۱-۱- مروری بر ریزهسته‌ها

در این بخش قصد داریم مروری بر مفهوم ریزهسته‌ها داشته باشیم. بعد از آن تاریخچه مختصری از ریزهسته‌ها را ارائه خواهیم داد، بدین ترتیب موضوعات مربوط به ریزهسته‌ها را که در گذر زمان مورد توجه قرار گرفته، مشاهده خواهیم کرد. در انتهای این بخش، برای کامل‌تر شدن بحث نگاهی هم به معماری ریزهسته‌ها خواهیم انداخت.

۱-۱-۱- تولد مفهوم ریزهسته‌ها

در سال ۱۹۷۹ دانشمندی به اسم Brinch Hansen در یکی از آزمایشگاه‌های IBM برای طراحی Hypervisor خود، ایده کمینه بودن طراحی به منظور کاهش پیچیدگی را ارائه داد [WIKA11]، و تا حد امکان این ایده خود را پیاده سازی کرد؛ وی معتقد بود که می توان این طراحی را حداکثر با سه مکانیزم پیاده سازی کرد. بدین ترتیب تنها چند مکانیزم ساده در هسته طراحی وجود خواهند داشت و سیاست‌های مختلف بر اساس این مکانیزم‌ها پیاده سازی خواهند شد. در همین سال‌ها و تقریباً توسط همین دانشمند بود که مفهوم جداسازی *سیاست از مکانیزم* برای کاهش پیچیدگی در *مهندسی سیستم‌ها* معرفی شد، این طراحی آقای Hansen، در حقیقت تجلی جداسازی مکانیزم از سیاست در زمان خود بوده است.

در همین سال‌ها ایده فناوری ریزهسته معرفی شد که می‌گفت: می‌توان هسته سیستم عامل‌ها را بقدری کوچک ساخت که تنها چند مکانیزم ساده در آن وجود داشته باشد، و تمام Functionality های سیستم در قالب برنامه‌های کاربردی (بصورت سرویس) می‌تواند توسعه داده شود.

از همان سال‌های اولیه مزایای ریزهسته‌ها برای طراحان سیستم و مهندسین نرم‌افزار آشکار بود؛ از مهمترین این مزایا، می‌توان به موارد زیر اشاره کرد [STAL00]:

- واسط‌های یکدست و خوب تعریف شده سرویس‌های مختلف،
- توسعه پذیری،
- قابلیت حمل،
- قابلیت اعتماد،

- حمایت از سیستم‌های توزیع شده،
- حمایت از سیستم عامل‌های شئ‌گرا.

سود این فناوری سرشار بود که محققین را برای تحقیق در زمینه ریزهسته‌ها ترغیب کرد؛ در ادامه به بررسی نتایج فعالیت دانشمندان مختلف خواهیم پرداخت.

۱-۱-۲- ریزهسته‌های نسل اول

در سال ۱۹۹۳ دو دانشمند به اسم Chen و Bershad در دانشگاه Carnegie Mellon شروع به تحقیق بر روی ریزهسته‌ها کردند [LIEB96]. نتایج کار آن‌ها منجر به توسعه ریزهسته‌های Spin و Mach شد؛ این سیستم عـامل‌ها با استفاده از کدهای موجود برای UNIX توسعه داده شده بودند. آن‌ها با آزمایش‌های بسیار دقیق بر روی این دو ریزهسته، متوجه شدند که ریزهسته‌ها در استفاده از حافظه نهان و TLB بسیار ناکارآمد عمل می‌کنند [LIEB96]. آن‌ها با ارائه چندین مقاله، ریزهسته‌ها را نا مناسب برای کارهای Real-time معرفی کرده و سیستم‌های تحت ریزهسته را از نظر کارایی *بسیار* ناکارآمد معرفی کردند. در این مقالات اشاره کردند که با یک ریزهسته کوچک ریزهسته عملا به یک کانال ارتباطی تبدیل می شود که IPC گلوگاه سیستم می گردد؛ بدین ترتیب برای انجام هر کاری باید از IPC استفاده شود، ولی از آنجایی که هزینه انجام هر عمل IPC در این ریزهسته‌ها به چندین هزار سیکل ساعت نیاز داشت، عملا ریزهسته‌ها با تمام منافعشان کنار گذاشته شدند. با انتشار این مقالات معتبر، پرونده ریزهسته‌ها تقریباً به مدت ۸ سال بسته شد. البته این دانشمندان در حین توسعه ریزهسته‌های خود مفاهیم بسیار ارزشمندی نظیر Pager خارجی را ابداع و معرفی کردند؛ در حقیقت آن‌ها نهایت ریز بودن یک ریز هسته را نشان دادند.

لازم است بدانیم سیستم عامل‌های Mach و Spin و سایر ریزهسته‌هایی که در این دوره توسعه داده شدند همگی جزو ریزهسته‌های نسل اول به شمار می‌آیند. این خانواده از ریزهسته‌ها به کارایی بسیار پایین مشهور هستند. بعدها همین دانشمندان ادعا کردند که می‌توان با برگرداندن تعدادی از سیاست‌ها به داخل هسته، می‌توان کارایی سیستم را افزایش داد. آن‌ها آزمایشات خود را بر روی ریز هسته Spin انجام دادند و مفهوم co-location در هسته[1] را معرفی کردند. آن‌ها با این کار سعی بر افزایش کارایی

[1] بدین معنی که می توان تعدادی از سرویس‌های سطح کاربر را در سطح هسته اجرا کرد؛ این سرویس‌ها از سوی برنامه نویسان برنامه‌های کاربردی تعریف می شوند و در زمان اجرا در حالت هسته اجرا می شوند.

این سیستم عامل‌ها داشتند، و حتی به یک کارایی جزئی دست یافتند. البته لازم است بدانیم ارائه مفهوم co-location در حقیقت دور زدن مسئله است و این دانشمندان به هیچ وجه متوجه علت اصلی تنزل کارایی نشدند. بعدها سیستم عامل Mach با قدری اصلاح به عرصه بازار راه پیدا کرد، بطوری که هسته سیستم عامل Mac OS X را با توسعه و تغییر ریزهسته Mach ساختند.

۱-۱-۳- ریزهسته‌های نسل دوم

بعد از یک بازه زمانی نسبتاً بلند، یک دانشمند آلمانی به اسم Jochen Liedtke در مرکز تحقیقات انفورماتیک آلمان (GMD)، شروع به مطالعه مقالات دو دانشمند قبلی کرد و ادعا کرد که ایراد کار دو دانشمند قبلی استفاده از هسته UNIX موجود می‌باشد. وی گفت که ایراد کار Chen و Bershad در این است *که آن‌ها مشکل را پیدا کردند ولی دنبال علت آن نگشتند* [LIED95]. در حقیقت آن‌ها از یک هسته از قبل توسعه داده شده استفاده کردند که در اصل برای این کار (ریزهسته بودن) طراحی نشده بود. او که (در عین حال که یک ریاضیدان بود) یک برنامه نویس بسیار با تجربه بود[۲]، شروع به نوشتن چهارمین سیستم عامل خود کرد (وی قبلاً سه سیستم عامل دیگر را نیز طراحی کرده بود که نام آخرین آن‌ها L3 بوده – لازم است بدانیم که این سیستم عامل‌ها ریزهسته نبودند)، و نام آن را L4 (یعنی چهارمین سیستم عامل Liedtke) گذاشت؛ این سیستم عامل یک ریزهسته بود که بدون استفاده از کدهای قبلی (نظیر UNIX) و با دقت فراوان بر روی کارایی، *مخصوص* ریزپردازنده 80486 توسعه داده شد. علی رغم اینکه ریزپردازنده 80486 یک ریزپردازنده فاقد کارایی قابل قبول می‌باشد (به دلیل طراحی نامناسب)، ریزهسته آقای Liedtke بسیار عالی عمل کرد؛ این ریزهسته برای انجام عمل IPC تنها به ۲۵۰ سیکل ساعت پردازنده نیاز داشت، این درحالی است که ریزهسته‌هایی نظیر Spin و Mach همین کار را با چند هزار سیکل ساعت انجام می‌دادند. در همین سال‌ها (البته قدری پیش‌تر) پروژه Exokernel در دانشگاه MIT شروع شد [TANE97]. آن‌ها هم توانسته بودند یک ریزهسته دارای کارایی نزدیک به کارایی L4 طراحی کنند، ولی این ریزهسته برای ریزپردازنده MIPS طراحی شده بود، و شاید به همین دلیل هم بوده که به اندازه L4 مورد توجه قرار نگرفت (این امری طبیعی است که طراحی یک ریزهسته با کارایی بالا بر روی یک ریزپردازنده با کارایی پایین جذاب‌تر باشد). این جهش در کارایی و سرعت موجب زنده شدن مفهوم ریزهسته شد، و از آن تاریخ به بعد

[۲] بر گرفته از ارائه پروفسور Heiser در کلاس سیستم‌عامل‌های پیشرفته در دانشگاه New South Wales.

همه‌ی افراد در جامعه محققین عرصه سیستم عامل، این درس را آموختند که *کارایی ریز هسته شدیداً وابسته به نحوه طراحی آن است*[۳]؛ در افراطی‌ترین حالت، باید برای هر ریزپردازنده بخصوص یک ریزهسته مخصوص خودش از ابتدا طراحی کرد تا اینکه بتوان به دقت کارایی ریزهسته را مورد بررسی قرار داد، این ویژگی ریز هسته‌ها را بسیار غیرقابل حمل (به دلیل وابستگی به یک ماشین خاص) ساخت.

سیستم عامل‌های L4، Exokernel، و تمام ریزهسته‌هـــایی که در این دوره طراحی شدند را ریزهسته‌های نسل دوم می‌نامند. توجه اصلی در این ریزهسته‌ها، بر روی کارایی بوده است. این دوره، دوره تولد مجدد مفهوم ریزهسته‌ها بود، در این دوره ریزهسته‌ها به عرصه بازار هم راه یافتند. چرا که کارایی آن‌ها با در نظر گرفتن منافع سرشارشان قابل قبول بود. بدین ترتیب سیستم عامل‌هایی نظیر QNX که هسته آن ها *ریزهسته محض* (ریزهسته محض یعنی یک ریزهسته با حداقل مکانیزم‌های پیاده سازی شده در هسته) می‌باشد متولد شدند.

۱-۱-۴- ریزهسته‌های نسل سوم

بعد از گذشت چندین سال از زمان معرفی L4، پروژه‌های زیادی برای توسعه این سیستم عامل معرفی شد؛ برای مثال می‌توان به نسخه‌های مختلف ریزهسته L4 از قبیل Pistachio و Hazelnut اشاره کرد. دانشگاه New South Wales در کشور استرالیا مهد ریزهسته‌های نسل سوم می‌باشد. در این دانشگاه علاوه بر نسخه‌های متعدد ریزهسته L4، پروژه مهم ریزهسته seL4، به عنوان یک L4 دارای مکانیزم‌های امنیتی جاسازی شده معرفی شد [WIKB11].

در حقیقت امروزه ریزهسته‌های نسل سوم را با seL4 می‌شناسند. پروژه ریزهسته seL4، تقریباً از سال ۲۰۰٤ شروع شد. یک تیم برنامه نویسی متشکل از چندین برنامه نویس، در یک پروژه به نام NICTA در دانشگاه New South Wales اقدام به توسعه seL4 کردند. یکی از افراد شاخص در توسعه این سیستم عامل پروفسور Gornet Heiser می‌باشد. توجه اصلی در توسعه ریزهسته‌های نسل سوم، *علی‌الخصوص* seL4، بیشتر بر روی مکانیزم‌های امنیتی است. در سیستم عامل seL4، برای ساخت یک Trusted Computing Base درست، اقدام به طراحی و *صحت‌سنجی صوری* (Formal

[۳] لازم است بدانید که تا به امروز هیچ ریزهسته‌ایی از نظر کارایی و سرعت به پای L4 نرسیده است. کارایی و سرعت این ریزهسته به عنوان یک معیار در بین محققین عرصه سیستم عامل‌ها، بکار می‌رود.

Verification) کردند. در حقیقت seL4 اولین ریزهسته همه‌منظوره صحت سنجی شده بصورت صوری در سطح دنیا می‌باشد. افراد پروژه NICTA در حدود پنج سال بر روی این پروژه طاقت‌فرسا کار کردند[٤].

پروژه‌های مشابهی در آزمایشگاه Open Kernel Labs از قبیل OKL4، در سال‌های اخیر توسعه داده شد. با وجود اینکه seL4 در حقیقت الهام گرفته شده از OKL4 است، اما seL4 کاملا بصورت صوری صحت‌سنجی شده است؛ این در حالی است که OKL4 در این حد صحت‌سنجی نشده است.

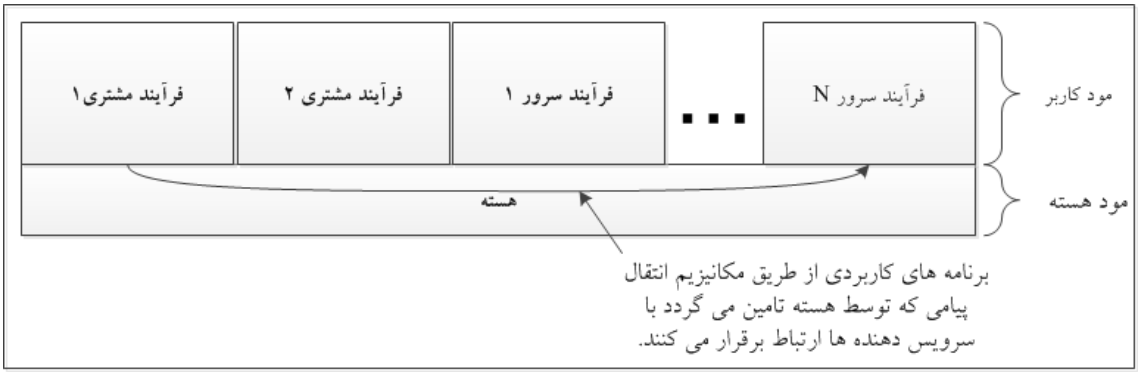
یکی از ریزهسته‌های مهمی که می‌توان آن را در زمره ریزهسته‌های نسل سوم دست بندی کرد، ریزهسته Chorus می‌باشد که در آزمایشگاه INRIA در کشور فرانسه توسعه داده شد. این ریزهسته یک ریزهسته خاص منظوره است، و برای استفاده در سیستم‌های توزیع شده در نظر گرفته شده است. ویژگی مهم این ریزهسته که زیاد مورد توجه ما است، شیوه طراحی آن است. این ریزهسته به *روش شیءگرا* طراحی شده است.

۱-۱-۵- معماری ریزهسته‌ها

در این بخش به منظور تکمیل‌تر شدن بخث، قصد داریم مرور کوتاهی بر معماری ریزهسته‌ها داشته باشیم. تلاشی که اخیراً در عرصه طراحی و پیاده‌سازی سیستم‌عامل‌ها صورت می‌گیرد این است که بسیاری از بخش‌های کد سیستم‌عامل را از هسته به بیرون از آن منتقل کنند؛ بدین ترتیب هسته سیستم‌عامل در کمترین حد خود قرار خواهد گرفت [TANE97]. وقتی که سرویس‌های سیستم‌عامل بصورت برنامه‌های خارجی پیاده‌سازی می‌شوند، برنامه‌های کاربردی باید با استفاده از مکانیزم انتقال پیام بین فرآیندها با این سرویس‌ها ارتباط برقرار کنند و درخواست‌های خود را به آن‌ها بفرستد. بسیاری از محققین عرصه سیستم‌های عامل مانند Liedtke معتقدند که می‌توان بصورت افراطی حداکثر تعداد سرویس‌های سیستم‌عامل (این سرویس‌ها حتی شامل مکانیزم صفحه بندی نیز می‌باشد) را به بیرون از هسته منتقل کرد، و در هسته صرفاً چندین مکانیزم پایه باقی بماند [LIEB96] و

[٤] البته انجام پروژه seL4 به این دلیل قدری بطول انجامیده است که تیم NICTA، مجبور بود که بسیاری از مسائل را برای اولین بار تصمیم‌گیری کند (با مطالعه مستندات پروژه)؛ لزوم این تصمیم‌گیری‌ها به این خاطر بوده که آن‌ها اولین سیستم‌عامل صحت سنجی شده را تولید می‌کردند. ما، مراحلی که seL4 از بدو تولد تا به حال طی کرده است مطاله کرده‌ایم و دیدیم که تیم NICTA بارها در تصمیم‌گیری‌های خود با شکست مواجه شدند و راه‌های دیگری را امتحان کردند. ما اکنون تجارب ارزشمند این تیم، و همچنین ابزارهای لازم برای کار را در اختیار داریم و می‌توانیم در مدت زمان بسیار کمتری پروژه را به پایان برسانیم.

[LIEC96]؛ این محققین معتقدند که انتقال سرویس‌ها به بیرون از هسته می‌تواند سربار قابل قبول و مقرون به صرفه‌ایی داشته باشد. این در حالی است که برخی از محققین و نویسندگان، معماری ریزهسته را قطعاً ناکارآمد به حساب می‌آورند و راه بهبودی را برای آن نمی‌بینند و یا اینکه معتقد هستند که موضوع کارایی ریزهسته‌ها هنوز در بین محققین عرصه سیستم‌های عامل حل و فصل نشده است [ELMA10] و [STAL00]. در شکل ۱ تصویری از دیاگرام بلوکی بخش‌های مختلف سیستمی فرضی با معماری ریزهسته را که حالت خاصی از معماری مشتری-خدمتگزار است می‌توان مشاهده کرد.



شکل ۱ - معماری Client-Server که معماری ریزهسته حالت خاصی از این معماری می‌باشد. برگرفته از [TANE97].

۱-۲- مروری بر طراحی شئ‌گرا

روش طراحی شئ‌گرا از جمله روش‌های نوین در مهندسی نرم‌افزار می‌باشد. طراحی شئ‌گرا به علت منافعی که دارد مورد توجه طراحان سیستم‌عامل، بخصوص تیم طراحی سیستم‌عامل ویندوز قرار گرفته است. در حالت کلی می‌توان مزایای طراحی شئ‌گرا را به صورت زیر خلاصه کرد:

- افزایش پایداری سیستم به علت تقسیم آن به ریزسیستم‌های مستقل و جدا که از طریق واسط خوب تعریف شده با هم در ارتباط‌اند، و همچنین عدم اطلاع اشیاء از نحوه پیاده‌سازی همدیگر
- مدیریت بهتر پیچیدگی ذاتی در مسئله
- کاهش تلاش لازم برای توسعه از طریق استفاده مجدد
- تولید سیستم‌هایی با قابلیت توسعه و نگهداری بالا.

تجارب و روش‌های مهندسی نرم‌افزار را می‌توان در طراحی سیستم‌های عامل نیز بکار برد، بدین ترتیب می‌توان بخش‌های مختلف سیستم‌عامل را متشکل از اشیائی دانست که با هم و از طرایق واسط‌هایشان در ارتباط هستند [ELMA10] و [STAL00]. تعدادی از ریزهسته‌های نسل سوم به روش شئ‌گرا طراحی شده‌اند. درســـت است که ریزهسته‌ها ذاتاً کوچک و دارای پیچیدگی کمتری می‌باشند، ولی ساده‌تر کردن همین مـسئلـه سـاده نیز می‌تواند منجر به تولید سیستم‌هایی امن‌تر و بی‌نقص‌تر شود.

با وجود اینکه عده‌ایی از توسعه دهندگان سیستم عامل معتقد هستند که روش شئ‌گرا در طراحی سیستم عامل ضروری نیست، منافع و مزایای بی‌شمار این روش برای همه فعالان عرصه مهندسی نرم‌افزار موضوعی اثبات شده می‌باشد. بـه نظر من با استفاده از روش طـراحی شئ‌گرا می‌توان سیستم‌هایی با پیچیدگی کمتر تولید کرد، بدین ترتیب می‌توان این سیستم‌ها را راحت‌تر صحت‌سنجی کرد.

۱-۳- کارهای مشابه

از جمله کارهای مشابه با پروژه ما، می‌توان به پروژه سیستم عامل Chorus اشاره کرد که به روش شئ‌گرا طراحی شده و بصورت خاص در سیستم‌های توزیع شده و برای حمایت از شئ‌گرائی مورد استفاده قرار می‌گیرد [CHOR91]. البته باید بدانیم این سیستم عامل یک ریزهسته محض نبوده و بصورت صوری هم صحت سنجی نشده است، بنابراین به میزان زیادی با پروژه ما متفاوت است و شباهت این پروژه صرفاً به دلیل شیوه طراحی شئ‌گرا، آن هم در مراحل اولیه می‌باشد.

پروژه دیگری که از نظر صحت سنجی بسیار شبیه به پروژه ما است، پروژه seL4 می‌باشد. این پروژه منجر به تولید یک ریـزهسته محض در اواخر سال ۲۰۱۰ میلادی شد که بطور کامل و بصورت صوری صحت سنجی شده بود و از سیستم‌های شئ‌گرا نیز حمایت می‌کرد. با مراجعه به مستندات این پروژه در می‌یابیم که سیستم عامل seL4 بصورت شئ‌گرا طراحی نشده است و در ضمن روش طراحی آن بگونه‌ایی است که مدت زیادی برای تولید یک ریزهسته دیگر به این روش نیاز می‌باشد [DRDB10] (یعنی اگر بخواهیم با روش توسعه استفاده شده در پروژه seL4 ریزهسته‌ایی را طراحی کنیم، قطعاً زمان زیادی را خواهد طلبید)، به عبارت دیگر سرعت تولید (یا Productivity) در این

روش پایین می‌باشد و در بسیاری از موارد کارهای مدلسازی فرمال بصورت دستی انجام می‌شود که این کار موجب می‌شود که کار مدلسازی علی‌رغم نیاز به نیروهای متخصص بسیار با تجربه در حوزه روش‌های فرمال، نیازمند صرف وقت زیاد است. همچنین در این روش احتمال به وجود آمدن فاصله بین جهان واقع و مدل زیاد می‌باشد. پروژه‌های مشابهی هم در آزمایشگاه OKLab انجام شده است، اما هیچ یک از آنها بطور کامل و بصورت صوری صحت سنجی نشده‌اند و نیز روش طراحی آن‌ها بگونه‌ایی نبوده است که طراحی ریزهسته‌های صحت سنجی شده را سرعت بخشد.

طبق مطالعات و دانش ما از حوزه طراحی و پیاده‌سازی سیستم‌های عامل تا کنون هیچ ریزهسته‌ایی با چنین روش مهندسی که در این پروژه معرفی شده طراحی و پیاده‌سازی نشده است. در این روش با استفاده از زبان مدلسازی UML می‌توان سیستم خود را مدل کرده و نیز با استفاده از زبان OCL می‌توان مدل خود را دقیق‌تر و خالی از ابهام کرد؛ سپس با استفاده از روش‌ها و راهنمایی‌هایی که ذکر شده می‌توان مدل UML خود را به یک مدل فرمال B تبدیل کرد. از آنجایی که ابزارهای فراوانی برای صحت سنجی و پالایش مدل‌های B وجود دارد می‌توان با صرف وقت بسیار کمتری اقدام به صحت سنجی و پیاده‌سازی سیستم خود نمائیم.

۱-۴- نتیجه گیری

همانطور که در بخش‌های قبل ذکر شد، و با قدری جستجو و تحقیق هم به آن خواهیم رسید، پروژه سیستم‌عاملی که اکنون توسعه آن را به پایان رسانیده‌ایم از لحاظ شیوه طراحی و مهندسی کاری جدید در عرصه طراحی و پیاده‌سازی سیستم‌های عامل می‌باشد.

این پروژه ریزهسته‌ایی می‌باشد که بصورت شئ‌گرا طراحی شده است. از دید مهندسی نرم‌افزار معماری ریزهسته برتر از معماری‌های دیگر می‌باشد چرا که در این معماری بخش‌های مختلف سیستم جدا از هم می‌باشند، بنابراین Loose Coupling در این سیستم در بالاترین حد ممکن خود قرار دارد. البته این جداسازی بخش‌های سیستم قطعاً سربار ارتباطات بین آن‌ها را هم به دنبال خواهد داشت، ولی با یک طراحی درست می‌توانیم این سربار را به کمترین حد خود برسانیم و از مزایای معماری ریزهسته استفاده کنیم. مدلسازی شئ‌گرا در عرصه مهندسی نرم‌افزار برای مدیریت پیچیدگی در توسعه، پشتیبانی، و تغییر سیستم‌های نرم‌افزاری استفاده می‌شود. استفاده از طراحی شئ‌گرا برای

طراحی یک ریزهسته موجب می‌شود که از ملزومات معماری ریزهسته (برای مثال Loose Coupling تا حد امکان) برای یک طراحی شئ‌گرای خوب و استادانه استفاده کنیم.

در ادامه روشی را برای تبدیل مدل‌های شئ‌گرا به مدل‌های B ارائه خواهیم داد، بدین ترتیب قادر خواهیم بود مدل‌های خود را صحت سنجی کنیم. بعد از صحت سنجی با استفاده از متد B قادر خواهیم بود که مدل خود را پالایش و نهایتاً پیاده‌سازی کنیم؛ خوشبختانه متد B بگونه‌ایی است که ابزارهای زیادی برای اتوماتیک کردن این فرآیند وجود دارد؛ در ادامه خواهیم دید که چگونه می‌توان با استفاده از ابزار AtelierB فرآیند استفاده از متد B را سریع‌تر انجام دهیم.