Trusted Services for Cyber Manufacturing Systems

André, Pascal and Cardin, Olivier

Abstract In the context of Industry 4.0, (cyber) manufacturing systems enters in a world of services which is a convenient paradigm to match virtual and physical systems covering Cloud computing, big data, Internet-of-Things (IoT) and mobility. The coordination and control of such complex system by the way of actors or services requires methods and techniques to design, verify and deploy the services, possibly on the fly making service engineering an unavoidable approach to develop new generation cyber manufacturing systems. In this position paper, we advocate a service based component model that would be helpful to reach this goal.

1 Introduction

Following the Web revolution, that enabled to connect non only people but also systems through unified communication protocols, a new step, called Industry 4.0, is engaged that interleave computation processes and physical processes in the recent so-called field cyber-physical systems (CPS) which defines a new and multidisciplinary that cover many engineering areas such as mechatronics (from electrical or mechanical), robotics... [22]. This new step arises with new converging technological computer science advanced like massively distributed and cloud computing, Internet-of-Things (IoT), revisited artificial intelligence and big data, mobility with variety of computer systems and sensors, new user customs and mobile applications. We face a new interconnected world, the production systems are not only connected to the management systems (ERP) but also opened to the client side (B2C) and the suppliers side (B2B) including supply management, manufacturing, client relationship... Cyber-Physical Production Systems (CPPSs) consist of autonomous

André, Pascal
AeLoS Team LS2N - University of Nantes , e-mail: `pascal.andre@univ-nantes.fr`

Cardin, Olivier
PSI Team, LS2N - University of Nantes, e-mail: `olivier.cardin@univ-nantes.fr`

and cooperative elements and subsystems that are connected based on the context within and across all levels of production, from processes through machines up to production and logistics networks [14] Business processes can be connected to manufacturing processes. Cyber manufacturing is transformative concept that involves the translation of data from interconnected systems into predictive and prescriptive operations to achieve resilient performance [10].

The glue paradigm between processes and systems is the notion of **service**, which apogee is the cloud stack XaaS and the Service Oriented Architectures (SOA). Services are also somewhere compatible with (high level) business processes in enterprise architecture frameworks. Of course the definition of service varies a lot from one context to another. At low levels a service can be a script, a procedure, an operation or a method, depending on the implementation language. At intermediate level a service is a kind of process the coordinates other service (through service composition and orchestration) to achieve a specific or independent goal. At high level a service can be interpreted as a process that monopolize shared resources (and maybe people) to fulfil all or part of a business or production process. The coordination and control of such complex system by the way of actors or services requires methods and techniques to design, verify and deploy the services, possibly on the fly making service engineering an unavoidable approach to develop new generation cyber-manufacturing systems. This is part of requirements of CPS mentioned by Wang et al. in the category 'design methodology' *"Research challenges include development of techniques for efficiently integrating or relating multiple models/viewpoints/data sets, CPS design methodology for trustworthy end-to-end services including adaptive and autonomous systems,and platforms for safe and secure CPS design that underpin design methodology, facilitating integration and establishing desired system level properties."* [22]. As mentioned by Bauer et al. the traditional automation pyramid is dissolving and manufacturing IT is moving towards service-orientation and app-orientation [7]. As an example, in cloud manufacturing, SOA was identified to meet the requirements of all higher level manufacturing CPS layers due to the reduced time constraints present [16].

However service engineering is still a craft activity at the implementation level [19, 15]. Two main levers are still required to go further. First, we need service models that can fit to various semantics and various granularity levels. Indeed, the concept of cyber-physical production system (CPPS) covers many classes of (physical) systems, from the manufacturing workshop to the power distribution network. Taking the example of Holonic Manufacturing Systems, the control of such systems is often recursive, if not fractal, in order to aggregate the available resources and enable a heterarchic control architecture. Therefore, services that might be used at various levels of the architecture need to fit various granularity and the portability of services between different applications with their own semantic requires an adaptability of the services to be effective. Second, we need analysis tools to check the service model properties on various aspects (structure, dynamics, functional and non-functional), and model transformations to compute new models or to generate code in the spirit of model driven engineering (MDE).

In this position paper, we advocate a service based component model with embedded contracts that would be helpful to reach this goal. This model is abstract, in the sense of Service Oriented Architectures (SOA), to capture various semantics. Applying the proposed methodology with early verification from formal models reinforces confidence in the services early in the development process: they are corrected, they embed evaluated test data, and contracts can be reinforced. That helps to correct software as soon as possible in the process and allows to apply then advanced development techniques such as agile ones (thanks to the qualified test data we constructed) or Design-by-Contract (thanks to contract we reinforced).

We present the general process in section 2, the service models in section 3, the service contract in section 4 and the way to supply them, the service implementation, by the way of model transformation in section 5. Section 6 illustrates the approach on a small part of a vehicule control system. Some related references are discussed in section 7. In conclusion, we draw some perspective for manufacturing.

2 A development process for trusted services

We assume a general presentation, which is not specialized to manufacturing systems, where services denote *software services*, whatever implementation links to physical systems. Service based systems, as well as component based systems, are realized according to a composition principle (maybe several composition operators) such as high level services are hierarchically composed of lower level services where the low level services (the leafs) are atomic. We assume that (software) *components* are containers of related services ; the reason for grouping services in a component can be multiple: same provider, same business, same time, same job, same deployment node...

The design activities associated to composition are (1) to define the goal and properties (functional and non-functional), (2) to find the adequate services, (3) to define a service orchestration and (4) to check that the composition is correct and consistent with the service goal and properties and delivers a trusted composite service. Step (2) and (3) can be manual or automatic, depending on the repository management. The implementation activities are transformations of design models to the target code.

In order to build trusted services we need to specify formal models, then to verify them before implementation and storage on the shelf. This development sketch is illustrated by Figure 1.

The principle of component or *service on the shelf* is not yet accessible in practice and software designers cannot pick up services and compose them the way we can design electronic devices or mechanical assemblies or building constructions. Furthermore automating these activities remains challenging. In practice again,the service market is hardly available at design time and is often implemented by simple API at the implementation level, the service interfaces are poorly defined, often only a signature and a comment, so that searching candidate service require human

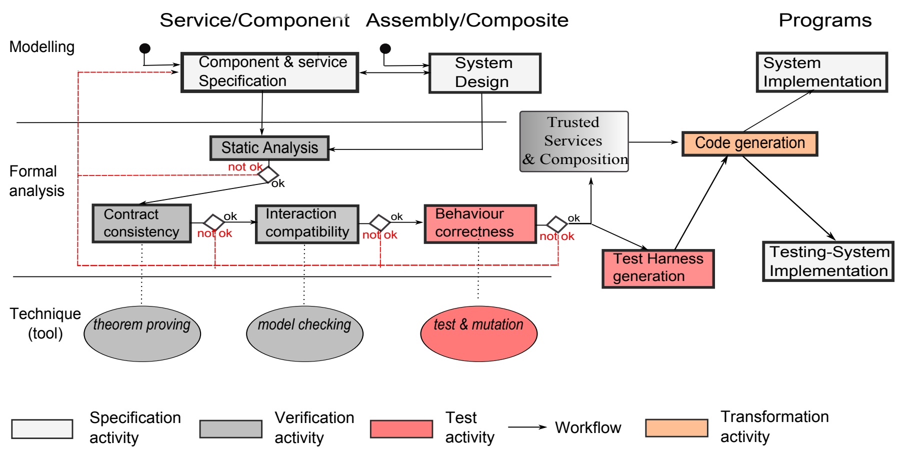

Fig. 1 Applied Specification, Verification and Implementation process

actions and understanding. *We will see section 3 a service model with rich interfaces to express service contracts*. Also services are merely final services, considered as a whole. This means that all its requirements are already given by some libraries or other services, *e.g.* we cannot change the providers of an intermediate service except if it belongs to us. *We will see section 3 a service model where the requirements are specified by service contracts, the notion of service contract will be developed in 4*. There is also a gap between the service paradigm, well-defined at the architecture level and its representation at the implementation level, where the notion of service does not exist: a service is implemented by XML descriptions and programming statements (java class for example), if service (or component) oriented programming would exist, the traceability links would be explicit. *We will see section 5 the basis for service implementation that preserve the service structure and traceability.*.

3 Service Modelling

In Service-based Component (SbC) models, a functionality is implemented by the services provided by some components. Provided services are not necessarily atomic calls and may possess a complex behaviour, in which other services might be needed (called). These needs are either satisfied internally by other services of the same component, or specified as required services in the component's interface. The required services can then be bound to provided services from other components, which might also require others, and so on. A provided service needs all its direct and indirect dependencies satisfied in order to be available for use. The process of providing trusted service on the shelf or designing is based on a triple: service model, property model and verification model.

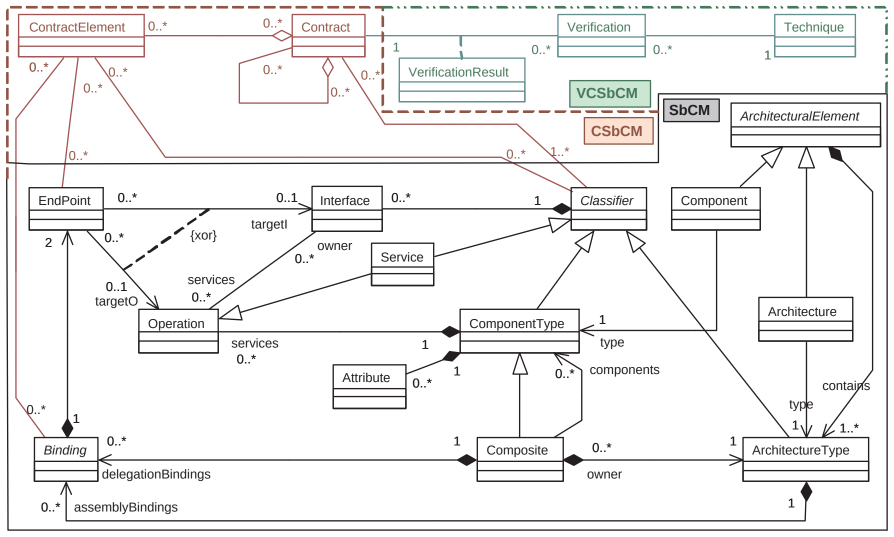

Fig. 2 Abstract Service based Component Metamodel, contracts and verifications

In the service based component model *SbCM* of Figure 2, the components (types) are characterised by an abstract state and services. Components are assembled on their access points (*EndPoint*) which can be ports, interfaces or services according to the concrete languages (SCA, Sofa, Fractal, Kmelia). A service specification may vary from the simple signature to the detailed description of dynamic behaviour with communications and service composition. An assembly (also called architecture) binds components, possibly using other assemblies, upon a client-ship relation (classical use of client/server contract). The connector paradigm is reduced to *binding* because it is usually its representation in component implementations. A composite encapsulates assemblies, and thereafter components, upon an inclusion relation (parent/child contract) that may promote child observable features (state, services). The involved features cover structural, functional and dynamic aspects. The service behaviour and the communications are described, depending on the specification language, by control structures, regular expressions, process algebra, state machines, ...

4 Service Contracts and Verification

We separate the specification and verification concerns. As illustrated by Figure 2, the *SbCM* meta-model is enriched with a contract layer (*CSbCM*) and a verification layer (*VCSbCM*). Layering enables to disconnect system models from properties to check (the contracts) and tools to verify. The layers enable to target verification on specific verification tools.

The contract layer is used to establish a service relation between *classifiers* (*e.g.* services, components), A classifier is a kind of black-box encapsulating an implementation. A contract is more than the assertions of Design by contract when including the composition of services, service interactions, quality of service [1].

According to [13], *a Trusted Component is a reusable software element possessing specified and guaranteed property qualities*. The notion of contract is helpful to model various kind of correctness properties. But it should be made precise and extended to cope with the expressiveness of the SbC models. The properties, *e.g. interoperability*, are classified at different contract levels ?

1. *Static*: the compatibility of interface signatures (names and types); does a service or component give enough information about its interface(s) in order to be (re)usable by others? The service call should respect the service signature. The signature matching between the involved services of component interfaces covers at least name resolution, visibility rules, typing and subtyping rules.
2. *Architectural*: the availability of the required components and services, the correctness of the linked service interfaces; Assuming that services can be composed from other (sub)services, connecting services is possible only if their structures are compatible (but not necessary identical).
3. *Functional or computational*: do the services do what they must do? These correctness properties may be checked both on individual service in regard to their container component and on the compositions. This third level deals with *service compliance*. If the services use a Hoare-like specification, post-conditions relate to their pre-conditions. The caller pre-condition is stronger than the called one. The called post-condition is stronger than the caller's one. Each part involved in the assembly should fulfil its counterpart of the contract.
4. *Behavioural or interactions*: the correct interaction between two or more services which are combined. he *behavioural consistency* property states that the execution of the service actions does not lead to inconsistent states (such as deadlock). The properties depends on the interaction model features: sequential vs. concurrent, call vs. synchronisations, synchronous vs asynchronous, pair vs. multipart communication, shared data, atomic/structured actions...
5. *Quality of service*: the non-functional requirements (time, size...) are fulfilled. For example, several services can be candidate but some are more efficient or more secure or more available than others.

In order to cope with different meaning and different context, we introduce the notion of **multi-level contract** in [1] where a contract is defined at different structure levels (service, component, assembly, composition) according to different expected requirement levels: syntax compatibility like CORBA IDL, structural compatibility, functional compatibility, behavioural compatibility and QoS compatibility. This vision of contracts provides a convenient framework to master both the incremental construction of SbC and the verification of multi-aspect properties by combined techniques. Also it provides a foundation for searching service in libraries (on the shelf).

The verification layer is based on a triple <*property, rule, technique*>. There is no wide-spectrum formal languages that enable the verification of all the properties in once. Usually, a modelling language is coupled with a verification technique and dedicated tools. *e.g.* the B notation is supported by the Atelier-B or Rodin theorem provers, LOTOS is supported by the CADP model checker. To target various kind of properties, a full formal analysis requires *model transformations* to target the adequate tools for the kind of properties to check. In the case of the process of Figure 1, the functional contract is checked using the theorem prover; the interaction contract is model checked and the conformance of the behaviour against the contract is controlled using test.Note that model testing uses the same MDE facilities to generate code for test harnesses than the one of service application models.

5 Service Implementation

Model Driven Engineering (MDE) emphasizes the use of models and meta-models to improve the software productivity and some aspects of the software quality such as maintainability or interoperability. According to the principles of Model Driven Engineering [8] one can transform and refine service models to executable programs by plunging in a technical domain (a framework) that preserve the service structure and traceability.

Figure 3 illustrates the mapping between a platform independent model and a platform specific level. This can result from a sequence of transformation, not necessary one step transformation. For example, one can target a specific implementation framework model, for example REST or SOAP web service model but high level models like WSDL or BPEL can also be intermediate models. This depends on the technical architecture but also the available model transformation.

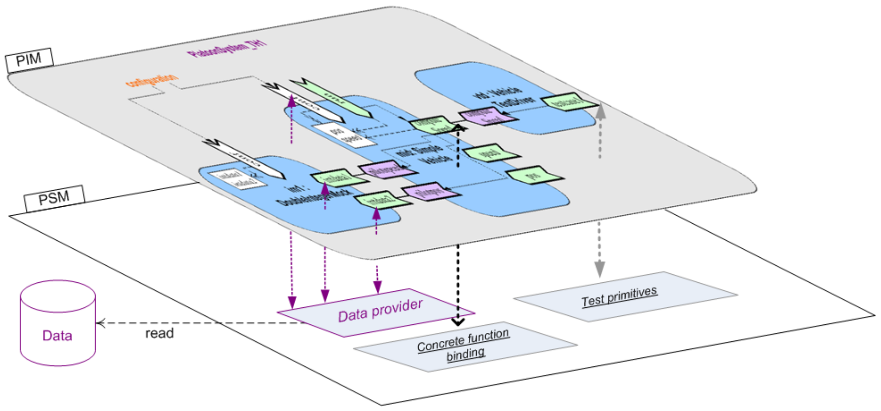

Fig. 3 Model concrete Data and Function Mapping

Some primitive functions of the component model are kept abstract at the model level and have to be mapped to concrete ones. This verification step checks if these mappings are well-defined and consistent.

The link with physical devices of CPS can be achieved in the same way.

6 Experimentations

This section will briefly introduce the experimentation context, an illustrative example and will include various references on previous works and experimentations.

We experimented the approach with a simplified version of a *platoon of vehicles* case study, which can share similarities with AGV (Automated Guided Vehicle), where the embedded application should ensure safety properties such as avoiding collisions or not losing a vehicle. The vehicles and the driver are components which interact to know their position and speed in order to control their move. We consider here only the speed and the position (X axis only) of the vehicles. The vehicles are designed to follow their predecessor (which they consider to be their pilot) except the first one which follows a component taking the role of the driver. The driver is assumed to be a special kind of vehicle that controls its own values according to a target position. Each running vehicle can compute its own speed by considering its current speed and position, its predecessor's position and speed and a safety distance with its predecessor.

The approach has been experimented with the COSTO, a CASE tool dedicated to the development of Service and Component Based Software Systems. The specification language is Kmelia a wide-spectrum language dedicated to the development of correct service based components [2]. To be short Kmelia allows to define service behaviours with extended state machines using an action language which includes synchronous communication on channels. Components contain several services and several components can be assembled using client-server links. A composite component encapsulate assemblies in which services of the subcomponents can be promoted at the composite level.

Figure 4 shows a small architecture composed of a *driver* and three *vehicle* components. We use the SCA notation [17] to make explicit the component's interfaces with provided and requires services (called references in SCA). Each component has a configuration service conf (used when instantiating the component), a main service run, which is launch automatically after configuration (autorun). The run and conf services assign values to the vehicle's state. The run service activates the vehicle behaviour and services to give their position and speed, it's a loop that ends when the platoon reach its goal. The computeSpeed service reads the vehicle's state to compute the next speed. Other services like stop which interrupts a vehicle, have been omitted for simplicity.

In Kmelia the components, assemblies and compositions can be analysed according to various facets and levels, as detailed in [2]. The general verification approach is explained in [1]. It is instrumented by the COSTO tool, a set of Eclipse plugins

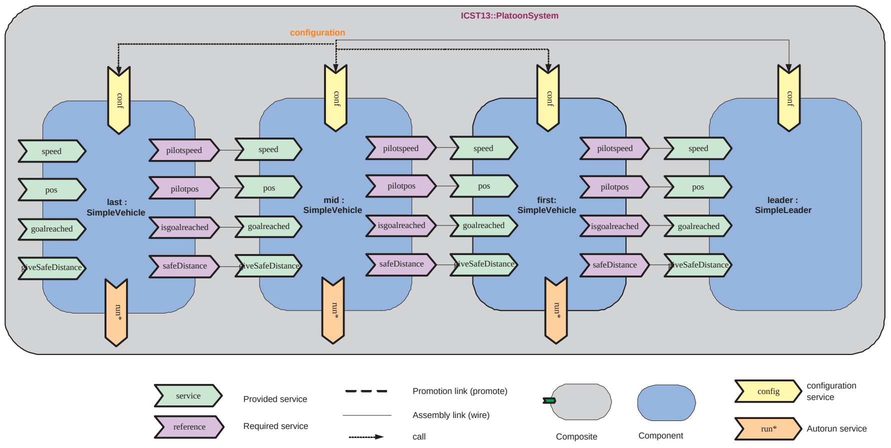

Fig. 4 Component model of the Platoon system

including an editor, a type checker, several analysis tools and exportations to external tools like MEC, CADP (Lotos), AtelierB and Rodin (Event-B), but also a Java code generator and an assistant to generate and execute test harnesses [5].

Previous works detail the verification techniques: the verification of functional contracts inside services/components [2] and in service composition [3] (theorem proving), the verification of behavioural compatibility [6] (model checking), the verification of consistency between service contracts and concrete behaviour [4] (model testing). For sake of space we redirect the reader to there references and we just enumerate in the remaining some kind of properties that can be checked on the platoon example (S-safety, L-liveness, F-Failure...):

S1 A vehicule moves in the bad direction because the given values are false or incorrect (not in the scope, not the same unit or representation...). This can lead to collision with other vehicules, people or environment.

S2 A vehicule leaves the platoon to be autonomous (with no driver) but does not reconnect its follower to its driver (dynamic service reconfiguration).

S3 A hacker intercepts the communications between two vehicules to send non-authorized information, the service contract should not be interruptible (broken subservice dependency).

L1 A vehicule waits because it cannot find its predecessor position or speed (service availability) or understand the given values (bad parameters or result type) or cannot read the values on time (bad QoS or synchronization).

L2 A vehicule cannot move because the physical situation is not consistent the virtual situation (twin failure), for the same reasons as above.

L3 The platoon is stopped due to bad configuration or reconfiguration ; *e.g.* vehicule is (directly or indirectly) driven by its own position and speed (infinite loop).

F1 An intermediate vehicule accelerates or decelerates suddenly due to a bad interpretation of its received values ; the followers will have the same jerky behaviour.

F2 The platoon turns round due to bad general configuration or reconfiguration ; at least this the case L3 where the position and speed are given with a delta of time which differs from case F1.

7 Related References

This section points out relevant information from related references

Bauer et al. [7] discuss current trends in manufacturing IT. They provide a good overview on manufacturing services and also apps, including concepts and implementation. Independent service vendors (ISV) are able to offer their services on the platform and users should be able to orchestrate services according to their needs in order to flexibly adapt to changing market conditions. We bring a new perspective on it by providing service engineering to put this need in practice.

As far as services can encapsulate or abstract not only software but physical behaviours (like digital twins do), orchestration refers to smart adaptable assembly systems as introduced by ElMaraghy [18] which creates the possibility of cross-fertilization of ideas.

Our proposal can take place in the integration part of the general frame for design, modelling, simulation and integration of cyber physical systems given by Hehenberger et al. [9] which includes mechatronic and internet of things (IoT). Abstraction and interoperability are key concepts for such systems. They denote often a lack of clearly specified and documented interactions and interfaces between the various disciplines and involved components and hence mutual understanding in communication is hindered. Our model can help in providing an abstract service layer for trust interoperability like an Architecture Description Language(ADL) which is more service oriented than the ADLs they mentioned. This takes place in the Service Enablement Layer of Monostori et al. [14] or in the 3rd level of the 5C architecture for implementation of Cyber-Physical System of Lee and al. [11]. Services can play roles in both *twin* models and the integration to higher level layers including production monitoring but also business processes or manufacturing simulation as in [20].

Liu et al. introduce a new paradigm of Cyber-Physical Manufacturing Cloud (CPMC) to bridge gaps among cloud computing, cyberphysical systems, and manufacturing [12]. They propose a four-layer service-oriented CPMC architecture where what we call atomic services could belong to layer 2 (resource virtualization) and our composed service would belong to layer 4 (core cloud) that handles API, security and publication. We think that our model of rich interfaces could be integrated in layer 4 with verification facilities. Morgan and O'Donnell investigated whether SOA could be integrated into a cyber-physical manufacturing execution system to enable cloud monitoring [16]. SOA was identified to not meet the deterministic requirements of low levels but meets those of all higher level manufacturing CPS layers due to the reduced time constraints present.

Our service meta-model covers a broad kind of services, it can be enriched with specialized meta-model like the UML4IoT to exploit IoT in cyber-physical manufacturing systems [21].

Compared with the work of Gamboa Quintanilla et al [18], our model is more rich and more open. Their concept of service corresponds to our notion of atomic service while their processes are atomic service assemblies. Building high level services would be possible at implementation only. In addition, they do not include the notion of required services, communications between services, dynamic behaviour... Those concepts that make possible rich interfaces and powerful automatic service search and composition.

8 Conclusion

The service orientation, enforced by cloud computing, become pregnant not only in software engineering but also in cyber physical systems, production systems and manufacturing control. The principles of component and service based computing have been set since two decades but the practice is far from those theories and the principle "on the shelf" ; services are usually composed by programmers. The Kmelia model to improves the service selection (search the adequate service) and the service orchestration (check the deep compatibility before assembling). Some features of Kmelia are really innovative *e.g.* rich interface and required service enables to really have modular descriptions of services which are fundamental for service interoperability or service substitution... Of course, the illustrated example is not directly related to CPS and manufacturing but we are convinced that the framework would be helpful to detect statically potential errors when assembling services. Open perspectives are (1) to extend existing CPS manufacturing service models to rich interfaces to enable searching and orchestration facilities, (2) to check statically or on the fly the provision of services and (3) to target various implementation model (SOA, WSDL...) or to generate implementation to deploy.

References

1. André, P., , Attiogbé, C., Mottu, J.M.: Combining techniques to verify service-based components. In: Proceedings of the International Workshop on domAin specific Model-based AppRoaches to vErificaTion and validaTiOn, AMARETTO@MODELSWARD 2017, Porto, Portugal, February 19-21, 2017., pp. 645–656 (2017)
2. André, P., Ardourel, G., Attiogbé, C., Lanoix, A.: Using assertions to enhance the correctness of kmelia components and their assemblies. Electr. Notes Theor. Comput. Sci. **263**, 5–30 (2010)
3. André, P., Ardourel, G., Messabihi, M.: Component service promotion: Contracts, mechanisms and safety. In: L.S. Barbosa, M. Lumpe (eds.) Formal Aspects of Component Software - 7th International Workshop, FACS 2010, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 6921, pp. 145–162. Springer (2010)

4. André, P., Mottu, J.M., Ardourel, G.: Building test harness from service-based component models. In: proceedings of the Workshop MoDeVVa (Models2013), pp. 11–20. Miami, USA (2013)
5. André, P., Mottu, J.M., Sunyé, G.: Costotest: A tool for building and running test harness for service-based component models (demo). In: Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, pp. 437–440. ACM, New York, NY, USA (2016)
6. Attiogbé, C., André, P., Ardourel, G.: Checking Component Composability. In: 5th International Symposium on Software Composition, SC'06, *LNCS*, vol. 4089. Springer (2006)
7. Bauer, D., Stock, D., Bauernhansl, T.: Movement towards service-orientation and app-orientation in manufacturing {IT}. Procedia {CIRP} **62**, 199 – 204 (2017), 10th {CIRP} Conference on Intelligent Computation in Manufacturing Engineering - {CIRP} {ICME} '16. [Edited by: Roberto Teti, Manager Editor: Doriana M. D'Addona]
8. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, 1st edn. Morgan & Claypool Publishers (2012)
9. Hehenberger, P., Vogel-Heuser, B., Bradley, D., Eynard, B., Tomiyama, T., Achiche, S.: Design, modelling, simulation and integration of cyber physical systems: Methods and applications. Computers in Industry **82**, 273 – 289 (2016)
10. Lee, J., Bagheri, B., Jin, C.: Introduction to cyber manufacturing. Manufacturing Letters **8**, 11 – 15 (2016)
11. Lee, J., Bagheri, B., Kao, H.A.: A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manufacturing Letters **3**, 18 – 23 (2015)
12. Liu, X.F., Shahriar, M.R., Sunny, S.N.A., Leu, M.C., Hu, L.: Cyber-physical manufacturing cloud: Architecture, virtualization, communication, and testbed. Journal of Manufacturing Systems **43, Part 2**, 352 – 364 (2017), high Performance Computing and Data Analytics for Cyber Manufacturing
13. Meyer, B.: The grand challenge of Trusted Components. In: ICSE '03: Proceedings of the 25th International Conference on Software Engineering, pp. 660–667. IEEE Computer Society, Washington, DC, USA (2003)
14. Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., Sauer, O., Schuh, G., Sihn, W., Ueda, K.: Cyber-physical systems in manufacturing. CIRP Annals - Manufacturing Technology **65**(2), 621 – 641 (2016)
15. Morariu, C., Morariu, O., Borangiu, T.: Customer order management in service oriented holonic manufacturing. Computers in Industry **64**(8), 1061 – 1072 (2013)
16. Morgan, J., O'Donnell, G.E.: The cyber physical implementation of cloud manufactuirng monitoring systems. Procedia CIRP **33**, 29 – 34 (2015), 9th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '14
17. OSOA: Service component architecture (sca): Sca assembly model v1.00 specifications. Specification Version 1.0, Open SOA Collaboration (2007)
18. Quintanilla, F.G., Cardin, O., L'Anton, A., Castagna, P.: A modeling framework for manufacturing services in service-oriented holonic manufacturing systems. Engineering Applications of Artificial Intelligence **55**, 26 – 36 (2016)
19. Rodrigues, N., Leitão, P., Oliveira, E.C.: Self-interested service-oriented agents based on trust and qos for dynamic reconfiguration. In: T. Borangiu, A. Thomas, D. Trentesaux (eds.) Service Orientation in Holonic and Multi-agent Manufacturing, *Studies in Computational Intelligence*, vol. 594, pp. 209–218. Springer (2015)
20. Tao, F., Cheng, J., Cheng, Y., Gu, S., Zheng, T., Yang, H.: Sdmsim: A manufacturing service supply–demand matching simulator under cloud environment. Robotics and Computer-Integrated Manufacturing **45**, 34 – 46 (2017), special Issue on Ubiquitous Manufacturing (UbiM)
21. Thramboulidis, K., Christoulakis, F.: Uml4iot—a uml-based approach to exploit iot in cyber-physical manufacturing systems. Computers in Industry **82**, 259 – 272 (2016)
22. Wang, L., Törngren, M., Onori, M.: Current status and advancement of cyber-physical systems in manufacturing. Journal of Manufacturing Systems **37**, 517 – 527 (2015)