
                 ↪ supports.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="supportTypeSettings" type="
             ↪ supportTypeSettingsType">
            <xs:annotation>
              <xs:documentation>Captures general settings about the
                 ↪ supports.</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="eosSupportStyle">
      <xs:annotation>
        <xs:documentation>Captures the information related to the
           ↪ support style as viewed by the EOS system. Note that it
           ↪ is empty, since there is no such information described by
           ↪  the EOS process plan.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:choice>
</xs:complexType>
<xs:complexType name="pointSupportSettingsType">
```

```
   <xs:sequence>
     <xs:element name="diameter" type="xs:float">
       <xs:annotation>
         <xs:documentation>the diameter of the support base</xs:
            ↪ documentation>
       </xs:annotation>
     </xs:element>
     <xs:element name="criticalAngle">
       <xs:annotation>
         <xs:documentation>the maximum angle between the build plate
            ↪ and the surface of model when
support is required</xs:documentation>
       </xs:annotation>
       <xs:simpleType>
         <xs:restriction base="xs:float">
           <xs:minInclusive value="0"/>
           <xs:maxInclusive value="90"/>
         </xs:restriction>
       </xs:simpleType>
     </xs:element>
     <xs:element name="capDiameter" type="xs:float">
       <xs:annotation>
         <xs:documentation>the diameter of the cap where the support is
            ↪  in contact with the model</xs:documentation>
       </xs:annotation>
     </xs:element>
     <xs:element name="capHeight" type="xs:float">
       <xs:annotation>
         <xs:documentation>the height of the cap (contact region) of
            ↪ the support</xs:documentation>
       </xs:annotation>
     </xs:element>
     <xs:element name="conicalAngle">
       <xs:annotation>
         <xs:documentation>the angle of the cone formed between the
            ↪ base and cap of the support</xs:documentation>
       </xs:annotation>
       <xs:simpleType>
         <xs:restriction base="xs:float">
           <xs:minInclusive value="0"/>
           <xs:maxInclusive value="90"/>
         </xs:restriction>
       </xs:simpleType>
     </xs:element>
     <xs:element name="crossSection" type="CrossSectionShapeEnum">
       <xs:annotation>
```

```
      <xs:documentation>the profile shape of the support, a choice
        ↪ from: square, circle, or hexagon</xs:documentation>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="clusterSupportSettingsType">
  <xs:sequence>
   <xs:element name="clusterSpacing" type="xs:float">
    <xs:annotation>
     <xs:documentation>the distance between clusters of supports</
       ↪ xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="supportCount" type="xs:unsignedInt">
    <xs:annotation>
     <xs:documentation>the number of supports per cluster</xs:
       ↪ documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="supportSpacing" type="xs:float">
    <xs:annotation>
     <xs:documentation>the distance between the supports within the
       ↪  cluster</xs:documentation>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="supportTypeSettingsType">
  <xs:sequence>
   <xs:element name="autoSupportAreaSpacing" type="
     ↪ AnnotatedFloatType">
    <xs:annotation>
     <xs:documentation>The distance between supports or support
       ↪ clusters.</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="autoSupportEdgeOffset" type="AnnotatedFloatType
     ↪ ">
    <xs:annotation>
     <xs:documentation>The offset from the edge of the model to the
       ↪  support or support cluster.</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="lineSupportEquidistant" type="OnOffEnum">
    <xs:annotation>
```

```
        <xs:documentation>Whether equal distance between supports is
           ↪ On/Off. If equidistant support is On, supports will be
           ↪ created at equidistant points along the line.</xs:
           ↪ documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```